cajthaml

SSPŠ | 2025/2026

Webové aplikace

Skripta

Autor
Ing. Matěj Cajthaml

Verze
14. 12. 2025

Obsah

Seznam zkratek

API	Application Programming Interface
ARIA	Accessible Rich Internet Applications
ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
BEM	Block Element Modifier
CC	Creative Commons
CMS	Content Management System
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DNS	Domain Name System
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
EU	European Union
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
GIF	Graphics Interchange Format
HSL	Hue Saturation Lightness
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifikátor
IDE	Integrated Development Environment
IDN	Internationalized Domain Names
IP	Internet Protocol
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
JPG	Joint Photographic Experts Group
JS	JavaScript
JSON-LD	JavaScript Object Notation for Linked Data
LAN	Local Area Network

LTR	Left to Right
MDN	Mozilla Developer Network
MIME	Multipurpose Internet Mail Extensions
MS	Microsoft
MVOP	Maturitní volitelný odborný předmět
MVP	Minimum Viable Product
PAN	Personal Area Network
PC	Personal Computer
PNG	Portable Network Graphics
PSI	Počítačové sítě
PX	Pixely
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RTL	Right to Left
SMS	Short Message Service
SPA	Single Page Application
SSH	Secure Shell
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TLD	Top Level Domain
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VPN	Virtual Private Network
WAN	Wide Area Network
WBA	Webové aplikace
WEBP	Web Picture format
WWW	World Wide Web
WebP	Web Picture
XML	eXtensible Markup Language
XSS	Cross-Site Scripting

ČVUT České vysoké učení technické v Praze

Předmluva

1.1 Úvod

Webové aplikace nás obklopují na každém kroku. Pro mě osobně představují jednu z nejzajímavějších oblastí programování a informačních technologií obecně.

Tento text je určen především studentům 2. ročníku Smíchovské střední průmyslové školy a gymnázia, ale své využití najdou i u dalších ročníků při opakování. Cílem je předat základní znalosti o webových aplikacích a jejich vývoji – a to v podobě, která vás nejen poučí, ale zároveň snad i nadchne natolik, že se jim budete chtít věnovat i do budoucna.

Vývoj webových aplikací je dynamická a neustále se rozvíjející disciplína. Každým rokem přicházejí nové technologie – programovací jazyky, frameworky, knihovny či nástroje – které posouvají možnosti jejich tvorby stále dál. Stejně tak i tato skripta budou postupně rozšiřována a aktualizována, aby držela krok s vývojem oboru.

Pevně věřím, že pro vás budou přínosná a užitečná na cestě za hlubším porozuměním webovým aplikacím. Přeji vám mnoho radosti při jejich studiu.

1.2 Poděkování

Tato skripta vznikala v průběhu mnoha let výuky na SSPŠ. Část obsahu vychází z dřívějších materiálů, které jsem postupně tvořil a využíval při hodinách. Tyto texty prošly rukama stovek studentů, kteří svými připomínkami a upozorněním na chyby významně přispěli k jejich vylepšení. Všem jim patří velké poděkování za cennou zpětnou vazbu.

Velký dík si zaslouží také tým stojící za nástrojem Typst, bez něhož by tato podoba materiálu nemohla vzniknout.

Pokud během čtení narazíte na chybu, nepřesnost nebo budete mít návrh na zlepšení, budu rád, když vytvoříte issue na GitHubu.

1.3 Jak používat skripta

Skripta slouží jako podpůrný materiál k předmětu Webové aplikace a svým obsahem z velké části kopírují probíranou látku. Místy obsahují doplňující informace, které se během hodin nestihnou probrat, a naopak – některé části výuky se zde nemusí objevit. Je proto důležité brát je jako doplněk, nikoliv náhradu výuky.

Je třeba zdůraznit, že samotným čtením se programovat nenaučíte. Programování (a i tvorba webových stránek) je dovednost, která se osvojuje především praktickým cvičením. Proto doporučuji při čtení vždy zkoušet ukázky v editoru a experimentovat s kódem na vlastní pěst.

Pokročilejší příklady naleznete často také v prezentacích z hodin, které mohou sloužit jako rozšíření a inspirace.

1.4 Obsah

Obsah skript je místy zjednodušen tak, aby byl více přehledný pro obyčejného studenta střední školy. Pro „zcela správné pochopení“ je nutné navštívit zejména standardy W3C pro jednotlivé technologie, které jsou dostupné na w3.org.

Tyto skripta totiž nejsou vytvořena jako úplný a vyčerpávající zdroj informací, ale spíše jako úvodní průvodce a přehled základních konceptů. Pokud vidíte nějaké zjednodušení, nepřesnost, nejspíše je to z tohoto důvodu. Musím-li někde udělat kompromis mezi přesností a srozumitelností, vždy volím srozumitelnost a pokud je to možné, uvedu poznámku pod čarou.

Internet a web

2.1 Komunikace v síti

Počítačová síť je lokální propojení alespoň dvou zařízení, které spolu umí komunikovat. Počítačová síť může vlastně obsahovat další počítačové sítě a může vznikat jakýsi hierarchický systém. Tyto počítačové sítě komunikují různými protokoly a rozumí si. Probíhá tedy výměna informací a používají stejné technické prostředky.

V komunikaci rozdělujeme dva typy komunikací, a to peer to peer a client-server. Peer to peer je způsob komunikace, kdy jsou si obě dvě strany rovny a zároveň si nepřikazují. Client-server je způsob, kdy je jedna strana autoritativní a je cílena různými klienty. Server zpracovává příkazy od klientů a může je tedy i odmítat.

Internet je celosvětový systém navzájem propojených počítačových sítí. Propojení sítí je realizováno různými způsoby (např. drátově a bezdrátově), které tu nebudeme hloubkově probírat[1].

Internet používá nespočet služeb, jedná se například o:

- Webové stránky (např. ČT24, Google),
- Přenos videa a zvuku (např. YouTube, Spotify),
- Komunikace, e-maily, sociální sítě (např. Facebook, Instagram, WhatsApp, Gmail),
- Hry (např. Fortnite, Minecraft),
- A mnoho dalšího.

Často se setkáte s tím, že počítačové sítě jsou dále děleny do různých kategorií podle jejich velikosti. Mezi nejčastější patří:

- PAN (Personal Area Network) – osobní síť, která propojuje zařízení v bezprostřední blízkosti uživatele (např. Bluetooth),
- LAN (Local Area Network) – lokální síť, která propojuje zařízení v omezené geografické oblasti, jako je domácnost nebo kanceláře,
- WAN (Wide Area Network) – široká síť, která propojuje zařízení na velkých vzdálenostech, například mezi městy nebo státy.

Dále se taktéž rozdělují na sítě veřejné a soukromé. Veřejné sítě jsou přístupné komukoliv, zatímco soukromé sítě jsou omezeny na určitou skupinu uživatelů.

2.1.1 Historie internetu

Začátek internetu lze trasovat do různých bodů v historii. Každý odborník definuje tento začátek jinak, obecně je však i začátkem point-to-point internet při a po druhé světové válce, kdy potřebovali dvě místa komunikovat mezi sebou. Více je však uznávaná síť ARPANET, která vznikla v roce 1969. Při vzniku byly propojené čtyři vysoké školy. Postupem se k nim připojovali další místa a tím se síť zvětšovala až se stala základem dnešního internetu.

Již po třech letech (1972) po vzniku sítě bylo připojeno 70 zařízení. Tato síť je nyní součásti internetu, který je po celém světě. V roce 1992 se připojila první Česká instituce k internetu a to bylo České vysoké učení technické v Praze (ČVUT). Přesněji se připojila na uzel v Linci v Rakousku.

Počet zařízení na internetu se stále zvětšuje. Aktuální odhad je mezi 50 miliardami zařízení, které aktivně používají internet.

[1] Vše byste se měli dozvědět v předmětu PSI v 1. a 2. ročníku.

2.1.2 World Wide Web

Hypertextový dokument je dokument, který může být vázán na jiné dokumenty. Propojují se tzv. hyperlinky. Tento dokument často není lineární, ve slova smyslu, jako jsou obyčejné dokumenty, které můžete číst ze shora dolů. Tento dokument může být definován tak, že se něco může zobrazit jinde, než bylo definováno.

World Wide Web (dále jen WWW) je soustava propojených hypertextových dokumentů. Zjednodušeně se jedná o mapu propojených dokumentů, které lze navštěvovat pomocí webového prohlížeče. Používá se protokol HTTP (a HTTPS) ve formátu HTML. Pro adresaci ve WWW používá URL.

2.1.3 Domény

Doména je částí URL adresy a určuje jednoznačný identifikátor nějakého místa (přesněji počítačové sítě či serveru) na internetu. Vznikla z důvodu, že komunikace pomocí IP adres je složitá:

- Pro lidi je složité si zapamatovat (v případě IPv4) 4 čísla pro každý web, který navštěvují,
- V případě HTTP se otevírá pouze jeden port – tím pádem by na jedné IP adrese mohl být jeden web. To by šlo obejít různými dalšími porty či rekonfigurace webů např. na `23.201.102.3/alza` ale to má stále problémy.

Doména se rozděluje do několik částí, které jsou oddělené tečkami, všechny části musí být ASCII a ignoruje se velikost písmen. Základní je generická (doména 1. řádu, top level domain = TLD), která určuje úplný „zdroj“ stránky, která je po poslední tečce. Hiearchický systém funguje tedy opačným čtením, než jsou domény napsané. Například v doméně `www.seznam.cz` je TLD `.cz`.

V rámci domén prvního řádu se většinou jedná o země, ale můžou se objevovat různé označení a nebo prakticky cokoliv[2]. Jedná se například o `.cz, .uk, .fr, .io, .website`.

Domény prvního řádu se většinou nepoužívají přímo, ale v nich si zákazník může registrovat domény druhého řádu. Doména druhého řádu nejčastěji určuje nějakou společnost, entitu či prostou stránku. Nejčastěji se kupuje doména druhého řádu na rok. Častou praxí je, že domény prvního rok stojí méně než na roky následující. Například se jedná o `seznam.cz, google.com, facebook.com`. Vlastněním domény druhého řádu můžete dále spravovat domény menších řádů.

Menším řádům domén již často říkáme subdoména[3]. Často se jedná o různé orgánizační jednotky, které mají vlastní weby. Například `support.google.com, mail.seznam.cz, ssps.cajthaml.eu`. S doménami s větším jak 4. rádem se v praxi setkáváme méně často, často se ale nepoužívají ke komunikaci vůči uživatelům.

Další využití subdomén může být:

- Různé služby (např. `ftp.seznam.cz`),
- Různé geografické oblasti (např. `eu.cnn.com`),
- Různé jazyky (např. `en.wikipedia.org`),
- Různé části firmy (např. `hr.microsoft.com`).

[2]Pokud máte dostatek prostředků, je možné ovlivnit různé organizace, aby vaší doménu prvního řádu schválily a tím jí budou správně rozpoznávat. Dále ale musíte spravovat další věci, jako je např. WHOIS či celkovou administraci.

[3]Sub, často „podřízený“.

- Různé projekty (např. `project1.company.com`).
- Různé verze (např. `v1.api.service.com`).
- A mnoho dalšího.

Iternationalized Domain Names (dále jen IDN) je standard, který dovoluje, aby se v doménách mohli vyskytovat libovolné znaky abeced z celého světa. Doméně je stále však tvořena ASCII, jen jsou znaky překódovány. Například znak `á` se přetvoří na `xn-1ca`. Pro českou doménu je vypnuto z bezpečnostních důvodů (přepis může přesměrovat na phishing stránku) a taky ze zmatení, protože pro jednu doménu by bylo nutné koupit další pro zaručení bezpečnosti. Společnost CZ.NIC, která českou doménu spravuje, pravidelně dělá dotazník[4], kde se na tuto technologii ptá. Obecně se uživatelé českého internetu staví proti IDN.

2.1.4 Uniform Resource Locator

Uniform Resource Locator (dále jen URL) je řetězec znaků s definovanou strukturou, který se používá pro adresaci na WWW a internetu obecně. Když chceme webovou stránku navštívit, tak nechodíme pouze na doménu, ale můžeme určovat další informace, nejčastěji však například cestu, na které najdeme nějaký zdroj. URL je rozdělena do částí a platí pro ně různé pravidla.

URL můžeme definovat následující části:

- **Protokol** – určujeme jakým protokolem budeme s cílem komunikovat,
- **Cíl** – určuje IP adresu nebo doménu, na kterou se chceme připojit,
- **Port** – určuje číslo portu, na který se chceme připojit. Pokud není uveden, tak se použije výchozí port pro daný protokol,
- **Cesta** – určuje cestu k nějakému zdroji na serveru,
- **Parametry** – určuje parametry, které se předávají na server,
- **Fragment** – určuje část zdroje, na kterou se chceme dostat.

V cestě můžeme použít stromovou strukturu souborů, která je oddělena lomítky (`/`). Porty jsou omezené na čísla mezi 0-65535. Parametry jsou odděleny otazníkem (`?`) a jednotlivé parametry jsou odděleny ampersandem (`&`). Parametr je ve formátu `název=hodnota`. Fragment je oddělený mřížkou (`#`) a může být libovolný řetězec.

Příklad URL: `https://www.ssps.cz:443/dejZnamku?name=Matej&znamka=5#vsechnyZnamky`

Spousta znaků nelze díky ASCII a speicálnímu významu v URL použít. Tyto znaky je nutné zakódovat pomocí procentového kódování. Například mezera se zakóduje jako `%20`.

Často se setkáte s tím, že před každou URL je `www.`. Tento prefix znamená World Wide Web a je to pouhá subdoména a nemá žádný speciální význam. Používá se z historických důvodů a pro lepší orientaci uživatelů – že mají hledat webovou stránku.

2.1.5 Vyhledávání na webu

Webový vyhledávač je automatický systém vyhledávající informace na WWW. Automaticky tedy server prochází (crawluje) stránky a zjišťuje si z nich informace, které poté uživatelovi, který vyhledává, ukazuje. Tyto systémy si tvoří tzv. index. Nikdy tedy uživateli neukazují aktuální stav webu, ale stav, který mají v indexu. Z různých odkazů se tvoří mapa (pavučina) WWW.

[4] https://háčkyčárky.cz/

Webový katalog je seznam odkazů, které vedou na různé weby. Je spravován ručně.

Webové vyhledávače lze omezit pomocí souboru robots.txt, ve kterém lze zakázat, aby tento web navštěvovali (nebo omezit nějaké URL). Etické crawleři toto Vaše rozhodnutí budou respektovat.

Na části WWW, kam se vyhledávače nemohou dostat, říkáme, že je to hluboký web (deep web). Jsou to jednoduše veškeré stránky, které potřebují přihlášení. Například se jedná o internetové bankovnictví, Vaše zprávy na Instagramu, či cokoliv obdobného. Jedná se o největší část WWW.

Hluboký web se často zaměňuje s temným webem (dark webem). To je web, na který se nedá dostat pomocí klasických technologií a je potřeba speciální software, speciální připojení, připojení v určitý čas a různé další ochrany. Dark web stojí na anonymitě pomocí různých technologií, které lze při určitých podmínkách obejít. Na dark webu se často konají ilegální prodeje a podobné věci.

2.1.6 Protokoly

Pro webové stránky existuje nespočet protokolů, které jsou pro jeho funkčnost nutné. My si ukážeme ty nejzákladnější z nich. Více o nich byste se měli dozvědět v předmětu PSI v 1. a 2. ročníku. Protokol vlastně definuje to, jak dva účastníci mezi sebou budou komunikovat tak, aby si rozuměli. Mají vlastní syntaxe, sémantiku a např. synchronizaci.

2.1.6.1 HTTP a HTTPS

Hypertext Transfer Protocol (dále jen HTTP) je aplikační protokol, který se používá pro přenos hypertextových dokumentů na WWW. Moderně se nepoužívá jenom pro přenos hypertextových dokumentů, ale i pro přenos různých dalších dat a považuje se za nejdůležitější protokol na internetu. Protokol je bezstavový, což znamená, že každý požadavek je nezávislý na ostatních. Běží na portu 80 a je textový.

Hypertext Transfer Protocol Secure (dále jen HTTPS) je rozšíření HTTP, které přidává vrstvu zabezpečení pomocí SSL/TLS, tedy šifrování. Šifrování probíhá pomocí certifikátů, které vydávají různé certifikační autority. Certifikáty poté nabízejí možnost použití asymetrického šifrování pomocí veřejného a soukromého klíče. Dříve se používal pouze na citlivé informace, jako jsou přihlašovací údaje, čísla karet a podobně. Nyní je v podstatě nutné ho používat všude, protože i běžný přenos dat může být citlivý. HTTPS běží na portu 443 a je také textový.

2.1.6.2 IP

Internet Protocol (dále jen IP) je určený na směrování paketů na internetu. Funguje na síťové vrstvě (komunikace mezi více sítěmi). Existují dvě verze, jedna je IPv4, která má již své adresy vyčerpané. Proto vznikla IPv6, která má i mimo počet adres další výhody.

Formát IPv4: `192.168.0.1`

Formát IPv6: `2001:0db8:85a3::8a2e:0370:7334`

2.1.6.3 DNS

Domain Name System (dále jen DNS) je systém, který překládá domény na IP adresy. Jedná se o hierarchický systém, který určuje že např. `seznam.cz` se převede na `77.75.76.3`. Může se totiž stávat, že nějaké subdomény mohou vést na jiné servery, aby se všechna doména nemusela nacházet na jednom serveru, což je nebezpečné. DNS neslouží jen na převod jmen, ale například i na určování e-mailových serverů a dalších záznamů pro počítače.

DNS se musí cachovat, tedy ukládat, aby se nemuseli počítače při připojení vždy ptát, kudy se mají vydat, aby navštívili tuhle doménu. Nejbližší uložiště DNS je například v počítači, u ISP a postupně dále. DNS server si ukládá svoje záznamy o daných doménách a komunikuje s ostatními, se kterými sdílí svoje záznamy.

Nejhlavnější DNS servery jsou root servery, které mají záznamy o doménách prvního řádu[5]. Autoritativní DNS servery jsou servery, které mají záznamy o nějaké doméně a na něm se spravují.

2.1.6.4 FTP

File Transfer Protocol (dále jen FTP) je protokol, který se používá pro přenos souborů mezi počítači na internetu. Podporuje nespočet operací, která každá dělá něco jiného. Využívá model klient-server a pro připojení je potřeba FTP klient, který je často i v prohlížeči.

Známí FTP klienti jsou např.: WinSCP, FreeCommander, FileZilla a další.

2.2 Nebezpečí

Internet je bez pozoru obecně velmi nebezpečné místo. Jeden z důvodu proč tomu tak je většinou přisuzována anonymita, kdy v podstatě kdokoliv může být kdokoliv.

Nejčastější nebezpečí je závislost na internetu, kde je uživatel závislý na samotném internetu, kde tráví například většinu dne a není schopen bez něj žít.

Obecně je dál nebezpečné zneužití osobních informací, do čeho patří údaje, fotografie, vaše pozice nebo třeba i záznamy webkamery a zvuky. Je důležité své údaje zadávat jen tam, kde je to potřeba a můžete dané společnosti věřit.

Dále se na internetu nachází podvody, které mohou vést ke ztrátě majetku, peněz, dat či zablokování zařízení. Je důležité si prověřovat, s kým komunikujete a například požadovat zálohy. Obecně je dobré se s daným člověkem sejít.

Nejdůležitější je téma kyberšikany, které souvisí i s tématem zneužití osobních informací. Kyberšikana může stát na vydíraní a dalších věcech a je dobré se ji bránit. Nejzákladnějším pravidlem by mělo být to, že danou věc řeknete další osobě, které věříte.

2.2.1 Malware

Zkratka názvu malicious software, který označuje nějaký nevyžádaný software nebo obecně program, který je určen k poškození počítače či uživatele. My si ukážeme jednotlivé typy.

[5] Je jich 13 po celém světě a spravují je různé organizace, proto je těžké vytvořit doménu prvního řádu, viz Kapitola 2.1.3.

2.2.1.1 Virus

Často je používaný jako synonymum pro malware. Samovolně se šíří bez vědomí uživatele. Obyčejně obtěžuje, ničí data a šíří se. Nejčastěji pomocí e-mailů a obecně přenosných médií.

2.2.1.2 Spyware

Určen ke sledování uživatele. Zaznamenává aktivitu uživatele a buď si vytváří log, který je jednou za čas poslán nebo aktivně posílá informace na server. Může se jednat o keylogger, tj. jen sledování kliku na klávesnice, ale i například pořizování snímků obrazovky či kamery.

2.2.1.3 Trojský kůň

Schovává se v zdánlivě nezávadném programu či souboru. Často při pirátění software, her a obdob. Uživatel nemůže jednoduše zjistit zda se jedná o trojského koně – velikost souboru se značně nemění (uživatel reálnou velikost neví). Často může pomáhat zkontrolovat hash daného souboru. Obsahuje jiné typy malware. Název pochází z řecké mytologie viz článek na Wikipedii.

Často se objevují i v často používaných souborech jako jsou zazipované balíčky (`.rar, .zip`), který například pro vybalení souborů něco udělá. Počítá se do toho i například Word se svými makry.

2.2.1.4 Počítačový červ

Šíří se po síti sám a bez vědomí uživatele. Často maže či odesílá soubory. Taktéž se často aktivuje až po čase, po tom, co se rozšíří více – tím zraní více míst/počítačů, což může vést ke větším škodám. Šíří např. i na routerech.

2.2.1.5 Adware

Po získání tohoto malwaru se uživateli začnou ukazovat různé nevyžádané reklamy, pop-up okna a další falešné ikony, které většinou žádaní uživatele něco koupit a kliknout na stránku, kde bude více reklam, za které útočník dostává zaplaceno. Často jsou takové programy „bezpečné“, jsou spíše otravné a mohou zpomalovat počítač.

2.2.1.6 Ransomware

Vychází z anglického slova ransom, které znamená výkupné. Po aktivaci toho malwaru se veškeré data na počítači (resp. discích) zašifrují klíčem. Na PC se uživateli ukáže informace o tom, že musí zaplatit v kryptoměně nějakou částku na určitou adresu. Za zaplacení nabízí útočník heslo k rozšifrování. Data nelze bez klíče v rozumném čase rozšifrovat – o data jste buď přišli, nebo věříte, že Vám klíč po zaplacení útočník předá. Když máte štěstí, o data nepřijdete, ale může se stát, že ransomware bude později znovu aktivován – v souborech může nadále existovat nebo v nich bude další malware.

Toto výkupné často má i časový limit, aby to uživatele nutilo vykonat nepromyšlenou akci. Po tomto časovém limitu budou všechny data smazána. Někdy obsahují i časový limit, který často zvedne částku, kterou je potřeba zaplatit.

2.2.2 Phishing

Zkratka pro password harvesting fishing. Jedná se o podvodnou techniku, která se používá k získávání citlivých údajů z různých míst. Často využívá podvodné e-maily, bannery, stránky, SMS, hovory, administrátory žádající údaje a další. Pro důvěryhodnost používají stejný design, kopírují stránky a snaží se působit důvěryhodně. Po získání údajů je použijí pro získání dalších informací či majetku a peněz.

Cílem tohoto úkolu jsou často nezkušení uživatele internetu jakéhokoliv věku. Často se však útočí i na starší obyvatele, kteří jsou obecně více důvěřiví. Česká stránka hoax.cz velké (nejen) phishingové útoky monitoruje a zveřejnuje.

2.2.3 Ochrana

Proti nebezpečí na internetu se lze chránit různými způsoby. Nejzákladnější poznámka by měla být ta, že většina nebezpečí vzniká pokud to uživatel dovolí. Tedy je obecně dobré prověřovat to co dělat a kontrolovat svůj bezpečnostní stav. Zjednodušeně neklikejte a nestahujte věci, které neznáte. Není dobré věřit e-mailům, které neznáte.

Je dobré mít na počítači antivirus či antimalware. Antivirus je program, který skenuje počítač pro nebezpečí, které jsou již dlouho známé a má svojí databázi, kterou aktualizuje. Na druhé straně, antimalware je program velmi podobný, ale má aktivní kontrolu nebezpečí. Kontroluje tedy často to, co daný program dělá a má velmi obsáhlou a aktualizovanou databázi (stahuje i několikrát denně) akcí, které jsou nebezpečné.

Další důležitá věc je mít vždy pravidelné aktualizace veškerého softwaru. Nové aktualizace mimo nové funkčnosti skoro vždy opravují nějaké chyby, které v předchozích verzí vznikli.

Při zadávání údajů je dobré kontrolovat certifikáty a domény. Důvěrná data nikdy nezadávejte na vyžádání.

2.2.4 Digitální stopa

Digitální stopa jsou záznamy na internetu, které vytváříme při procházení WWW a internet. Tyto záznamy se nacházejí jak v prohlížeči (klientovi) tak i na serverech. Patří do toho očekávané věci – příspěvky, komentáře, objednávky a další věci, které logicky musí server uchovávat.

Digitální stopa patří do cenných informací, což jsou informace, které mohou společnosti využívat k lepšímu marketingu a obecně k cílení. Dělíme je na vědomé (výše popsané, např. příspěvky, ...) a nevědomé (historie procházení, samotná návštěva stránky, kde na webu klikáte, ...). Tyto informace nemusí být určitě použité k špatné věci (tj. zanechané nepřáteli), ale mohou být využity k zlepšení služby.

Pro minimalizaci zanechávání stopy je dobré si nastavit prohlížeč tak, aby posílal webům co nejméně informací. Poté je dobré při první návštěvě webu nastavit cookies tak (podle cookies law od EU), že nechcete využívat soubory ke sledování. Proti některé stopě se nelze bránit – jak se říká, to co nahrajete na internet tak bude velmi pravděpodobně navždy.

K tématu patří taky metadata. Jsou to data, které popisují data (tj. data o datech). Jako ukázka můžou být například fotografie, kde se často ukládá, kdo je vyfotil, kdy byly vyfoceny, kdy byl soubor naposledy upraven a podobně. Tyto informace existují skoro ke každým datům – souborech, příspěvek, e-mailech.

K realizaci toho, jak moc jen samotný prohlížeč posílá na internet informací (metadat), doporučuji navštívit stránku AmIUnique, na které si můžete ověřit jak unikátní jste. Webové stránky mohou například skenovat nainstalovaná písma a podle toho je velmi jednoduché kohokoliv poznat.

2.3 Webové prohlížeče

Webový prohlížeč je software, který je určený k procházení WWW, zjednodušeně webů. Jeho cílem je komunikovat se servery v různých protokolech, nejčastěji pomocí HTTP/S. Prohlížeč však má různé věci, které s procházením nemusí přímo souviset. Často řeší i jiné úlohy (záložky, kalendáře, emaily, historii) a jsou nastavitelné.

Hlavní částí prohlížeče je vykreslovací (renderovací) jádro, které slouží pro vykreslování stránek v jazyce HTML, se kterým se tento rok budeme učit. Nejčastější jádra jsou:

- Gecko (Mozilla FireFox),
- Webkit (Safari),
- Blink (Google Chrome),
- a např. Trident (Internet Explorer).

Jádra se hlavně liší v tom, jak se webové stránky vykreslují.

Chromium je vlastně takový „prohlížeč“ bez ničeho na víc. V podstatě obsahuje jen vykreslovací jádro Blink, které obsahuje Google Chrome. Je to velmi oblíbený nástroj, který se používá pro vykreslování webových stránek bez zbytečných věcí na více. Velmi mnoho prohlížečů na něm staví svůj základ a nepoužívají tím své jádra – jedná se např. o Vivaldi, Operu, MS Edge, Brave a spoustu dalších. Chromium je opensource a tedy každý do něj může nahlížet a upravovat jej. Chromium lze používat samostatně jako náhrada Google Chrome, ale neobsahuje například ani synchronizaci mezi zařízeními. Je to kvůli tomu, že nemá v sobě žádné Google služby.

Hlavní nevýhodou Chromia je to, že jej využívá již tolik prohlížečů, že se stal Google gigantem, který může velmi jednoduše ovlivňovat vývoj webu – velmi se snížila kompetivnost na tomto trhu. Jediné použitelné možnosti jsou Gecko a Blink, kdy Webkit od Safari s weby velmi stagnuje.

Moderní prohlížeče by se měli co nejvíce přizpůsobit potřebám uživatele, jedná se např. o:

- Úpravy barev,
- Úpravy velikosti písmen a fontů,
- Umí ochránit uživatele před reklamou či sledovači,
- Má anonymní režim,
- Být rychlý a nebrat zbytečné zdroje počítače,
- Podporuje rozšíření,
- Má synchronizaci,
- A další.

2.3.1 Certifikáty

Certifikát je určen pro jednoznačnou identifikaci entity (stránky, služby, uživatele) na internetu. Stojí na asymetrickém šifrování a podpisech. Podepsání je vlastně akt, kdy je certifikát podepsán (jiným) certifikátem a tím je schválena jeho existence – pravost. I bez podepsání certifikátu jeho šifrování funguje, ale nikdo nemůže ověřit, že je certifikát od té správné entity.

Certifikát obsahuje veřejný (encrypt) klíč, který slouží k šifrování dat. Poté obsahuje soukromý (decrypt) klíč, který slouží k rozšifrování dat. K tomu obsahuje ještě expiraci daného certifikátu – jedná se o datum, po kterém by se certifikátu nemělo důvěřovat. Obsahuje další informace o tom, kdo jej podepsal (vydal), pro koho je určen, pro jaké je domény a podobně.

Certifikační autorita je společnost, která podepisuje cizí certifikáty za určitý obnos. Poté je certifikát platný a kdokoliv si jeho pravost díky klíče certifikační autority ověřit. Tyto služby jsou často zpoplatněné (poskytují více věcí a služeb) a nebo i zdarma, jako např. Let's Encrypt. Certifikát může autorita zneplatnit.

Důvěryhodnost certifikátu je založen právě na tom, že si kdokoliv může zkontrolovat kdo jej podepsal. Lze zkontrolovat i jaký certifikát podepsal certifikát, který jej podepsal a tak dále. Klíče pro asymetrické šifrování jsou natolik bezpečné na normální počítačích (na kvantových tomu tak nemusí být), že je nemožné je v rozumném čase dešifrovat i se všemi PC na světě. Přesto se používá životnost certifikátů, aby se certifikáty často měnili. Když si sám vytvoříte certifikát, tak si jej sám můžete podepsat svým certifikátem. Jedná se tzv. o self-signed certifikáty, které se hodí na osobní použíti a prohlížeče a další služby jej berou jako nebezpečný. Hlavní certifikáty, tj. root certifikáty jsou právě podepsány samy sebou. Věříme jim protože uložené jako důvěryhodné v prohlížeči či operačním systému a nebo nám je někdo spravuje.

Certifikáty se používají všude – např. v HTTPS, VPN, SFTP, SSH a spoustě dalších službách.

Méně bezpečné je symetrické šifrování, které používá jeden klíč pro šifrování i dešifrování.

2.4 Hostování stránek

Hosting je služba, která Vám dovoluje nějak pronajmout část nebo celý počítač. Není ani nutné, aby byl počítač připojen k internetu. Typů hostingů je nespočet. V moderní době si pod pronájem představme nějaký cloud, který je schopen komunikovat s internetem a nebo např. něco vypočítávat.

Serverhosting je pronájem celého serveru (počítače), na kterém si lze spustit jakýkoliv software. Zákazník má přístup k celému serveru a může složit úplně na cokoliv – např. hostování stránek, her či různé vnitřní systémy. Často to není celý fyzický server, ale virtuální server, který je na fyzickém serveru vytvořený pomocí virtualizace.

Webhosting je pronájem místa na cizím serveru, který je určen na hostování stránek. Je to omezené místo a jsou omezeny i další parametry. Server a web spravuje daná organizace, většinou nejsou práva pro pokročilé nastavení. Často se omezuje kolik místa webové stránky mohou zabrat, kolik je maximální přenos dat, jaké technologie jsou povolené a další věci.

Freehosting je nějaký typ hostingu, který je zdarma. Často chybí technologie, databáze a další služby. Vkládá se například reklama nebo je jinak omezena. Používá se taky jako ukázka služeb daného hostingu. Často je k tomu dispozici i doména 3. řádu zdarma[6].

Mezi další služby, které hostingy nabízí, patří:

- Domény,
- Certifikáty,
- Databáze,
- E-maily,
- Výpočetní služby,

[6]Proč? Protože po koupi domény 2. řádu je možné vytvořit neomezené množství domén 3. řádu.

- A další.

Pro výběr hostingu existuje nespočet kritérií. Je nutné odhadnout, kolik webová stránka zabere místa a jaké technologie budou potřeba. Mezi známe české hostingy patří:

- Wedos,
- Endora,
- Forpsi,
- Stable,
- Active24.

2.4.1 Právní aspekty

Webové stránky jsou chráněny autorským zákonem. Často se na stránkách uvádí formulace např. `© 2026 Cajthaml, All right reserved`, která je vlastně jen pro upozornění. Bez uvedení licence je stránka stále pod autorským zákonem. Vašeho práva jako autora díla není možné se vzdát. Tento termín je však takovým standardem, že je velmi dobré jej tam psát.

Bez používáním jakékoliv věci je dobré zkontrolovat zdroj, licenci a podmínky, pod kterými můžete danou věc používat. V žádném případě licenci neignorujte – můžete z toho mít velké problémy.

Pro ulehčení licencování věci existují předpřipravené licence, které definují jak se věci mohou používat již předem. Jedná se například o Creative Commons, které definují různé kombinace licencí, které povolují a zakazují různé akce s dílem. Jedná se například o zákaz používat věc pro komerční využití, nebo že musí po úpravě díla být použita stejná licence.

Druhá strana mince je fair use, kde použijete věc v zájmu sebe, ale však uvedete správně zdroj – jedná se např. pro školní účely či kritiku.

Na služby se normálně vztahují nějaké podmínky, které je dobré definovat. Při jakékoliv práci s osobními údaji je nutné dodržovat různé zákony (např. GDPR v EU) a uživatele o tom informovat. Podobně na tom je tzv. cookies law, které definuje, že uživatel musí dát souhlas s ukládáním cookies, které neslouží pro fungování webu (např. sledovací cookies). Cookies jsou ale však jakékoliv lokální soubory, které web může ukládat v prohlížeči uživatele.

2.4.2 Správa obsahu

Content Management System (CMS) je systém pro správu obsahu. Jedná se o nějaký redakční nebo publikační systém, ve které pomocí uživatelského prostředí lze editovat obsah. Existuje pro různé jazyky a platformy.

Správcům obsahu nechcete dávat přístup k zdrojovému kódu, databázi a dalším věcem, protože by mohli nechtěně něco pokazit. Navíce správci nemusí mít technické znalosti, aby mohli obsah spravovat.

Jedná se např. o: WordPress, Strapi, Joomla!, Drupal a další.

Webové stránky

3.1 HTML

Hyper Text Markup Language (dále jen HTML) je značkovací jazyk. Nejedná se tedy o klasický programovací (jako například C# či Python) nebo skriptovací (JavaScript/Lua) jazyk, ale o jazyk, který jenom definuje určité data a to v XML s vlastními definicemi tagů. Technologií XML se v předmětu WBA zabývat ale nebudeme — není nutné si to ani pamatovat. Tedy webová stránka v nejzákladnější podobě je jenom textový soubor, který obsahuje text, který je označen různými vlastnostmi, co text je.

HTML bylo představeno v roce 1990. Od té doby vznikly různé verze, ta nejdůležitější je HTML5 z roku 2012, kterou se ve WBA budeme zabývat. Tato verze se hlavně zaměřila na to, aby šel multimediální obsah spouštět na webu. Cílilo se i na to, aby webové stránky byly přehledné i pro počítače a mohly se jednodušeji crawlovat, resp. indexovat.

HTML se používají zejména pro webové stránky, protože definují různé tagy, které vytváří na stránce různé prvky, které může uživatel vidět. Tyto soubory mají obvykle příponu `.html`.

3.1.1 Značky

Tag je značka v HTML. Jména jsou rezervovaná a určují obsah a často i chování prvku na stránce. Definujeme tím tedy to, co se v prohlížeči vykreslí. Tyto tagy se vždy zapisují v špičatých závorkách.

> 💡 **Sémantika**
>
> Tomu, co tag popisuje za obsah, říkáme sémantika. Sémantika je slovo, které v obecném studiu webových stránek uslyšíte často na začátku a poté až se dostanete k velmi pokročilým částem webových technologií. Sémantika se zaměřuje na to, jaký význam a funkci jednotlivé tagy mají. Chceme aby vždy byla správna a to z několika důvodů. Pro nás nejdůležitější je strojové zpracování stránek pro vyhledávače. Vyhledávače, viz Kapitola 2.1.5, používají sémantiku k tomu, aby pochopili, o čem stránka je a jaký obsah na ní je. Mimo tagy to lze řešit i jinými způsoby[7], ale to už je mimo rozsah tohoto předmětu.
>
> Další důležité použití sémantiky je přístupnost, která nám umožňuje vytvářet weby, které jsou použitelné i pro lidi s různými handikapy. Například pro nevidomé lidi, kteří používají čtečky obrazovky, se místo zobrazení stránky přečte obsah stránky. Pokud bude sémantika špatná, čtečka obrazovky nebude schopná správně interpretovat obsah stránky a uživatel nebude vědět, co se na stránce nachází.

html

Zápis nepárového tagu

```
1 <tag>
```

[7] Například pomocí ARIA atributy, JSON-LD a další.

> ⚠ **Interpretace tagu a jeho vzhledu**
>
> Základní chybou při práci s HTML je to, že si někdo myslí, že musí použít konkrétni tag HTML na to, aby prvek vypadal určitým způsobem. To ale není pravda, protože tagy mají přesně daný význam a účel. Webové prohlížeče (viz Kapitola 2.3), resp. vykreslovací jádra, pak podle toho, jaký tag je použitý, prvek vykreslí. Prohlížeče sami připravují základní styly pro jednotlivé tagy, což tuto častou chybu ještě více podporuje.

Výše představený tag je nepárový a není moc používaný. Existuje ale párový tag, který určuje, že něco někde začíná a někde končí (vymezuje kde a co). Skládá se ze dvou tagů, kde jeden je počáteční a druhý koncový. Koncový poznáme tak, že po úvodní špičaté závorce se nachází lomeno. Takový zápis tedy vypadá následovně. Mimo tento zápis daný tag, pokud nebyl dříve definován, neplatí.

html
```
Párový tag
1 <tag></tag>
```

Jednotlivé tagy lze spojovat dohromady — vkládat je do sebe, např.:

html
```
Vnořené tagy
1 <tag1>uz <tag>nevim</tag> co</tag1>
```

Tento zápis znamená, že vše bude označené tag1 a slovo nevim uvnitř bude označeno tagem tag.

Nějakým nedopatřením se však může stát, že se tagy pokazí a nastane křížení tagů. To znamená, že nějaký párový tag začal uvnitř jiného párového tagu, ale ukončí se až někdy mimo něj. Taková ukázka je např.:

html
```
Křížení tagů
1 <tag1>uz <tag>nevim</tag1> co</tag>
```

V tomto případě má prohlížeč několik možností. První z nich je ta, že se tagy pokusí opravit. Při těchto změnách se Vám ale velmi může rozbít web, protože to není to, co jste původně zadali. To rozbije nejen význam, ale později i vzhled stránky. Proto se křížení tagů musí vyřešit již při vytváření stránky. Tento zapsaný tag by se například opravil následovně:

html
```
Opravené křížení tagů
1 <tag1>uz </tag1><tag1><tag>nevim</tag></tag1><tag> co</tag>
```

Webové prohlížeče vždy chtějí, aby se stránka vykreslila správně. Prohlížeče nikdy neudělají oznámení o chybě tak, že by se stránka nevykreslila. Celý web se sanží být zpětně kompatibilní, což znamená, že i velmi staré weby musí nějakým způsobem fungovat. Proto na webových stránkách vznikají tzv. deprecated obsah (tagy, pravidla, ...), tedy zastaralé věci, které by se již neměli používat, ale prohlížeče je stále podporují.

Podobně je to i s dalšími chybami, jako je například použití nesprávného tagu (neexistuje či pro daný účel není určený), špatnou hiearchii, chybějící tagy apod.

Tagy jsou definované přesně a my je nebudeme využívat k jiným účelům, než byly vytvořeny.

3.1.2 Atributy

Jakýkoliv tag může mít mnoho atributů. Atributy definují nějaká další data či nastavení, které popisují přesnější chování či význam prvku. Některé atributy definují chování i pro děti (tj. pro tagy uvnitř) daného tagu. Tyto atributy jsou většinou speciální pro daný tag a nikde jinde nefungují. Existují však i globální atributy, které lze nastavit v podstatě na cokoliv. Tagy se vždy určují u počátečního tagu. Atribut má vždy své jméno a může mít hodnotu.

Atributy zapisujeme následovně:

html
Zápis atributu
```
1 <tag atribut="hodnota">
```

html
Zápis atributu bez hodnoty
```
1 <tag atribut></tag>
```

Pro lepší pochopení ukážeme atributy na tagu `a`, který dle HTML5 definuje na stránce odkaz, na který lze kliknout. Pro určení odkazu používáme atribut `href`. Pro vytvoření odkazu na Google by kód vypadal následovně:

html
Tag a s atributem href
```
1 <a href="https://www.google.com">Odkaz na Google</a>
```

Mezi zmíněné nejpoužívanější atributy se řadí `id` a `class`. Oba dva slouží na stylování a skriptování stránek. Používají se na vybírání prvků na stránce pro další práci. Identifikátor, neboli atribut `id`, určuje jeden prvek na stránce s daným jménem. Na stránce může být pouze jedenkrát. Třídy, neboli atribut `class`, je seznam tříd, které mohou být definovány více prvkům. Uvnitř hodnoty atributu `class` se jednotlivé třídy oddělují mezerou.

html
Tag a s atributem id
```
1 <a id="home">Domů</a>
```

html
Tag a s atributem class
```
1 <a class="btn primary">Odkaz jako tlačítko</a>
```

3.1.3 HTML entity

HTML entita je zkratka pro znak, který se z různých důvodů těžko v prohlížeči zobrazuje. Častý důvod pro to je to, že to buď koliduje s něčím, co definuje HTML (resp. XML) a nebo díky kódování může být znak namapován úplně jinde. Nejčastěji se používají pro znaky < a >, které se používají všude v HTML.

Entity se vždy zapisují s ampersandem (&) na začátku a středníkem (;) na konci. Text mezi těmito znaky je to, co se vykreslí (najde a namapuje).

Jako ukázka může sloužit zápis `&copy;` který se přepíše na ©. Veškerý seznam entit lze nalézt např. na Wikipedie.

3.1.4 Komentáře

Komentáře jsou velmi potřebné všude. V HTML se zapisují jedním způsobem a to pomocí sekvence `<!--` a `-->`, kde mezi nimi se vše považuje za komentář. Tyto komentáře jsou víceřádkové.

Komentář v HTML

```html
<!-- Toto je komentář -->
<!-- Může být i víceřádkový
<p></p>
> Všimněte si, že zde je i tag p považován za komentář a tedy se nevykreslí.
-->
```

Viditelnost komentářů

Komentáře jsou viditelné v kódu stránky a tedy neměly by obsahovat žádné citlivé informace.

3.1.5 Základní struktura

Základní struktura HTML slouží k definici stránky. Tato struktura se musí používat na všech stránkách (výjimky v podstatě neexistují). Jedná se o základních šest tagů, do kterých se vkládají další. My si jednotlivé tagy ukážeme a popíšeme.

Doplnění chybějících tagů

Pokud některý z těchto tagů chybí, prohlížeč se je pokusí doplnit, tj. obalí stránku do těchto tagů.

<doctype>: Typ dokumentu

Deklarace typu dokumentu. Určuje, že se jedná o HTML5 dokument.

```html
<!DOCTYPE html>
```

<html>: Kořenový tag

Kořenový tag celé stránky. Určuje, že se jedná o HTML dokument.

Atributy:

lang

Nepovinný atribut, který určuje jazyk stránky[8], Seznam všech jazykových kódů lze nalézt zde. Tento tag používají prohlížeče např. pro překlady. Je taktéž velmi důležitý vzhledem k přístupnosti a pro vyhledávače. Jazyk stránky tedy vždy definujte. Pokud používáme více jazyků, je lepší nenastavit žádný jazyk na kořenový tag.

html
```
<!DOCTYPE html>
<html lang="cs">
  ...
</html>
```

<head>: Hlavička stránky

Obsahuje metadata o stránce, odkazy na styly, skripty a další. Tyto tagy se nevykreslují.

html
```
<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
</html>
```

<meta>: Metadata

Definuje data o stránce, zejména pro prohlížeče a vyhledávače. Pro nás je povinné určit kódování stránky a zobrazení na různých zařízeních.

Atributy:

charset Určuje kódování znaků stránky. Nejčastěji se používá `UTF-8`.

name Název metadat. Některé běžné hodnoty jsou `author`, `description`, `keywords` a další.

content Hodnota metadat. Obsahuje informace podle atributu `name`.

html
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="author" content="Jméno autora">
    <meta name="description" content="Popis stránky">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

[8]Pokročilí studenti si najdou informace o tom, že atribut lang můžeme používat i na jiných tagech. Poté můžeme mít na stránce více jazyků. Většinou se ale používá jen na kořenovém tagu.

```
      > Definuje, jak se má stránka zobrazit na různých zařízeních,
    konkrétní hodnota znamená, že šířka stránky bude odpovídat šířce
    zařízení a počáteční zoom bude 1.
  </head>
</html>
```

<title>: Název stránky

Definuje název stránky, který se zobrazuje v záložce prohlížeče a ve výsledcích vyhledávání.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Název stránky</title>
  </head>
</html>
```

<body>: Tělo stránky

Obsahuje veškerý viditelný obsah stránky, jako jsou texty, obrázky, odkazy a další.

```html
<!DOCTYPE html>
<html>
  <body>
    ...
  </body>
</html>
```

Ukázková kompletní struktura HTML dokumentu vypadá následovně:

Kompletní HTML dokument

```html
<!DOCTYPE html>
<html lang="cs">
   <head>
      <title>Smíchovská střední průmyslová školy a gymnázium</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
  scale=1.0">
   </head>
   <body>
      <p>Dobrý den!</p>
      <p>Vyzkoušejte <a href="https://google.com">tuto úžasnou stránku</
  a>!</p>
   </body>
```

```
12 </html>
```

Kdybysme tento kód otevřeli v prohlížeči, uvidíme dva řádky textu, kde část posledního řádku je odkaz na Google. Stránka by taktéž obsahovala titulek. Stránku by bylo možné otevřít na různých zařízeních a vypadala by správně.

3.2 Vývojové prostředí

Vývojové prostředí, často pojmenovávané jako Integrated Development Environment (dále jen IDE), je software, který slouží k programování, kódování či skriptování v určitých jazycích a tzv. development stack. To jsou skupiny různých nástrojů, technologií a jazyků, které slouží pro vytváření určitých věcí.

Vývojové prostředí není jen prostý editor textových souborů. Obsahuje totiž další nástroje, které vývoj ulehčuje. Jedná se o např. kompilátor, debugger a další věci.

Mezi klasické IDE pro vývoj webových aplikací patří například:

- Visual Studio
- Visual Studio Code
- WebStorm

Často jsou různé chytré textové editory nesprávně označované jako IDE. Jedná se například o Visual Studio Code, který nemá primárně určený žádný jazyk nebo stack. Různými rozšířeními však z něj můžete vytvořit něco, velmi blízké IDE.

V předmětu WBA budeme ve škole používat Visual Studio Code. Pro úspěšné splnění předmětu je nutné jej mít nainstalované doma na počítači či notebooku, na kterém můžete pracovat. Můžete však používat jakékoliv IDE či chytřejší editor. Není možné používat např. notepad či obdoby.

Klíčové pro rychlou tvorbu webových stránek je tzv. live server (či hot-reload). To je nástroj, který sleduje změny v souborech a pokud dojde ke změně, automaticky stránku znovu načte v prohlížeči. To nám ulehčí rychle měnit stránku a styly a ihned vidět výsledek s ještě menší námahou.

3.3 Soubory na webových stránkách

Na webových stránkách máme různé soubory a některé mají speciální význam. My si ukážeme jen hrstku nejpodstatnějších z nich. Z předchozích stránek známe soubory HTML.

Jeden z nejdůležitějších souborů je tzv. index soubor. Jedná se o soubor, který se jmenuje index a má jakoukoliv příponu. Tento soubor se (v základním nastavení většiny webových hostingu) načte právě tehdy, když přistupujeme v URL na složku. Při přístupu server proskenuje složku a dle priority (obvykle např. php > html > txt > ...) zobrazí právě indexový soubor.

> 📁 **Obsah složky místo indexového souboru**
>
> Některé hostingy místo indexového souboru či když není indexový soubor nalezen, zobrazí obsah složky. Obecně toto považujeme za nebezpečné vzhledem k tomu, že uživatel může vidět soubory, na které by třeba obyčejně tak snadno neměl přístup.

Na webových stránkách chceme například i obrázky. Nejzákladnější typy jsou rastrové obrázky, které uvnitř ukládají jednotlivé pixely a v případě komprimace nějaké zmenšená data. Druhé jsou vektorové obrázky, které místo jednotlivých pixelů ukládají rovnice, podle kterých se obrázek na počítači vykreslí. Rovnice jsou různého typu a způsobují, že se obrázky mohou nekonečně zvětšovat bez ztráty kvality. Na druhou stranu bývají (když jich máte desítky na webových stránkách), pomalé, protože se musí CPU zapojit v počítání. Pro rastrové obrázky známe přípony (resp. typy) `.jpg`, `.png`, `.gif`, `.webp` a pro vektorové `.svg`.

Soubory na webových stránkách vždy pojmenováváme bez diakritiky a speciálních znaků, bez mezer a nejlépe s malými písmeny. Často mají být názvy krátké a výstižné. Důvod je dobrý — protože se soubory zobrazují v URL, často je nutné aby se v nich uživatel vyznal a případně je byl schopen i napsat. Diakritika je špatná zejména pro uživatelé z jiných zemích[9]. Špatné názvy mohou generovat i řadu problémů v prohlížeči.

V prohlížečích, jak již víme z URL (viz Kapitola 2.1.4), umíme přistupovat i ke konkrétním souborům. O cestách, resp. složkách, se budeme bavit později (viz Kapitola 3.6).

3.4 Základní značky

Nyní si ukážeme některé základní značky, které se na webových stránkách používají. Jedná se o ty nejzákladnější, abychom se mohli začít věnovat grafické části webových stránek. Pro přesný význam značek doporučuji nahlédnout do oficiální specifikace HTML.

3.4.1 Textové značky

<p>: Odstavec

Definuje odstavec textu. Prohlížeče obvykle přidávají před a za odstavec mezeru.

```html
1 <p>Toto je odstavec textu.</p>
```

Text mimo tagy

Častá chyba začátečníků s HTML je to, že do HTML vkládají text do tagu bez jakéhokoliv obalení. Jediné validní místo, kam lze vložit text, je do tagů určených pro text. Jedná se například o tagy `p`, `h1-h6`, `pre` a další. Pokud text není obalený v žádném tagu, prohlížeč je vykreslí.

**
: Zalomit řádek**

Vloží zalomení řádku. Tento tag je nepárový. Používáme například pro jednotný text, který ale je správně rozdělen na více řádků.

```html
```

[9] Chtěli byste zapisovat či číst čínské znaky v obrázku?

```
1 <br>
```

⚠ **Špatné použití tagu br**

Znovu, velmi častá chyba začátečníků je to, že tag `br` používají tam, kde se nehodí. Tento tag nikdy nepoužíváme např. na vytvoření místa mezi odstavci.

: Důležitý text

Definuje text, který je důležitý a má být zvýrazněn.

html
```
1 <p>Toto zkoušení je <strong>povinné</strong>!</p>
```

<i>: Alternativní význam

Definuje text, který má jinou výslovnost, tón nebo je z jiného důvodu odlišený od okolního textu.

html
```
1 <p>Kočka (<i>Felis catus</i>) je domácí zvíře.</p>
```

<u>: Anotace

Definuje text, který je z nějakého důvodu anotovaný bez dalšího významu. Například se používá pro označení chyb v textu.

html
```
1 <p>Myslím si, že HTML je <u>programovací jazyk</u>.</p>
```

<sup>: Horní index

Definuje text, který je zobrazen jako horní index. Používáme jen tehdy, kdy by absence tohoto tagu změnila význam textu.

html
```
1 <p>Alfa částice jsou jádra helia: He<sup>2+</sup>.</p>
```

<sub>: Dolní index

Definuje text, který je zobrazen jako dolní index. Používáme jen tehdy, kdy by absence tohoto tagu změnila význam textu.

html
```
```

```
1 <p>Voda má chemický vzorec H<sub>2</sub>O.</p>
```

<del>: Smazaný obsah

Definuje obsah, který byl smazán z dokumentu.

html

```
1 <p>Nákup</p>
2 <p>- <del>mléko</del></p>
3 <p>- chleba</p>
```

<s>: Neaktuální obsah

Definuje obsah, který již není aktuální.

Zejména používáme tehdy, kdy je potřeba ukázat, že něco je špatně. Zároveň ale potřebujeme, aby bylo vidět, co původní informace byla.

html

```
1 <p>Cena je <s>100 Kč</s> 80 Kč.</p>
```

<hr>: Tématické oddělení

Rozdělí čast obsahu na dvě části, ve kterých proběhla změna tématu, resp. kontextu. Tento tag je nepárový.

html

```
1 <hr>
```

<b>: Zvýrazněný text bez významu

Definuje text, kterému je věnována vizuální pozornost bez dodatečného významu. Používáme například pro klíčová slova, názvy produktů nebo akční instrukce.

html

```
1 <p>Produkt: <b>SuperMix 3000</b></p>
2 <p>Klikněte na <b>Start</b> pro spuštění programu.</p>
```

<ins>: Nově vložený obsah

Definuje obsah, který byl nově vložen do dokumentu. Používáme například při opravách nebo doplňování textu.

html

```
1 <p>Nákup</p>
```

```
2 <p>- <ins>mléko</ins></p>
3 <p>- chleba</p>
```

3.4.2 Nadpisy

Nadpisy jsou velmi důležitá součást webových stránek. Slouží k označení a určení části webů, kde se nachází text a velmi krátce je sumarizuje. Používají se kvůli spoustě věcí, nejzákladnější z nich je ta, aby se na webu lépe orientovalo. Tedy, aby šlo co nejrychleji najít část webu. Nadpisy se na webových stránkách dělí do úrovní. V HTML existuje šest úrovní, které jsou číslované od 1 do 6, kde jedna je nejdůležitější (nejvyšší). Na webu tyto čísla používáme v hierarchickém systému, tedy, že nadpis úrovně dva patří pod úroveň jedna a tím říká, že patří do nadřazené úrovně.

<h1>: Nadpis

Definuje obsah, který má brán jako nadpis. Můžeme definovat šest úrovní nadpisů, kde `h1` je nejdůležitější a `h6` nejméně důležitý. Jiné nadpisy mimo rozmezí `h1` až `h6` neexistují. Měli by uvádět následující obsah na stránce.

html
```
1 <h1>Nadpis úrovně 1</h1>
```

Nadpisy nejsou jen pouhé číslování a škatulkování obsahu, ale určuje to rozložení celého webu a to, jak se čte. Na stránce musí být pouze jeden nadpis úrovně jedna, který definuje nejdůležitější část stránky, podobně jako tag title. Poté se na stránce musí logicky objevovat pouze nižší úrovně a to tak, že dávají smysl pro čtení návštěvníky tak i v kódu.

🔢 **Limitace hlavního nadpisu**

Tag `h1` tedy musí být vždy na stránce – i kdyby nebyl vidět na finální stránce. Tag `title` určuje název stránky, včetně například názvu webové stránky. Tag `h1` určuje hlavní nadpis stránky – co je na aktuální stránce za obsah. Studenti často do `h1` dávají název webu a opakují ho skrz webové stránky.

Správné hierarchické uspořádání nadpisů je například takovéto:

html
```
1 <h1>Jak upéct dort</h1>
2   <h2>Ingredience</h2>
3     <h3>Ingredience těsta</h3>
4     <h3>Ingredience krému</h3>
5   <h2>Instrukce</h2>
6     <h3>Příprava</h3>
7     <h3>Pečení</h3>
8     <h3>Krém</h3>
9     <h3>Sestavení</h3>
10  <h2>Servírování</h2>
```

```
Odsazení je jen pro ukázku hierarchie.
```

Výše uvedená ukázka je čitelná jak pro člověka, tak i pro robota, který by stránku načetl. Ví totiž, že Jak upéct dort obsahuje Instrukce jak vytvořit Krém. V základním formátováním od prohlížeče je každá úroveň nižšího řádu menší svojí velikostí textu, kvůli čitelnosti. Ale tím se znovu v HTML nesmíme řídit.

3.5 CSS

Cascading Style Sheets (dále jen CSS) je jazyk, který slouží na stylování webových stránek. Často CSS říkáme kaskádové styly.

Jedná se o technologii, která se používá pro stylování a designování webových stránek. Je přímo závislá na technologii HTML. CSS nemá na webových stránkách v podstatě žádnou alternativu a musí se tedy používat pokud chceme, aby webové stránky vypadaly nějak hezky. Kód jsou různé definice a nastavení, které HTML obsah upravují designově. Tyto definice a nastavení čte vykreslovací jádro (viz Kapitola 2.3)[10].

Jako stylování si můžeme představit velikost písma, barvy pozadí, textu či obrázky a rámečky.

3.5.1 Definice

CSS definice se nazývají pravidla. Pravidlo se skládá ze dvou částí:

- Selektor,
- Deklarace.

Uvnitř deklarace se nacházejí jednotlivé vlastnosti a jejich hodnoty.

Vlastnosti jsou jednotlivé klíčové věci, které lze nastavit. Za každou vlastnost s hodnotou se musí psát středník. Celá deklarace je zaobalena do složených závorek.

Pod vlastnostmi si znovu můžeme představit například barvu pozadí, velikost písma, okraje a další. CSS definuje stovky vlastností, které lze nastavovat. K těm je možné nastavovat tisíce různých hodnot. Hodnoty se většinou zjednodušují do „skupin“, například existuje typ „barva“ a „velikost“, které mají své specifické dovolené hodnoty. Tyto zápisy existují zejména pro zjednodušenou implementaci vykreslovacího jádra. V celých těchto skriptech jsou zápisy zjednodušeny, aby byly lépe pochopitelné pro studenty.

css

Základní struktura CSS pravidla

```
1 selektor {
2   vlastnost1: hodnota1;
3   vlastnost2: hodnota2;
4 }
```

3.5.2 Složené vlastnosti

Jednotlivé vlastnosti se umí skládat dohromady do tzv. složených vlastností. To znamená, že jedna vlastnost může obsahovat více hodnot, které se oddělují mezerou. Často tento zápis

[10] Vzpomeňte si na zpětnou kompatibilitu a přidávání nových funkcí do vykreslovacích jader.

používáme, když nechceme např. u designu nastavovat všechny okraje zvlášť, ale chceme je nastavit najednou – na společnou hodnotu. Většinu složených vlastností poznáme tak, že mají v názvu - (pomlčku).

Ještě žádné konkrétní vlastnosti neznáme, ale když si například představíme vlastnosti `background-color`, tak její hodnotu je možno nastavit uvnitř složené vlastnosti `background`, která může obsahovat další vlastnosti.

3.5.3 Propojení CSS s HTML

CSS lze s HTML propojit těmito způsoby:

- Inline styly,
- Interní styly,
- Externí styly,
- Importované styly.

Inline styly jsou styly, které se píší přímo do HTML tagu pomocí atributu `style`.

Inline styly

```html
<p style="color: red; font-size: 20px;">Toto je červený text s velikostí 20px.</p>
```

Inline styly se používají velmi zřídka[11] a to jen na výjimečné případy, kdy je potřeba změnit konkrétní prvek na stránce oproti jeho základnímu stylu. Například používáme pro uživatelskou možnost změnit si barvu profilu a podobně.

Interní styly jsou styly, které se píší přímo do HTML dokumentu uvnitř tagu `style`, který se nachází v tagu `head`. Tato možnost nedovoluje opakovné použití stylů na více stránkách.

<style>: Interní styly

Dovoluje psát CSS přímo do HTML dokumentu. Vkládáme zejména pouze do tagu `head`, ostatní jsou možné, ale nedoporučované.

```html
<html>
  <head>
    <style>
      p {
        color: blue;
        font-size: 18px;
      }
    </style>
  </head>
</html>
```

[11] Čistě prakticky se používají hodně, problémem je ale to, že je vývojář sám nezapisuje přímo do HTML, ale do JS (či jiných šablon). To si ukážeme později.

Externí styly jsou styly, které se píší do samostatného souboru s příponou `.css`. K tomuto souboru se pak v HTML dokumentu odkazujeme pomocí tagu `link`, který se nachází v tagu `head`. K cestám souboru se dostaneme později (viz Kapitola 3.6).

<link>: Odkaz na externí styly

Slouží k připojení externího CSS souboru do HTML dokumentu. Vkládáme zejména pouze do tagu `head`, ostatní jsou možné, ale nedoporučované.

Atributy:

rel Určuje vztah mezi aktuálním dokumentem a propojeným souborem. Pro CSS musí být hodnota `stylesheet`.

href Určuje cestu k externímu CSS souboru. Může být relativní nebo absolutní.

html

```html
<html>
  <head>
    <link rel="stylesheet" href="styles.css">
  </head>
</html>
```

Importované styly jsou styly, které se píší do CSS souboru pomocí pravidla `@import`. Funkce `url()` slouží k určení cesty k souboru.

css

Importované styly

```css
@import url("styles.css");
```

3.5.4 Základní selektory

Selektor je část pravidla, která určuje, na jaké elementy (tagy) se má dané pravidlo aplikovat. Existuje mnoho různých selektorů, které se liší tím, jak vybírají elementy. Základu se dělí dle typu výběru – podle vlastností, podle hierarchie a podle stavu.

Selektor: Tag

Syntax: `tag`

Selektor, který vybírá všechny elementy daného tagu.

css

```css
p {
  color: red;
}
```

Selektor: Třída

Syntax: `.trida`

Selektor, který vybírá všechny elementy s danou třídou.

```css
.trida {
  color: blue;
}
```

Třídy v HTML

Třídy se v HTML zapisují pomocí atributu `class`. Jeden prvek může mít více tříd, které se oddělují mezerou. Jedna třída může být použita na více prvcích.

Selektor: Identifikátor

Syntax: `#id`

Selektor, který vybírá jeden element s daným identifikátorem.

```css
#id {
  color: green;
}
```

Identifikátory v HTML

Identifikátor se v HTML zapisuje pomocí atributu `id`. Jeden prvek může mít pouze jeden identifikátor. Jeden identifikátor může být použit pouze na jeden prvek na stránce.

Pokud jich nastavíte více, prohlížeče si většinou vyberou první.

Selektor: Všechny elementy

Syntax: *

Selektor, který vybírá všechny elementy na stránce.

```css
* {
  margin: 0;
  padding: 0;
}
```

3.5.5 Komentáře

Komentáře v CSS se zapisují pomocí `/*` a `*/`, kde mezi nimi se vše považuje za komentář.

```css
```

Komentář v CSS

```
/* Toto je komentář */
/* Může být i víceřádkový
p {
> Všimněte si, že zde je i část pravidla považována za komentář a tedy se neaplikuje.
    color: red;
}
*/
```

3.5.6 Hodnoty a jednotky

V následujících kapitolách si ukážeme řadu vlastností a jejich možných hodnot. Většina hodnot je avšak stejná a proto si je ukážeme zde.

3.5.6.1 Barvy

Nejzákladnější hodnotou je barva. Barvy se dají zapsat různými způsoby.

Název dle anglických názvů barev, předdefinované v CSS. Např. `red`, `blue`, `green`, `black`, `white` a další.

Hexadecimální zápis pomocí šestnáctkové soustavy, kde první dvě číslice jsou červená, druhé dvě zelená a poslední dvě modrá. Např. `#FF0000` je červená, `#00FF00` je zelená a další.

RGB zápis pomocí tří čísel, kde první číslo je červená, druhé zelená a třetí modrá. Čísla jsou v rozsahu 0-255. Např. `rgb(255, 0, 0)` je červená, `rgb(0, 255, 0)` je zelená a další.

RGBA zápis pomocí čtyř čísel, kde první číslo je červená, druhé zelená, třetí modrá a čtvrté je průhlednost (alfa kanál). Čísla jsou v rozsahu 0-255 pro RGB a 0-1 pro alfa kanál. Např. `rgba(255, 0, 0, 1)` je červená, `rgba(0, 255, 0, 0.5)` je poloprůhledná zelená a další.

Další HSL, HSLA, LAB, LCH a další.

3.5.6.2 Číselné hodnoty

V některých vlastnostech se používají číselné hodnoty. Je možné zapisovat desetiné čísla pomocí tečky, např. `1.5`. V některých vlastnostech se používají i záporná čísla, např. `-10`.

U čísel se často používají jednotky. Některé jednotky jsou absolutní - neváží se na nic jiného, některé jsou relativní – počítají se vůči něčemu jinému.

Pixely (px) nejčastěji používaná jednotka pro velikosti na webu. Např. `16px`, `100px` a další, neodpovídá nutně fyzickým pixelům na obrazovce.

Procenta (%) relativní jednotka, která se počítá vůči rodičovskému elementu. Např. `50%` je polovina velikosti rodiče.

Em (em) relativní jednotka, která se počítá vůči velikosti písma aktuálního elementu. Např. `1.5em` je 1.5násobek velikosti písma. Může se kumulovat (pokud je element uvnitř jiného elementu s em jednotkou ve velikosti písma, počítá se vůči němu).

Rem (rem) relativní jednotka, která se počítá vůči velikosti písma kořenového elementu (obvykle `html`). Např. `2rem` je dvojnásobek velikosti písma kořene.

Viewport (vw, vh) relativní jednotky, které se počítají vůči velikosti viewportu (viditelné části okna prohlížeče). `1vw` je 1% šířky viewportu, `1vh` je 1% výšky viewportu.

Velikost (mm, cm, in) absolutní jednotky, které odpovídají fyzickým velikostem. Např. `10mm`, `1.5in` a další. Tyto jednotky se na webu používají zřídka, protože závisí na fyzické velikosti obrazovky a rozlišení.

Další pt (pointy), pc (picas) a další.

3.5.6.3 Další hodnoty

Kromě barev a číselných hodnot se používají i další hodnoty. Nejčastěji se jedná o klíčová slova – například `auto`, `none`, `solid` a další, které vždy mají specifický význam pro danou vlastnost.

Dále se používají různé funkce, které tvoří či upravují hodnoty – například `calc()`, `var()`, `rgb()` a další. O některých z nich se budeme bavit později.

3.5.7 Specifičnost a kaskáda

Specifičnost je systém, který určuje, které pravidlo se aplikuje na element, pokud na něj cílí více pravidel. Kaskáda je systém, který určuje, jak se kombinují různé styly z různých zdrojů (např. uživatelské styly, autorovy styly, výchozí styly prohlížeče). Specifičnost se počítá na základě typu selektoru. Obecně platí, že čím více specifický selektor je, tj. co určuje za požadavky na prvek, tím vyšší má specifičnost. Pozdější pravidla přepíší předchozí. Toto chování je důležité, ale více se s ním v předmětu WBA nebudeme zabývat.

3.5.8 Základní styly

Nyní si ukážeme některé základní vlastnosti, které se používají na skoro všech webových stránkách.

color: Barva textu

Určuje barvu textu elementu.

Definice hodnoty:

Jakékoliv barevné hodnoty (viz Kapitola 3.5.6.1).

css

```css
p {
  color: red;
}
```

font-size: Velikost písma

Určuje velikost písma elementu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Možné použít i klíčová slova jako `small`, `medium`, `large` a další, to ale není doporučované.

css

```css
p {
  font-size: 16px;
}
```

text-align: Zarovnání textu

Určuje zarovnání textu uvnitř elementu.

Definice hodnoty:

`left` zarovnání doleva (výchozí hodnota pro levostranné jazyky),
`right` zarovnání doprava,
`center` zarovnání na střed,
`justify` zarovnání do bloku (text je roztáhnutý tak, aby zaplnil celý řádek).

```css
p {
  text-align: center;
}
```

font-family: Rodina písma

Určuje rodinu písma (font) pro text elementu.

Definice hodnoty:

Název konkrétního fontu (např. `Arial`, `Times New Roman`, `Courier New`).

Obecné rodiny fontů (např. `serif`, `sans-serif`, `monospace`, `cursive`, `fantasy`). Například `serif` označuje písma s patkami, zatímco `sans-serif` označuje písma bez patky.

Hodnot lze zadat více, oddělené čárkou. Prohlížeč použije první dostupný font, případně obecnou rodinu. Font musí být nainstalován na zařízení uživatele, případně může být načten, viz Kapitola 5.1.

```css
p {
  font-family: "Arial", sans-serif;
}
```

font-weight: Tloušťka písma

Určuje tloušťku (nebo řez) písma elementu.

Definice hodnoty:

Klíčové hodnoty, jako `normal` (normální, výchozí hodnota), `bold` (tučné), `bolder` (tučnější než rodič), `lighter` (tenčí než rodič).

Číselné hodnoty: 100, 200, 300, 400, 500, 600, 700, 800, 900, kde 400 odpovídá `normal` a 700 odpovídá `bold`.

```css
p {
  font-weight: bold;
}
```

font-style: Styl písma

Určuje styl písma elementu.

Definice hodnoty:

`normal` normální styl (výchozí hodnota),
`italic` kurzíva,
`oblique` šikmý text (podobné kurzívě, ale méně výrazné).

```css
p {
  font-style: italic;
}
```

text-decoration: Dekorace textu

Určuje dekoraci textu elementu.

Definice hodnoty:

Určujeme buď `none` (žádná dekorace, výchozí hodnota), `underline` (podtržení), `overline` (přeškrtnutí nad textem), `line-through` (přeškrtnutí přes text) a další.

K některým hodnotám (těm, co zobrazují čáru) lze přidat další hodnoty:
Styl čáry `solid`, `dashed`, `dotted`.
Tloušťka čáry např. `1px`, `2px`, `0.1em` (jakákoliv číselná hodnota s jednotkou).
Barva čáry jakákoliv barevná hodnota.

Toto je složená vlastnost, jednotlivé vlastnosti pro dané hodnoty jsou `text-decoration-line`, `text-decoration-style`, `text-decoration-thickness` a `text-decoration-color`.

```css
p {
  text-decoration: underline 2px solid red;
}
```

background-color: Barva pozadí

Určuje barvu pozadí elementu.

Definice hodnoty:

Jakékoliv barevné hodnoty (viz Kapitola 3.5.6.1).

```css
1 div {
2   background-color: lightgray;
3 }
```

3.6 Linkování

Jeden web může obsahovat mnoho různých souborů – webových stránek. Tyto soubory se mohou nacházet v různých složkách a podadresářích. Pro přístup k těmto souborům se používají tzv. cesty.

> 🔗 **Propojení mezi dokumenty**
>
> Vzpomeňte si, že webové stránky jsou tak populární zejména proto, že umožňují snadné propojování mezi různými dokumenty.

Jedno z propojení jsme si již ukázali – jednalo se o propojení na externí CSS soubor pomocí tagu `link`, viz Kapitola 3.5.3. I uvnitř CSS používáme následující zápisy pro cesty k souborům – například obrázkům, fontům a dalším.

Pokud chceme odkázat na jinou stránku obsahově, používáme proto tag `a` (anchor).

<a>: Odkaz

Slouží k vytvoření odkazu na jiný dokument nebo část dokumentu. Odkazovaný dokument může být na stejném webu nebo na jiném webu.

Atributy:

href Určuje URL adresu, na kterou odkaz směřuje. Může být absolutní nebo relativní.

target Určuje, kde se má odkaz otevřít. Například `_blank` otevře odkaz v novém okně nebo záložce.

rel Určuje vztah mezi aktuálním dokumentem a odkazovaným dokumentem. Například `noopener` a `noreferrer` se často používají s `target="_blank"` pro bezpečnostní důvody.

```html
1 <a href="https://example.com">Navštivte Example.com</a>
```

Pokud je stránka z jiného webu, odkazujeme pomocí celé URL adresy, viz Kapitola 2.1.4. Pokud odkazujeme na aktuální web, můžeme použít relativní cesty.

3.6.1 Relativní cesty

Relativní cesta je cesta, která je relativní k aktuální otevřené stránce ve webovém prohlížeči. Relativní cesta může začínat několika způsoby:

- Přímým názvem souboru či složky, což znamená, že cesta je relativní k aktuální složce.
- Pomocí `./`, což znamená, že cesta je relativní k aktuální složce (stejné jako předchozí).
- Pomocí `../`, což znamená, že cesta je relativní k nadřízené složce (o úroveň výš).

Relativní cesty se tedy mění podle toho, kde se aktuální stránka nachází. To je často velmi užitečné, protože pokud přeneseme celý web na jinou doménu, relativní cesty zůstanou funkční. Naopak ale mohou být velmi nepřehledné, pokud je web složitější a má mnoho podadresářů. Nejlepší praxí je používat absolutní cesty, které se váží na kořenovou složku webu (začínají `/`)[12].

Příklad relativních cest:

- `obrazky/obrazek.jpg`: odkaz na soubor `obrazek.jpg` ve složce `obrazky`, která je v aktuální složce.
- `./obrazky/obrazek.jpg`: stejné jako předchozí.
- `../obrazky/obrazek.jpg`: odkaz na soubor `obrazek.jpg` ve složce `obrazky`, která je v nadřazené složce.

3.6.2 Absolutní cesty

Absolutní cesta je cesta, která začíná od kořenové složky webu. Často absolutní cesty značíme i celé URL adresy. Zapisujeme je pomocí `/`, což znamená, že cesta je relativní k kořenové složce[13] webu[14] (nejvyšší úroveň), často se jedná o doménu.

Příklad absolutních cest:

- `/obrazky/obrazek.jpg`: odkaz na soubor `obrazek.jpg` ve složce `obrazky`, která je v kořenové složce webu.
- `https://example.com/obrazky/obrazek.jpg`: odkaz na soubor `obrazek.jpg` ve složce `obrazky` na webu `example.com`.
- `https://example.com/`: odkaz na kořenovou stránku webu `example.com`.
- `/`: odkaz na kořenovou stránku aktuálního webu.

3.6.3 Výchozí stránky

Pokud odkazujeme na složku a otevřeme ji v prohlížeči, tak webový server se pokusí najít výchozí stránku v této složce. O těchto souborech jsme se bavili, viz Kapitola 3.3

Nejběžnější výchozí stránkou jsou soubory, které mají název `index`. Server poté dává prioritu jednotlivým příponám, které umí zpracovat. Nejběžnější přípony jsou `.html`, `.htm` a `.php`.

Pokud server nenajde žádnou výchozí stránku, zobrazí buď seznam souborů ve složce (pokud je to povoleno), nebo chybu 403 (zakázáno) či 404 (nenalezeno).

[12] Linkování vůči kořenové složce je často problémové, pokud je web uložen v podadresáři na serveru. V takovém případě je potřeba buď používat relativní cesty vůči aktuální složce, nebo nastavit správně základní URL pomocí tagu `<base>`.

[13] Pozor, že pokud otevíráme stránku například pouze jako soubor v prohlížeči, bude se brát v úvahu kořenová složka disku – např. „C://“.

[14] Vzpomeňte si na URL a jeho „umístění na serveru“.

3.6.4 Návěstí

Uvnitř URL můžeme definovat návěstí (anchor, kotvu), viz Kapitola 2.1.4 Odtuď víme, že návěstí je část URL, která začíná znakem # a následuje za ní název návěstí. Návěstí slouží k odkázání na konkrétní část stránky.

Uvnitř HTML dokumentu se návěstí definuje pomocí atributu `id` na jakémkoliv elementu. Při návštěvě URL s návěstím se prohlížeč automaticky posune na element s odpovídajícím `id`. Návěstí můžeme použít i v odkazech na dané stránce pro snadnou navigaci.

Definice návěstí v HTML

```html
<h2 id="sekce1">Sekce 1</h2>
<p>Obsah sekce 1...</p>
```

Na výše uvedeném příkladu je návěstí `sekce1`. V odkazu bychom na tuto část stránky odkazovali pomocí `#sekce1`.

Odkaz na návěstí v HTML

```html
<a href="#sekce1">Přejít na Sekci 1</a>
```

V základním nastavení je to „přeskok“ na danou část stránky. Pokud chceme, aby byl přechod plynulý, musíme použít CSS vlastnost `scroll-behavior`.

scroll-behavior: Chování posunu

Určuje, jakým způsobem se má stránka posunout na danou část při použití návěstí nebo programového posunu.

Definice hodnoty:

`auto` výchozí chování, okamžitý posun bez animace.
`smooth` plynulý posun s animací.

```css
html {
  scroll-behavior: smooth;
}
```

I pokud odkazujeme na jinou stránku, můžeme použít návěstí.

3.6.5 Speciální odkazy

Pro lepší uživatelský zážitek můžeme použít speciální typy odkazů, které mají předdefinované chování v prohlížeči. Často se jedná o otevření specifické aplikace nebo akce.

V HTML máme několik takovýchto speciálních odkazů[15]. Pokud prohlížeč tento typ odkazu podporuje, nabídne se uživateli k použití.

[15] Mimo uvedené se jedná ještě o `javascript`, případně si každá webová stránka může definovat vlastní schéma.

Pokud uživatel nemá potřebnou aplikaci nainstalovanou, prohlížeč obvykle zobrazí chybovou zprávu nebo neprovede žádnou akci. Na desktopu nahrazují telefonní aplikace různé jiné aplikace – například Skype, WhatsApp, Zoom a další.

3.6.5.1 E-mailové odkazy

E-mailové odkazy slouží k otevření výchozí e-mailové aplikace s předvyplněnou adresou příjemce či dalšími informacemi.

html
E-mailový odkaz
```
1 <a href="mailto:info@example.com?subject=Dotaz&body=Dotaz...">Napsat e-
  mail</a>
```

Za `mailto:` následuje e-mailová adresa příjemce. Poté můžeme přidat parametry pomocí `?`, kde jednotlivé parametry se oddělují pomocí `&`, stejně jako v URL. Mezi běžné parametry patří:

- `subject`: předmět e-mailu,
- `body`: tělo e-mailu,
- `cc`: kopie na další e-mailovou adresu,
- `bcc`: skrytá kopie na další e-mailovou adresu.

3.6.5.2 Telefonní odkazy

Telefonní odkazy slouží k otevření výchozí telefonní aplikace s předvyplněným telefonním číslem.

html
Telefonní odkaz
```
1 <a href="tel:+420123456789">Zavolat</a>
```

Za `tel:` následuje telefonní číslo v mezinárodním formátu (s předvolbou země, např. +420 pro Českou republiku).

3.6.5.3 SMS odkazy

SMS odkazy slouží k otevření výchozí SMS aplikace s předvyplněným telefonním číslem a textem zprávy. Telefonní číslo je opět v mezinárodním formátu a není povinné.

html
SMS odkaz
```
1 <a href="sms:+420123456789?body=Ahoj!">Poslat SMS</a>
```

3.7 Zobrazení uvnitř CSS

V CSS rozdělujeme zobrazení na různé typy. Způsob, jakým se blok vykresluje a chová v rozložení stránky, určuje vlastnost `display`.

display: Zobrazení

Určuje, jakým způsobem se má element vykreslit a chovat v rámci rozložení stránky.

Definice hodnoty:

`block` element se vykreslí jako blokový (zabírá celou šířku rodiče a začíná na novém řádku). Můžeme u něj nastavovat šířku a výšku. Např. `<div>`, `<p>`, `<h1>` a další.
`inline` element se vykreslí jako řádkový (zabírá pouze potřebnou šířku a nezasahuje na nový řádek). Např. `<span>`, `<a>`, `<strong>` a další.
`inline-block` kombinuje vlastnosti blokového a řádkového zobrazení (může mít šířku a výšku, ale nezasahuje na nový řádek).
`none` element se nevykreslí (je skrytý a nezabírá žádné místo).

Dále existují i další hodnoty jako `flex`, `grid`, `table` a další, které slouží pro pokročilejší rozložení. O některých z nich si povíme později.

css
```
1 .container {
2   display: inline-block;
3 }
```

width: Šířka

Určuje šířku elementu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči rodičovskému elementu. Klíčová slova jako `auto` (automatická šířka, výchozí hodnota) a `max-content` (maximální šířka podle obsahu).

css
```
1 .container {
2   width: 100%;
3 }
```

height: Výška

Určuje výšku elementu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči rodičovskému elementu. Klíčová slova jako `auto` (automatická výška, výchozí hodnota) a `max-content` (maximální výška podle obsahu).

css
```
1 .container {
2   height: 200px;
3 }
```

Velmi podobně můžeme prvky na stránce skrývat pomocí jiných vlastností – například `visibility` a `opacity`.

visibility: Viditelnost

Určuje, zda je element viditelný nebo skrytý.

Definice hodnoty:

`visible` element je viditelný (výchozí hodnota).
`hidden` element je skrytý, ale stále zabírá místo v rozložení. Nelze kopírovat ani interagovat s ním.
`collapse` pro tabulky a jejich řádky/sloupce – skryje je a nezabírá místo (chová se jako `display: none`), viz Kapitola 3.15.

css
```
1 .container {
2   visibility: hidden;
3 }
```

opacity: Průhlednost

Určuje průhlednost elementu.

Definice hodnoty:

Číselné hodnoty od `0` (zcela průhledný) do `1` (zcela neprůhledný, výchozí hodnota).

S obsahem lze, i s hodnotou `0`, stále interagovat a kopírovat, ale není viditelný.

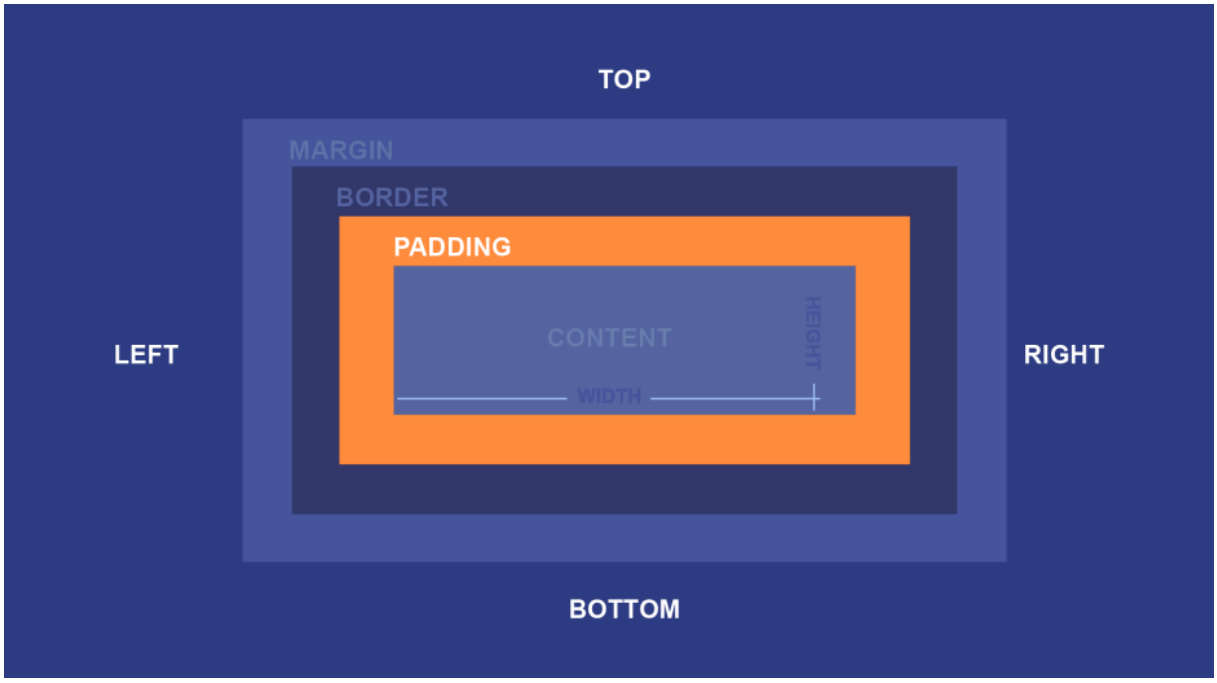
css
```
1 .container {
2   opacity: 0.5;
3 }
```

3.7.1 Box model

Každý element na webové stránce je reprezentován jako obdélníkový box. Tento box se skládá ze čtyř částí:

- Obsah (content): samotný obsah elementu (text, obrázky, atd.).
- Vnitřní okraje (padding): prostor mezi obsahem a okrajem boxu.
- Okraje (border): rámeček kolem boxu.
- Vnější okraje (margin): prostor mezi boxem a okolními elementy[16].

[16]Okraje mezi sousedními elementy se nesčítají, ale bere se větší z nich. Toto chování se nazývá „margin collapsing“.



Obrázek 1: Reprezentace box modelu

Tyto části dohromady tvoří tzv. box model. Box model je důležitý pro pochopení, jak se elementy na stránce zobrazují a jak se jejich velikost počítá.

To, jak se počítá celková velikost elementu, závisí na vlastnosti `box-sizing`.

box-sizing: Způsob počítání velikosti boxu

Určuje, jakým způsobem se má počítat velikost elementu (šířka a výška).

Definice hodnoty:

`content-box` výchozí hodnota, šířka a výška se počítají pouze z obsahu. Vnitřní okraje a okraje se přidávají navíc.

`border-box` šířka a výška se počítají včetně vnitřních okrajů a okrajů. To znamená, že pokud nastavíme šířku na `100px` a máme `10px` vnitřní okraje a `2px` okraje, celková šířka boxu bude stále `100px`, místo na obsah bude `76px`.

```css
.container {
  box-sizing: border-box;
}
```

Obecně je doporučováno používat `border-box`, protože je více intuitivní a usnadňuje práci s rozložením. V základu však není nastaven a proto je nutné ho nastavit na všechny prvky na stránce ručně – to dělá například tzv. reset, viz Kapitola 3.12.

3.7.2 Odsazení

Odsazení je prostor mezi buď:

- Obsahem a okrajem boxu (vnitřní okraje, padding),
- Okrajem boxu a okolními elementy (vnější okraje, margin).

Odsazení se nastavuje pomocí vlastností `padding` a `margin`.

Vnitřní odsazení způsobí, že obsah elementu není přímo u okraje boxu, ale je od něj odsazený. Všechny vnitřní šířky a výšky poté počítají s tímto odsazením.

Vnější odsazení způsobí, že element není přímo u sousedních elementů, ale je od nich odsazený.

Na každé straně boxu (nahoře, vpravo, dole, vlevo) můžeme nastavit různé hodnoty.

padding: Vnitřní okraje

Určuje vnitřní okraje (padding) elementu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči šířce rodičovského elementu.

Můžeme definovat až čtyři hodnoty najednou:
Jedna hodnota nastaví stejnou hodnotu pro všechny čtyři strany.
Dvě hodnoty první hodnota pro horní a dolní stranu, druhá pro pravou a levou stranu.
Tři hodnoty první hodnota pro horní stranu, druhá pro pravou a levou stranu, třetí pro dolní stranu.
Čtyři hodnoty[17] první hodnota pro horní stranu, druhá pro pravou stranu, třetí pro dolní stranu, čtvrtá pro levou stranu.

```css
.container {
  padding: 10px;
}
.container2 {
  padding: 10px 20px;
  padding-top: 5px;
}
```

margin: Vnější okraje

Určuje vnější okraje (margin) elementu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči šířce rodičovského elementu. Klíčová slova jako `auto`, která nastaví automatické odsazení (často se používá pro horizontální centrování blokových elementů).

Můžeme definovat až čtyři hodnoty najednou:
Jedna hodnota nastaví stejnou hodnotu pro všechny čtyři strany.
Dvě hodnoty první hodnota pro horní a dolní stranu, druhá pro pravou a levou stranu.
Tři hodnoty první hodnota pro horní stranu, druhá pro pravou a levou stranu, třetí pro dolní stranu.

[17] Strany jdou po směru hodin a začínají nahoře.

Čtyři hodnoty první hodnota pro horní stranu, druhá pro pravou stranu, třetí pro dolní stranu, čtvrtá pro levou stranu.

```css
.container {
  margin: 2em;
}
.container2 {
  margin-bottom: 1em;
}
```

3.7.3 Okraje

Okraje jsou rámečky kolem boxu. Okraje se nastavují pomocí vlastností `border`. Na každé straně boxu (nahoře, vpravo, dole, vlevo) můžeme nastavit různé hodnoty.

Dle nastavení vlastnosti `box-sizing` se okraje buď přidávají k šířce a výšce boxu (content-box), nebo jsou zahrnuty v šířce a výšce boxu (border-box).

border: Okraje

Určuje okraje (border) elementu.

Definice hodnoty:

Šířka okraje jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2), např. `1px`, `0.1em` a další.
Styl okraje klíčová slova jako `none` (žádný okraj, výchozí hodnota), `solid` (plný okraj), `dashed` (čárkovaný okraj), `dotted` (tečkovaný okraj) a další.
Barva okraje jakékoliv barevné hodnoty (viz Kapitola 3.5.6.1).

Můžeme nastavit i jednotlivé vlastnosti pro každou stranu zvlášť:
`border-top`, `border-right`, `border-bottom`, `border-left` nastavují celý okraj pro danou stranu.
`border-width`, `border-style`, `border-color` nastavují šířku, styl a barvu okraje pro všechny strany.
`border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width` nastavují šířku okraje pro danou stranu, možné strany jsou `right`, `left`, `bottom`.

```css
.container {
  border: 2px solid black;
}
```

Jednotlivé rohy okraje můžeme zaoblit pomocí vlastnosti `border-radius`. To způsobí, že rohy okraje nejsou ostré, ale zaoblené podle nastavené hodnoty.

border-radius: Zaoblení rohů okraje

Určuje zaoblení rohů okraje (border) elementu. Obecně tvoří kruhový tvar, ale je možné vytvořit i eliptický tvar pomocí lomítka (`/`).

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2), např. `5px`, `0.5em` a další. Můžeme použít i procenta (%), která se počítají vůči šířce a výšce elementu (např. `50%` vytvoří kruhový roh, pokud je element čtvercový).

Můžeme definovat až čtyři hodnoty najednou:
Jedna hodnota nastaví stejné zaoblení pro všechny čtyři rohy.
Dvě hodnoty první hodnota pro horní levý a dolní pravý roh, druhá pro horní pravý a dolní levý roh.
Tři hodnoty první hodnota pro horní levý roh, druhá pro horní pravý a dolní levý roh, třetí pro dolní pravý roh.
Čtyři hodnoty první hodnota pro horní levý roh, druhá pro horní pravý roh, třetí pro dolní pravý roh, čtvrtá pro dolní levý roh.

Můžeme nastavovat i jednotlivé vlastnosti pro každý roh zvlášť:
- `border-top-left-radius`,
- `border-top-right-radius`,
- `border-bottom-right-radius`,
- `border-bottom-left-radius`.

```css
.container {
  border-radius: 10px;
}
```

3.7.4 Ohraničení

Ohraničení je čára, která se zobrazuje uvnitř okraje boxu. Ohraničení se nastavuje pomocí vlastností `outline`. Ohraničení není součástí box modelu a neovlivňuje velikost boxu nehledě na nastavení `box-sizing`. Ohraničení se často používá pro zvýraznění elementu, například při fokusu (zaměření), viz později, na prvek se vstupem z klávesnice nebo myši. Oproti okraji nelze nastavovat jednotlivé strany, ohraničení je vždy kolem celého boxu.

outline: Ohraničení

Určuje ohraničení (outline) elementu.

Definice hodnoty:

Šířka ohraničení jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2), např. `1px`, `0.1em` a další.
Styl ohraničení klíčová slova jako `none` (žádné ohraničení, výchozí hodnota), `solid` (plné ohraničení), `dashed` (čárkované ohraničení), `dotted` (tečkované ohraničení) a další.
Barva ohraničení jakékoliv barevné hodnoty (viz Kapitola 3.5.6.1).

Můžeme nastavit i jednotlivé vlastnosti pro ohraničení zvlášť:
`outline-width` nastavuje šířku ohraničení.
`outline-style` nastavuje styl ohraničení.
`outline-color` nastavuje barvu ohraničení.
`outline-offset` nastavuje odsazení ohraničení od okraje boxu (může být kladné i záporné).

css
```
.container {
  outline: 2px solid red;
}
```

3.8 Pokročilé selektory

Dříve, viz Kapitola 3.5.4, jsme si ukázali základní selektory, které vybírají elementy podle jejich typu či vlastností. Nyní si ukážeme pokročilejší selektory, které nám umožňují vybírat elementy podle jejich vztahu k jiným elementům (tzv. hiearchie). Tyto selektory nám umožňují vytvářet složitější a specifičtější pravidla pro stylování.

Selektor: Potomek

Syntax: `E F`

Vybere všechny elementy `F`, které jsou potomky elementu `E` (tedy jsou uvnitř `E` na jakékoliv úrovni). `E` a `F` jsou zde zástupné znaky pro jakékoliv jiné selektory.

Například struktura, která následující selektor odpovídá:

html
```
<div>
  <p>Text</p> <!-- Bude vybrán -->

  <article>
    <p>Text</p> <!-- Bude vybrán -->
  </article>
</div>
```

css
```
div p {
  color: blue;
}
```

Selektor: Přímý potomek

Syntax: `E > F`

Vybere všechny elementy `F`, které jsou přímými potomky elementu `E` (tedy jsou uvnitř `E`, ale pouze na první úrovni). `E` a `F` jsou zde zástupné znaky pro jakékoliv jiné selektory.

Například struktura, která následující selektor odpovídá:

```html
<div>
  <p>Text</p> <!-- Bude vybrán -->

  <article>
    <p>Text</p> <!-- *Ne*bude vybrán -->
  </article>
</div>
```

```css
div > p {
  color: blue;
}
```

Selektor: Seskupovací

Syntax: `E, F`

Vybere všechny elementy, které odpovídají selektoru `E` nebo selektoru `F`. `E` a `F` jsou zde zástupné znaky pro jakékoliv jiné selektory.

```css
h1, h2 {
  font-family: Arial, sans-serif;
}
```

Seskupovací s ostatními selektory

Častá chyba studentů je to, že si myslí, že seskupovací selektor lze používat i s ostatními selektory, například:

```css
Špatné použití seskupovacího selektoru
div, .container > p
```

Tohle při špatném pochopení může vypadat, že vybírá všechny `div` a prvky s třídou `container`, které mají přímého potomka `p`. Ve skutečnosti ale vybírá všechny `div` a všechny `p`, které jsou přímými potomky jakéhokoliv elementu s třídou `container`. Seskupovací selektor vždy rozděluje celé selektory, nikoliv jednotlivé části selektorů.

3.8.1 Pseudotřídy

Nyní si ukážeme selektory, které vybírají elementy na základě jejich stavu nebo pozice v rámci rodičovského elementu. Tyto selektory se nazývají pseudotřídy a zapisují se pomocí dvojtečky (`:`) s následujícím názvem pseudotřídy. Mohou mít i parametry, které se zapisují v závorkách.

Selektor: Při najetí myši

Syntax: `E:hover`

Vybere všechny elementy `E`, na které je právě najeta myš[18]. `E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

```css
button:hover {
  background-color: lightblue;
}
```

Selektor: Odkaz

Syntax: `E:link`

Vybere všechny elementy `E`, které jsou odkazy a ještě nebyly navštíveny. `E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

```css
a:link {
  color: blue;
}
```

Selektor: Navštívený odkaz

Syntax: `E:visited`

Vybere všechny elementy `E`, které jsou odkazy a byly již navštíveny. `E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

```css
a:visited {
  color: purple;
}
```

Selektor: Aktivní stav

Syntax: `E:active`

Vybere všechny elementy `E`, které jsou právě aktivní (například při kliknutí, resp. držení, myši). `E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

[18] Toto je jeden z prvních momentů, kde se nám projeví význam specificity a kaskády CSS. Nyní záleží, kde uvedeme styly tagu button a kde styly při najetí. Bude vždy platit s větší prioritou ten, který je uveden později.

```css
a:active {
  color: red;
}
```

Selektor: Fokus

Syntax: `E:focus`

Vybere všechny elementy `E`, které jsou právě ve stavu fokusu (zaměření), například při kliknutí myši nebo navigaci pomocí klávesnice (tabu). `E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

```css
input:focus {
  outline: 2px solid blue;
}
```

Selektor: První potomek

Syntax: `E:first-child`

Vybere všechny elementy `E`, které jsou prvními potomky[19] svého rodičovského elementu. `E` je zde zástupný znak pro jakýkoliv jiný selektor.

Podobně funguje `E:last-child` pro výběr posledního potomka.

```css
p:first-child {
  font-weight: bold;
}
```

Selektor: N-tý potomek

Syntax: `E:nth-child(n)`

Vybere všechny elementy `E`, které jsou n-tými potomky svého rodičovského elementu. `n` může být číslo (1, 2, 3, ...), klíčová slova `odd` (liché) a `even` (sudé), nebo vzorec ve formátu `an+b`, kde `a` a `b` jsou celá čísla (např. `2n+1` pro liché, `2n` pro sudé, `3n` pro každého třetího atd.). `E` je zde zástupný znak pro jakýkoliv jiný selektor.

Podobně funguje `E:nth-last-child(n)` pro výběr n-tého potomka od konce.

```css
li:nth-child(odd), li:nth-child(4n), li:nth-child(6) {
  background-color: #f0f0f0;
```

[19] Častý problém bývá, že si studenti myslí, že to vybírá první prvek uvnitř daného tagu, na který pseudutřídu dáme, to není pravda – pseudotřída popisuje stav daného selektoru.

```
3 }
```

Selektor: N-tý potomek daného typu

Syntax: `E:nth-of-type(n)`

Vybere všechny elementy `E`, které jsou n-tými potomky svého rodičovského elementu, přičemž se počítají pouze elementy stejného typu jako `E`. `n` může být číslo (1, 2, 3, ...), klíčová slova `odd` (liché) a `even` (sudé), nebo vzorec ve formátu `an+b`, kde `a` a `b` jsou celá čísla (např. `2n+1` pro liché, `2n` pro sudé, `3n` pro každého třetího atd.). `E` je zde zástupný znak pro jakýkoliv jiný selektor.

Podobně funguje `E:nth-last-of-type(n)` pro výběr n-tého potomka daného typu od konce.

```css
p:nth-of-type(2), p:nth-of-type(4n) {
  color: green;
}
```

Selektor: První daného typu

Syntax: `E:first-of-type`

Vybere všechny elementy `E`, které jsou prvními potomky svého rodičovského elementu, přičemž se počítají pouze elementy stejného typu jako `E`. `E` je zde zástupný znak pro jakýkoliv jiný selektor.

Podobně funguje `E:last-of-type` pro výběr posledního potomka daného typu.

```css
p:first-of-type {
  font-style: italic;
}
```

3.8.2 Násobné selektory

Jak jste si již mohli všimnout, tak selektory můžeme kombinovat. Například můžeme vybrat všechny `p` elementy, které jsou uvnitř `div` elementu, které mají třídu `highlight`. To uděláme tak, že spojíme více selektorů dohromady. Skoro každý typ lze kombinovat s jiným[20].

Důležité pro nás je, abyste jakýkoliv selektor uměli přečíst[21] a pochopit, co vybírá. Proto následuje několik příkladů s vysvětlením.

[20] Hodně jde o zkušenosti co dává smysl a co ne.

[21] Skoro všechny selektory se vždy čtou zprava doleva, resp. zcela vpravo bývá to, co se vybírá za elementy a předcházejí omezení.

`div.highlight p`

Tento selektor vybere všechny `p` elementy, které jsou potomky (na jakékoliv úrovni) `div` elementů, které mají třídu `highlight`.

`div > p.highlight`

Tento selektor vybere všechny `p` elementy, které jsou přímými potomky `div` elementů a mají třídu `highlight`.

`a.button:hover`

Tento selektor vybere všechny `a` elementy, které mají třídu `button` a na které je právě najeta myš.

`div p span:hover`

Tento selektor vybere všechny `span` elementy, které jsou potomky `p` elementů, které jsou potomky `div` elementů, a na které je právě najeta myš.

`div#main:hover hr`

Tento selektor vybere všechny `hr` elementy, které jsou potomky (na jakékoliv úrovni) `div` elementů s ID `main`, na které je právě najeta myš.

`#header, #main > div .highlight`

Tento selektor vybere všechny elementy s ID `header` a všechny elementy s třídou `highlight`, které jsou potomky (na jakékoliv úrovni) `div` elementů, které jsou přímými potomky elementu s ID `main`.

3.9 Bezvýznamové tagy

Uvnitř HTML často potřebujeme seskupit nebo označit nějaký obsah, aniž bychom mu chtěli dávat nějaký speciální význam. Jednoduše to děláme z designových důvodů – a to tehdy, pokud CSS neumí vybrat dané prvky jinak.

Pro tento účel máme v HTML dva speciální tagy – `div` a `span`.

** : Blokový kontejner**

Slouží k seskupení blokových elementů. Je to bezvýznamový kontejner, který nemá žádný speciální význam. Používá se hlavně pro účely stylování a rozložení pomocí CSS.

```html
<div class="container">
  <p>Odstavec textu...</p>
</div>
```

: Řádkový kontejner

Slouží k seskupení řádkových elementů. Je to bezvýznamový kontejner, který nemá žádný speciální význam. Používá se hlavně pro účely stylování a rozložení pomocí CSS.

```html
```

```html
1 <span class="highlight">Tento text je zvýrazněný.</span>
```

Kdy používat div a span

Využití těchto tagů by mělo být omezeno na nevyřešitelné situace. Pokud je možné použít sémantický tag, je nutné ho použít. Nadměrné používání `div` a `span` vede k tzv. „div soup“ (polévce z divů), což ztěžuje čitelnost a údržbu kódu. Dále je těžké pro vyhledávače a asistivní technologie pochopit strukturu stránky.

3.10 Naformátovaný text

Často potřebujeme na stránku vkládat text, který nechceme, aby prohlížeč sám upravoval – například aby zalamoval řádky, měnil mezery a další. Určitě jste si všimli, že pokud do HTML vložíte více mezer nebo nové řádky, prohlížeč je ignoruje a zobrazí pouze jednu mezeru. Pro tento účel máme v HTML speciální tag `pre` (preformatted text), který zajistí, že text uvnitř bude zobrazen přesně tak, jak je napsán v kódu[22].

<pre>: Naformátovaný text

Slouží k zobrazení textu přesně tak, jak je napsán v kódu. Prohlížeč zachová všechny mezery, nové řádky a další formátování. Text uvnitř `pre` je obvykle zobrazen pomocí monospaced písma (každý znak má stejnou šířku).

```html
1 <pre>
2   Toto je naformátovaný text.
3   Mezery a     nové řádky jsou zachovány.
4 </pre>
```

Pokud uvnitř `pre` potřebujeme použít HTML značky, musíme speciální znakům vynutit zobrazení pomocí HTML entit, viz Kapitola 3.1.3.

Naformátovaný text s HTML značkami

```html
1 <pre>
2   &lt;div&gt;Toto je naformátovaný text.&lt;/div&gt;
3   Mezery a     nové řádky jsou zachovány.
4 </pre>
```

Pokud se sémanticky jedná o kód, je mimo pre vhodné použít tag `code`.

[22] To znamená, že se zanechají i formátovadí taby či mezery, často je nutné odsadit kód uvnitř tagu zcela na začátek řádků.

<code>: Kód

Slouží k označení kusu kódu nebo programovacího jazyka. Obvykle se zobrazuje pomocí monospaced písma (každý znak má stejnou šířku).

```html
1 <code><pre>console.log("Hello, world!");</pre></code>
```

3.11 Další textové vlastnosti

text-transform: Transformace textu

Určuje, jakým způsobem se má text transformovat (změnit velikost písmen).

Používáme, pokud je gramaticky správně napsaný text, ale z designových důvodů chceme, aby vypadal jinak (například všechny velké znaky).

Definice hodnoty:

`none` žádná transformace (výchozí hodnota).
`capitalize` první písmeno každého slova bude velké.
`uppercase` všechny písmena budou velká.
`lowercase` všechny písmena budou malá.

```css
1 h1, h2, h3, h4, h5, h6 {
2   text-transform: uppercase;
3 }
```

text-align-last: Zarovnání posledního řádku textu

Určuje zarovnání posledního řádku textu v bloku. Používáme, pokud chceme mít jinak zarovnaný poslední řádek než ostatní řádky.

Definice hodnoty:

`auto` výchozí hodnota, prohlížeč rozhodne podle jazyka a směru textu.
`left` zarovná poslední řádek doleva.
`right` zarovná poslední řádek doprava.
`center` zarovná poslední řádek na střed.
`justify` roztáhne poslední řádek tak, aby zaplnil celou šířku (pokud je to možné).

```css
1 p {
2   text-align: justify;
3   text-align-last: left;
4 }
```

text-indent: Odsazení prvního řádku

Určuje odsazení prvního řádku textu v bloku.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči šířce rodičovského elementu.

```css
p {
  text-indent: 2em;
}
```

letter-spacing: Mezery mezi písmeny

Určuje mezery mezi jednotlivými písmeny v textu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči aktuální velikosti písma. Můžeme použít i záporné hodnoty pro zmenšení mezer.

```css
h1, h2, h3, h4, h5, h6 {
  letter-spacing: 0.05em;
}
```

word-spacing: Mezery mezi slovy

Určuje mezery mezi jednotlivými slovy v textu.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2). Můžeme použít i procenta (%), která se počítají vůči aktuální velikosti písma. Můžeme použít i záporné hodnoty pro zmenšení mezer.

```css
p {
  word-spacing: 0.1em;
}
```

3.12 Resetování stylů

Jak jsme zjistili při naší první práci s prohlížečem a HTML, tak prohlížeče aplikují základní styly. Na každý prvek má prohlížeč nastavené styly, které si myslí, že jsou z jeho sémantického významu vhodné.

Pro nás jsou tyto styly zcela nevhodné. Protože nám rozbijí styly, které aplikuje mi. Když tvoříme stránku, tak v našem prohlížeči počítáme se základními styly – ty nutně nenastavujeme, protože nás to ani nenapadne. Každý prohlížeč[23] má své styly, které se velmi liší. Pokud bychom tento problém nevyřešili, stránky by se nám vždy zobrazovali jinak.

Jeden ze způsobů jak vyřešit tento problém je tzv. resetování stylů. Resetování stylů znamená, že na začátku našeho CSS souboru nastavíme všechny vlastnosti na výchozí hodnoty, resp. hodnoty, se kterými budeme počítat. Díky vlastnosti kaskády[24] se styly prohlížeče přepíší těmi našimi.

Existuje mnoho různých resetů, které můžeme použít. V základu ale dělají totéž.

css

Jednoduchý reset

```css
*, *::before, *::after {
    box-sizing: border-box;
}
* {
    margin: 0;
}
body {
    line-height: 1.5;
    -webkit-font-smoothing: antialiased;
}
img, picture, video, canvas, svg {
    display: block;
    max-width: 100%;
}
input, button, textarea, select {
    font: inherit;
}
p,h1, h2, h3, h4, h5, h6 {
    overflow-wrap: break-word;
}
p {
    text-wrap: pretty;
}
h1, h2, h3, h4, h5, h6 {
    text-wrap: balance;
}
```

Některé ze selektorů, vlastností a tagů jsme si již ukázali. Není pro nás aktuálně důležité jim rozumět do detailu. Na mém studijním portálu naleznete odkaz na sdílený reset, který můžete použít např. ve svých úkolech. Reset je ale nutné používat ve všech projektech.

[23] Některé dovolují psát i vlastní CSS uživatelům.

[24] A specifity selektorů.

3.13 Obrázky

Obrázky na webových stránkách jsou jeden z nejdůležitějších prvků. Dovolují totiž uživateli lépe pochopit obsah stránky, zaujmout ho a udělat stránku vizuálně atraktivní. Říká se, že obrázky řeknou více než tisíc slov.

Na webu (a mimo něj) se setkáme s různými typy ukládání obrázkových dat a samotných formátů. Nejčastěji rozdělujeme dva typy obrázků:

- **Rastrové obrázky** které ukládají tzv. bitmapu – což je matice bodů (pixelů), kde každý bod má svou barvu.
- **Vektorové obrázky** které ukládají matematický popis tvarů, čar a barev.

Obecně platí, že rastrové obrázky jsou násobně velikostně větší než vektorové obrázky. Rastrové obrázky jsou vhodné pro fotografie a složité obrázky s mnoha barvami a detaily. Vektorové obrázky jsou vhodné pro jednoduché grafiky, loga, ikony a ilustrace s omezeným počtem barev a tvarů.

Důležitá vlastnost vektorových obrázků je, že jsou škálovatelné – můžeme je zvětšovat nebo zmenšovat bez ztráty kvality. Rastrové obrázky při zvětšení ztrácejí kvalitu a mohou vypadat rozmazaně nebo pixelově.

Mezi nejznámější formáty obrázků patří:

- **JPEG** (nebo JPG) – je to ztrátový formát, který je vhodný pro fotografie a obrázky s mnoha barvami. Umožňuje vysokou kompresi[25] – často určujeme kvalitu komprese v procentech (čím vyšší procento, tím menší komprese a lepší kvalita). Nedovoluje průhlednost.
- **PNG** – je to bezztrátový formát, který je vhodný pro obrázky s průhledností a ostrými hranami. Umožňuje zachovat kvalitu, ale soubory mohou být větší než JPEG.
- **GIF** – je to formát, který podporuje animace a průhlednost. Je omezen na 256 barev[26]. Můžeme mu zapnout opakování. Jedná se ale o velmi neúsporný formát, který se dnes již moc nepoužívá. Může obsahovat zcela průhledné pixely (nikoliv však s částečnou průhledností). Obecně se doporučuje místo GIF používat modernější formáty jako APNG nebo video[27].
- **SVG** – je to vektorový formát, který je vhodný pro loga, ikony a ilustrace. Umožňuje škálování bez ztráty kvality a podporuje animace a interaktivitu. Je založen na XML, takže je možné ho upravovat jako textový soubor.
- **WebP** – je to moderní formát, který podporuje jak ztrátovou, tak bezztrátovou kompresi. Je navržen pro web a nabízí lepší kompresi než JPEG a PNG při zachování kvality. Podporuje průhlednost a animace. Není ale podporován ve všech prohlížečích (např. Safari).

3.13.1 Obsahové obrázky

Do HTML lze vkládat obrázky dvěma způsoby. Ten první je jako obsah stránky.

`<img>`: Obrázek

Slouží k vložení obrázku na webovou stránku. Tento obrázek má vztah k obsahu stránky – je součástí významu stránky. Je to nepárový tag, který nemá žádný obsah.

[25] Nejčastěji se spojují podobné barvy vedle sebe

[26] Podobně jako JPG.

[27] Mezi nejznámější formáty videa patří MP4, AVI, MOV a WebM.

Atributy:

src Určuje cestu k obrázku. Může být relativní (vztahující se k umístění HTML souboru) nebo absolutní (celá URL adresa), viz Kapitola 3.6.

alt Určuje alternativní text, který se zobrazí, pokud obrázek nelze načíst. Tento text je také důležitý pro přístupnost, protože ho čte software pro zrakově hendikepované uživatele. Měl by stručně a výstižně popisovat obsah obrázku. Často využívají i vyhledávače pro indexaci obrázků.

html
```
1 <img src="obrazek.jpg" alt="Popis obrázku">
```

3.13.2 Designové obrázky

Další způsob vkládaní obrázků je vložením do CSS. Veškeré obrázky v CSS jsou určeny pro design a nesmí souviset s obsahem stránky.

Obrázky se kládají zejména jako pozadí elementů.

background-image: Obrázek na pozadí

Určuje obrázek, který se zobrazí na pozadí elementu. Obrázek může být jediný nebo více obrázků oddělených čárkou (více vrstev).

Definice hodnoty:

`url("cesta/k/obrazku.jpg")` cesta k obrázku, může být relativní (vztahující se k umístění CSS souboru) nebo absolutní (celá URL adresa), viz Kapitola 3.6.
`none` žádný obrázek (výchozí hodnota).

css
```
1 .container {
2   background-image: url("obrazek.jpg");
3 }
```

background-size: Velikost obrázku na pozadí

Určuje velikost obrázku na pozadí elementu.

Definice hodnoty:

`auto` výchozí hodnota, obrázek si zachová svou původní velikost.
`cover` obrázek se zvětší nebo zmenší tak, aby pokryl celý element, přičemž si zachová poměr stran. Některé části obrázku mohou být ořízntuty.
`contain` obrázek se zvětší nebo zmenší tak, aby byl celý viditelný uvnitř elementu, přičemž si zachová poměr stran. Mohou zůstat prázdná místa.
Číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2), např. `100px`, `50%` a další. Můžeme zadat jednu hodnotu (šířka) nebo dvě hodnoty (šířka a výška).

```css
.container {
  background-image: url("obrazek.jpg");
  background-size: cover;
}
```

background-repeat: Opakování obrázku na pozadí

Určuje, zda a jak se má obrázek na pozadí opakovat.

Definice hodnoty:

`repeat` výchozí hodnota, obrázek se opakuje jak horizontálně, tak vertikálně.
`no-repeat` obrázek se neopakuje.
`repeat-x` obrázek se opakuje pouze horizontálně.
`repeat-y` obrázek se opakuje pouze vertikálně.
`space` obrázek se opakuje tak, aby vyplnil celý element, přičemž mezi obrázky jsou mezery.
`round` obrázek se opakuje tak, aby vyplnil celý element, přičemž se může zvětšit nebo zmenšit, aby přesně zapadl.

Případně kombinace dvou hodnot (první pro horizontální opakování, druhá pro vertikální opakování).

```css
.container {
  background-image: url("obrazek.jpg");
  background-repeat: no-repeat repeat;
}
```

background-position: Pozice obrázku na pozadí

Určuje pozici obrázku na pozadí elementu.

Definice hodnoty:

Klíčová slova `left`, `center`, `right`, `top`, `bottom` a jejich kombinace (např. `top left`, `center bottom`).
Číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2), např. `10px 20px`, `50% 50%` a další. První hodnota je pro horizontální pozici, druhá pro vertikální pozici.

```css
.container {
  background-image: url("obrazek.jpg");
  background-repeat: no-repeat;
  background-position: center;
}
```

S obrázkami na pozadí můžeme dále pracovat pomocí dalších vlastností, například `background-attachment`, `background-clip` a další. O nich se ale více bavit nebudeme.

3.13.3 Optimalizace obrázků

Při práci s obrázky na webu je důležité myslet na jejich optimalizaci. Obrázky jsou obecně příliš velké soubory a každé jeho použití bychom měli zvážit. Velké obrázky zpomalují načítání stránek, což může negativně ovlivnit uživatelský zážitek. Pokud má například uživatel pomalé připojení k internetu, může trvat dlouho, než se stránka načte. Naopak když si uživatelé platí za přenesené data, tak je pro ně nevhodně velký obrázek zbytečnou zátěží.

Obecně vždy při použití obrázku řešíme:

- Jaký formát obrázku použít (viz výše),
- Jak velký[28] bude prvek (jakou bude mit velikosti) a jak velký obrázek do něj vložíme (rozměry v pixelech),
- Jak moc obrázek komprimovat (pokud je to rastrový obrázek).

Žádný limit na velikosti ani počet obrázků na stránce není. Obecně se doporučje mít nějaký horní limit (například 15 obrázků) a celkovou velikost obrázků na stránce do 5-50MB. Pokud potřebuje více místa na web obecně, je vhodné se zamyslet proč takové místo potřebujete a zda by nebylo lepší použít jiné technologie.

Obrázky prohlížeč nutně nestahuje při každé návštěvě stránky. Existuje totiž cache, což je místo, kam si prohlížeč ukládá již stažené soubory. Při další návštěvě stránky pak prohlížeč nejdříve zkontroluje, zda již daný soubor nemá v cache a pokud ano, použije ho. To, jestli seo něco do cache uloží a případně na jak dlouho můžeme určovat pomocí HTTP hlaviček a nastavit třeba na straně serveru.

Další možnosti optimalizace obrázku je tzv. lazy loading (líné načítání). To znamená, že obrázek se nestáhne hned při načtení stránky, ale až ve chvíli, kdy se má zobrazit (například když uživatel scrolluje stránku a obrázek se dostane do viditelné oblasti). To můžeme udělat pomocí atributu `loading="lazy"` u tagu `img`.

html

Lazy loading obrázku

```
1 <img src="obrazek.jpg" alt="Popis obrázku" loading="lazy">
```

Tento atribut je dnes již podporován ve většině moderních prohlížečů.

Poslední možnost je poskytnutí více verzí obrázku pro různá zařízení. Tagu `img` můžeme určit sadu adres, a každé určit podmínky, za kterých se má použít. Atribut se jmenuje `srcset` a `sizes`.

html

Více verzí obrázku pro různá zařízení

```
1 <img
2   src="obrazek-small.jpg"
3   srcset="obrazek-small.jpg 500w, obrazek-medium.jpg 1000w, obrazek-
  large.jpg 1500w"
4   sizes="(max-width: 600px) 500px, (max-width: 1200px) 1000px, 1500px"
5   alt="Popis obrázku">
```

Více o těchto a dalších možnostech optimalizace obrázků se dozvíte v dokumentaci na MDN Web Docs.

[28] Pokud máme 20kx20kpx obrázek ikony knihy, který se na stránce zobrazuje jako 10x10px, tak něco děláme špatně.

Toto ale duplikuje obrázky na serveru, ale za to ušetříme nejen uživatelům přenesená data, ale i čas na načítání stránky.

3.13.4 Zaplnění a ohraničení obrázků

Časté je, že potřebujeme obsahovým obrázkem (ten v HTML) zaplnit nějaký prostor. Na to můžeme jít různými způsoby.

První z nich je použití klíčového slova `auto` u vlastností `width`, př. `height`. Jedné z vlastností nastavíme konkrétní hodnotu (například v pevných jednotkách nebo v procentech) a druhou vlastnost necháme na `auto`. Tím zajistíme, že obrázek zaplní daný prostor a zároveň si zachová svůj poměr stran. To nutně ale nezaplní celé místo, které má.

Druhý způsob je použití vlastnosti `object-fit`.

object-fit: Zaplnění obrázku

Určuje, jakým způsobem se má obrázek přizpůsobit svému kontejneru (oblast, kterou má zaplnit). Používáme, pokud chceme mít (nejen) obrázek přesně v daném prostoru a zároveň si zachovat jeho poměr stran.

Definice hodnoty:

`fill` výchozí hodnota, obrázek se roztáhne nebo smrští tak, aby zaplnil celý kontejner, přičemž může dojít ke změně poměru stran (obrázek může být deformován).
`contain` obrázek se zvětší nebo zmenší tak, aby byl celý viditelný uvnitř kontejneru, přičemž si zachová poměr stran. Mohou zůstat prázdná místa.
`cover` obrázek se zvětší nebo zmenší tak, aby pokryl celý kontejner, přičemž si zachová poměr stran. Některé části obrázku mohou být ořízntuty.
`none` obrázek si zachová svou původní velikost, i když to znamená, že nebude celý viditelný v kontejneru.
`scale-down` obrázek se zmenší tak, aby byl celý viditelný v kontejneru (jako `contain`), ale nikdy se nezvětší nad svou původní velikost (jako `none`).

```css
img {
  width: 300px;
  height: 200px;
  object-fit: cover;
}
```

3.13.5 Obrázkové mapy

Obrázkové mapy nám dovolují definovat různé oblasti na obrázku, které mohou mít různé odkazy. To se hodí například pro mapy, kde chceme mít různé oblasti klikací. Obrázkové mapy se definují pomocí tagu `map` a atributu `usemap` u tagu `img`.

<map>: Obrázková mapa

Slouží k definování klikacích oblastí na obrázku. Každá oblast může mít svůj vlastní odkaz a tvar.

Atributy:

name Určuje jméno mapy, které se používá v atributu `usemap` u tagu `img`.

```html
<img src="mapa.jpg" alt="Mapa" usemap="#mapa">
<map name="mapa">
  <area shape="rect" coords="34,44,270,350" href="oblast1.html" alt="Oblast 1">
  <area shape="circle" coords="337,300,44" href="oblast2.html" alt="Oblast 2">
  <area shape="poly" coords="130,150,180,200,230,150,180,100" href="oblast3.html" alt="Oblast 3">
</map>
```

<area>: Oblast na obrázkové mapě

Slouží k definování jednotlivých klikacích oblastí na obrázkové mapě. Každá oblast může mít svůj vlastní tvar, souřadnice a odkaz.

Atributy:

shape Určuje tvar oblasti. Může být `rect` (obdélník), `circle` (kruh) nebo `poly` (mnohoúhelník).

coords Určuje souřadnice oblasti v závislosti na tvaru. Pro `rect` jsou to čtyři hodnoty (levý horní roh a pravý dolní roh), pro `circle` jsou to tři hodnoty (střed a poloměr) a pro `poly` je to seznam souřadnic vrcholů.

href Určuje odkaz, na který se má přejít po kliknutí na oblast.

alt Určuje alternativní text pro oblast, který je důležitý pro přístupnost.

```html
<area shape="rect" coords="34,44,270,350" href="oblast1.html" alt="Oblast 1">
```

Obecně se ale obrázkové mapy dnes již moc nepoužívají. Co často děláme, je, že pomocí CSS umístíme jiné prvky nad obrázek a těmto prvkům dáme odkazy. To nám dovolí mnohem lépe stylovat a umisťovat klikací oblasti. Na druhou stranu ztratíme kontrolu nad tím, jak přesný je klikací prostor. Umístění budeme řešit později, viz Kapitola 5.5.

Těmto prvkům často měníme kurzor myši, aby uživatel poznal, že je prvek interaktivní.

cursor: Kurzor

Určuje vzhled kurzoru myši, když je nad daným elementem. Používáme pro zlepšení uživatelského zážitku a indikaci interaktivity.

Definice hodnoty:

`auto` výchozí hodnota, prohlížeč rozhodne podle kontextu.
`default` standardní kurzor (obvykle šipka).
`pointer` kurzor ve tvaru ruky, obvykle používaný pro odkazy.
`text` kurzor ve tvaru I, obvykle používaný pro textová pole.
`move` kurzor ve tvaru čtyřsměrné šipky, obvykle používaný pro přesouvání.
`not-allowed` kurzor ve tvaru zakázaného symbolu (kruh s přeškrtnutím), obvykle používaný pro nepovolené akce.
Další kličová slova viz MDN Web Docs.

Můžeme také použít vlastní obrázek jako kurzor pomocí `url("cesta/k/obrazku.png")`, přičemž můžeme specifikovat i pozici „klikového bodu“ kurzoru pomocí dvou číselných hodnot (s jednotkami) za URL.

css
```css
button {
  cursor: pointer;
}
```

3.14 Seznamy

Seznam je strukturovaný způsob, jak uspořádat položky do skupin. Seznamy v HTML známe tři. Jedná se (možná nečekaně) o skoro nejpoužívanější HTML elementy. Nyní si je všechny ukážeme.

3.14.1 Neuspořádaný seznam

Nejjednodušší seznam je neuspořádaný seznam. Jednotlivé položky v tomto seznamu nemají žádné pořadí. Resp. jejich pořadí není důležité pro význam seznamu. Můžeme si to představit jako seznam nákupů – tam nehraje roli, v jakém pořadí jsou položky uvedeny[29].

`<ul>`: Neuspořádaný seznam

Slouží k vytvoření neuspořádaného seznamu položek. Položky v tomto seznamu nemají žádné specifické pořadí.

Může obsahovat pouze tagy `li` jako přímé potomky.

html
```html
<ul>
  <li>Položka 1</li>
  <li>Položka 2</li>
  <li>Položka 3</li>
</ul>
```

[29] Pro někoho možná ale ano.

<li>: Položka seznamu

Slouží k definování jednotlivých položek v seznamu. Může být použit uvnitř `ul` (neuspořádaný seznam), `ol` (uspořádaný seznam) a dalších.

Může obsahovat jakýkoliv jiný HTML obsah **včetně dalších seznamů**.

Atributy:

value Tento atribut je specifický pro použití uvnitř uspořádaného seznamu (`ol`). Určuje číselnou hodnotu položky v uspořádaném seznamu. Číslo se používá i pro různá číslování položek – například i v písmeném číslování. Může být použit k přepsání výchozího číslování položek.

html
```
1 <li>Položka 1</li>
```

3.14.2 Uspořádaný seznam

Uspořádaný seznam je seznam, kde jednotlivé položky mají specifické pořadí. To znamená, že pořadí položek je důležité pro význam seznamu. Můžeme si to představit jako seznam kroků v receptu – tam je důležité, v jakém pořadí jsou kroky uvedeny.

<ol>: Uspořádaný seznam

Slouží k vytvoření uspořádaného seznamu položek. Položky v tomto seznamu mají specifické pořadí.

Může obsahovat pouze tagy `li` jako přímé potomky.

Atributy:

type Určuje typ číslování položek v seznamu. Možné hodnoty jsou:
- `1`: čísla (výchozí hodnota),
- `A`: velká písmena (A, B, C, ...),
- `a`: malá písmena (a, b, c, ...),
- `I`: velká římská čísla (I, II, III, ...),
- `i`: malá římská čísla (i, ii, iii, ...).

Typ určujeme, pokud je to důležité pro význam seznamu – například v právních dokumentech se často používají římská čísla.

start Určuje počáteční číslo pro číslování položek v seznamu. Výchozí hodnota je 1.

reversed Pokud je tento atribut přítomen, číslování položek v seznamu bude sestupné (od nejvyššího čísla k nejnižšímu).

html
```
1 <ol reversed>
2   <li>Krok 1</li>
3   <li>Krok 2</li>
```

```
4   <li>Krok 3</li>
5 </ol>
```

3.14.3 Vnořené seznamy

Jak bylo již zmíněno, položky seznamu mohou obsahovat jakýkoliv HTML obsah a to včetně dalších seznamů. Seznamy můžeme tedy vnořovat do sebe. Nezáleží na tom, zda se jedná o uspořádaný nebo neuspořádaný seznam – můžeme je kombinovat jak chceme.

html

Vnořené seznamy

```
 1 <ul>
 2   <li>Položka 1
 3     <ol>
 4       <li>Podpoložka 1.1</li>
 5       <li>Podpoložka 1.2</li>
 6     </ol>
 7   </li>
 8   <li>Položka 2
 9     <ul>
10       <li>Podpoložka 2.1</li>
11       <li>Podpoložka 2.2</li>
12     </ul>
13   </li>
14   <li>Položka 3</li>
15 </ul>
```

Důležité je, že prvky se vnořují pouze uvnitř položek seznamu (`li`), nikoliv přímo do seznamu (`ul` nebo `ol`).

3.14.4 Definiční seznam

Definiční seznam je speciální typ seznamu, který se používá k definování termínů a jejich popisů. Často používáme jako slovník nebo glosář. Tedy seznam, kde máme termín a jeho definici.

<dl>: Definiční seznam

Slouží k vytvoření definičního seznamu.

html

```
1 <dl>
2   <dt>HTML</dt>
3   <dd>HyperText Markup Language -- značkovací jazyk pro tvorbu
  webových stránek.</dd>
4   <dt>CSS</dt>
5   <dd>Cascading Style Sheets -- kaskádové styly pro úpravu vzhledu
  webových stránek.</dd>
```

```
6 </dl>
```

<dt>: Definiční termín

Slouží k definování termínu v definičním seznamu. Může být použit uvnitř `dl` (definiční seznam).

Může obsahovat jakýkoliv jiný HTML obsah.

html
```
1 <dt>HTML</dt>
```

<dd>: Definiční popis

Slouží k definování popisu termínu v definičním seznamu. Může být použit uvnitř `dl` (definiční seznam).

Může obsahovat jakýkoliv jiný HTML obsah.

html
```
1 <dd>HyperText Markup Language -- značkovací jazyk pro tvorbu webových
  stránek.</dd>
```

Pro definiční seznamy je důležité, že jeden termín může mít více popisů a naopak, tedy, že pro jeden popis může být pro více termínů.

html
Definiční seznam s více popisy pro jeden termín
```
1 <dl>
2   <dt>HTML</dt>
3   <dd>HyperText Markup Language -- značkovací jazyk pro tvorbu webových
  stránek.</dd>
4   <dd>Jazyk používaný k vytváření struktury webových stránek.</dd>
5 </dl>
```

3.14.5 Stylování seznamů

Uvnitř CSS můžeme seznamům měnit vzhled pomocí různých vlastností. Textové vlastnosti již známe.

list-style-type: Typ odrážek nebo číslování

Určuje typ odrážek pro neuspořádané seznamy (`ul`) nebo typ číslování pro uspořádané seznamy (`ol`).

Definice hodnoty:

Pro `ul` `disc` (plná tečka, výchozí hodnota), `circle` (prázdná tečka), `square` (čtverec), `none` (žádné odrážky) a další,

Pro `ol` `decimal` (číslování čísly, výchozí hodnota), `lower-alpha` (malá písmena), `upper-alpha` (velká písmena), `lower-roman` (malá římská čísla), `upper-roman` (velká římská čísla) a další.

Taktéž můžeme definovat vlastní seřazené odrážky pomocí at-pravidla `@counter-style`.

```css
ul {
  list-style-type: square;
}
```

list-style-position: Pozice odrážek nebo číslování

Určuje, zda se mají odrážky nebo číslování zobrazit uvnitř nebo vně bloku položky seznamu.

Definice hodnoty:

`inside` odrážky nebo číslování jsou zarovnány s textem položky (uvnitř bloku),
`outside` odrážky nebo číslování jsou zarovnány vlevo od textu položky (vně bloku, výchozí hodnota).

```css
ul {
  list-style-position: inside;
}
```

list-style-image: Obrázek jako odrážky

Určuje obrázek, který se použije jako odrážky pro seznam[30]

Definice hodnoty:

`url("cesta/k/obrazku.png")` cesta k obrázku, může být relativní (vztahující se k umístění CSS souboru) nebo absolutní (celá URL adresa),
`none` žádný obrázek (výchozí hodnota).

```css
ul {
  list-style-image: url("cesta/k/obrazku.png");
}
```

3.15 Tabulky

Další často používaný prvek na webu jsou tabulky. Tabulky se používají k zobrazení dat v řádcích a sloupcích. Obecně do nich tedy vkládáme soubory hodnot, které lze rozdělit do

[30] Moderní CSS má tzv. pseudotřídu `::marker`, která se používá pro změnu odrážek obecně, ale to je nad rámec WBA.

více kategorií. Jsou velmi přehledné a uživatel je zná z běžných kancelářských aplikací (Excel, Google Sheets a další).

V HTML máme tabulky velmi omezené – definujeme jim pouze obsah a strukturu, a to ve smyslu jednoho řádku a jejich buněk. Nemáme tedy moc možnost definovat jednotlivé sloupce - je to možné, ale omezené, viz Kapitola 3.15.4. To bývá problém, pokud chceme mít například první sloupec širší než ostatní. Místo jednoho sloupce vybereme všechny první buňky v řádcích a těm nastavíme šířku.

Buňky ale mohou být rozdělené do více sloupců a podobné věci, což může být někdy komplikované.

3.15.1 Historie tabulek

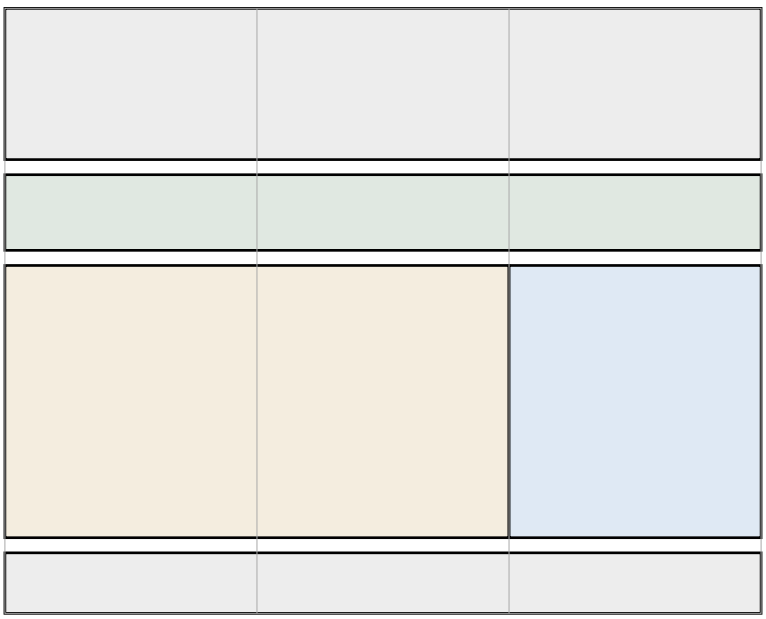
Tabulky mají hodně zvláštní historii používání na webu. Původně se tabulky používaly i pro tzv. rozložení (layout) stránek. To znamená, že se pomocí tabulek určovalo, kde na stránce bude jaký obsah.

To bylo velmi špatné řešení, protože tabulky nejsou pro tento účel vhodné. Tabulky jsou určeny pro zobrazení dat, nikoliv pro rozložení stránky. Použití tabulek pro layout mělo několik problémů:

- Nevhodně se řešila přístupnost – čtečky obrazovky měly problém s interpretací tabulek jako rozložení,
- Tabulky jsou rigidní a těžko se přizpůsobují různým velikostem obrazovek,
- Kód byl složitý a těžko se udržoval.

Jak stránky vypadaly pomocí tabulek si lze ukázat na jednoduchém příkladu. Jednoduše definujeme jednotlivé řádky a v nich buňky, které budou obsahovat různé části stránky. První řádek tedy mohla být hlavička, která obsahovalo logo. Poté následoval řádek s navigací. Poté řádek s hlavním obsahem a postranním panelem. Nakonec řádek s patičkou.

V jednotlivých řádcích se poté spojily buňky, aby se vytvořily větší oblasti. Například v řádku s obsahem jsou dva sloupce – jeden pro hlavní obsah a druhý pro postranní panel. Všechny ostatní musely být sloučeny do jedné (či více) buněk.



Obrázek 2: Rozložení pomocí tabulky

Další věc, co se v praxi používala je, že tabulky obsahovaly další tabulky. To se dělalo proto, že tabulky měly problémy s rozdělením na více řádků.

Dnes se tabulky pro rozložení nepoužívají (stále jsme v HTML, tady nám jde o obsah!) a místo toho se správně řeší rozložení až uvnitř CSS, viz Kapitola 4.2.

3.15.2 Tabulky nyní

Tabulky nejsou vhodné pro layout stránek, to jsme si řekli v předchozí kapitole. Tabulky ale nejsou responzivní a to je jejich největší problém v moderním webu. Co myslíme responzivním webem je, že se stránka přizpůsobí velikosti zařízení, na kterém je zobrazena. Tedy, že stránky vypadá jinak na mobilu, jinak na tabletu a jinak na počítači. Responzivitou se budeme zabývat později, viz Kapitola 4.4.

Tabulky sice můžeme měnit, ale v základním zobrazení tabulek jsou vždy pevné a to je problém. Nejčastější co se s tabulkami dělá, je, že se jim nastaví horizontální scrollování. To způsobí, že tabulka nepřeteče šířku rodiče, ale uživatel musí scrollovat, aby viděl celý obsah. To není na přenosných zařízeních moc uživatelsky přívětivé.

overflow: Přetečení obsahu

Určuje, jak se má chovat obsah, který přesahuje velikost svého kontejneru. Používáme pro řešení přetečení obsahu, například u tabulek.

Definice hodnoty:

`visible` výchozí hodnota, obsah není ořezán a může přesahovat velikost kontejneru,
`hidden` obsah je ořezán a přetečená část není viditelná[31],
`scroll` vždy se zobrazí scrollbary (horizontální i vertikální), i když není potřeba[32],
`auto` scrollbary se zobrazí pouze tehdy, když je obsah větší než kontejner.

Můžeme nastavit samostatně pro horizontální (`overflow-x`) a vertikální (`overflow-y`) přetečení.

```css
table {
  overflow-x: auto;
}
```

Později si ukážeme, že i data definované tabulkou můžeme zobrazit jinak než v tabulce, viz Kapitola 4.2. Tabulky nepoužíváme tedy pro rozložení (layout) stránek, ale pro určení závislostí mezi daty.

[31] Je stále možné uvnitř kontejneru scrollovat (pokud vůbec přetéká), používáme například v JavaScriptu.

[32] Některé prohlížeče tohle interpretují jako `auto`, dejte pozor, že správně poté má být vždy vidět kurzor, což často nechcete.

3.15.3 Definice tabulek

<table>: Tabulka

Hlavní prvek pro následující definice dat v tabulce. Může obsahovat tagy `caption`, `thead`, `tbody`, `tfoot` a `tr` jako přímé potomky.

```html
<table>
</table>
```

<tr>: Řádek tabulky

Slouží k definování řádku v tabulce. Může obsahovat tagy `th` a `td` jako přímé potomky.

```html
<table>
  <tr></tr>
</table>
```

<th>: Hlavička tabulky

Slouží k definování buňky v tabulce. Jedná se o speciální buňku, která označuje hlavičku sloupce nebo řádku. Může obsahovat jakýkoliv jiný HTML obsah.

Atributy:

scope Určuje, zda hlavička platí pro sloupec, řádek nebo skupinu sloupců či řádků. Možné hodnoty jsou:
- `col`: hlavička platí pro celý sloupec,
- `row`: hlavička platí pro celý řádek,
- A další.

Používáme pro zlepšení přístupnosti tabulek, aby čtečky obrazovky mohly lépe interpretovat strukturu tabulky.

colspan Určuje, kolik sloupců má buňka zabírat. Výchozí hodnota je 1. Používáme, pokud chceme, aby buňka přesahovala více sloupců.

rowspan Určuje, kolik řádků má buňka zabírat. Výchozí hodnota je 1. Používáme, pokud chceme, aby buňka přesahovala více řádků.

```html
<table>
  <tr>
    <th>Hlavička 1</th>
    <th>Hlavička 2</th>
  </tr>
</table>
```

<td>: Buňka tabulky

Slouží k definování buňky v tabulce. Může obsahovat jakýkoliv jiný HTML obsah. Nemá žádný další význam.

Atributy:

colspan Určuje, kolik sloupců má buňka zabírat. Výchozí hodnota je 1. Používáme, pokud chceme, aby buňka přesahovala více sloupců.

rowspan Určuje, kolik řádků má buňka zabírat. Výchozí hodnota je 1. Používáme, pokud chceme, aby buňka přesahovala více řádků.

```html
<table>
  <tr>
    <td>Buňka 1</td>
    <td>Buňka 2</td>
  </tr>
</table>
```

<thead>: Hlavička tabulky

Slouží k seskupení hlaviček tabulky. Může obsahovat pouze tagy `tr` jako přímé potomky. Používáme abychom jasně oddělili hlavičku tabulky od jejího těla, tedy hlavního obsahu.

```html
<table>
  <thead>
    <tr>
      <th>Hlavička 1</th>
      <th>Hlavička 2</th>
    </tr>
  </thead>
</table>
```

<tbody>: Tělo tabulky

Slouží k seskupení hlavního obsahu tabulky. Může obsahovat pouze tagy `tr` jako přímé potomky. Používáme abychom jasně oddělili tělo tabulky od její hlavičky a patičky.

Pokud v tabulce nepoužijeme `tbody`, tak prohlížeč automaticky vytvoří jeden `tbody` a vloží do něj všechny řádky, které nejsou v `thead` nebo `tfoot`. Tabulka bez `tbody` je validní HTML.

```html
<table>
  <tbody>
    <tr>
      <td>Buňka 1</td>
```

```
5       <td>Buňka 2</td>
6     </tr>
7   </tbody>
8 </table>
```

<tfoot>: Patička tabulky

Slouží k seskupení patičky tabulky. Může obsahovat pouze tagy `tr` jako přímé potomky. Používáme abychom jasně oddělili patičku tabulky od jejího těla a hlavičky.

Patička tabulky se často používá pro souhrnné informace, například celkové hodnoty nebo poznámky.

html

```
1 <table>
2   <tfoot>
3     <tr>
4       <td>Celkem</td>
5       <td>100</td>
6     </tr>
7   </tfoot>
8 </table>
```

<caption>: Popisek tabulky

Slouží k přidání popisku tabulky. Může obsahovat jakýkoliv jiný HTML obsah. Popisek se zobrazuje nad tabulkou.

html

```
1 <table>
2   <caption>Popisek tabulky</caption>
3 </table>
```

3.15.4 Sloupce v tabulkách

Sloupce v tabukách můžeme definovat pomocí tagu `colgroup`. Uvnitř uvádíme tagy `col`, které definují jednotlivé sloupce. Hlavní nevýhoda tohoto přístupu je, že nemůžeme definovat obsah sloupců, pouze jejich vlastnosti, například šířku. Tyto vlastnosti jsou avšak velmi omezené, a prohlížeče podporují pouze jednotky vlastností, které známe z CSS.

Kvůli tomuto omezení je často nepoužíváme. Místo toho vybíráme buňky buď pomocí pseudotříd pozice (například `td:first-child` pro první buňku v řádku) nebo pomocí tříd a identifikátorů přímo na buňkách. To má ale problém s údržbou, protože pokud přidáme nový sloupec, musíme upravit všechny řádky, nebo naopak při přidání nového řádku nesmíme zapomenout na třídy.

<colgroup>: Skupina sloupců tabulky

Slouží k seskupení sloupců tabulky. Může obsahovat pouze tagy `col` jako přímé potomky. Používáme pro definování vlastností více sloupců najednou.

```html
<table>
  <colgroup>
    <col style="width: 50%;">
    <col style="width: 25%;">
    <col style="width: 25%;">
  </colgroup>
</table>
```

<col>: Sloupec tabulky

Slouží k definování vlastností jednotlivého sloupce tabulky. Může být použit uvnitř `colgroup` (skupina sloupců).

Styly, které určíme třídě, kterou definujeme na atributu `class`, se aplikují na všechny buňky v tomto sloupci. Povolené vlastnosti jsou omezeny.

Atributy:

span Určuje, kolik sloupců má tento element pokrývat. Výchozí hodnota je 1. Používáme, pokud chceme, aby tento element ovlivňoval více sloupců najednou.

class Určíme třídu, která bude aplikována na všechny buňky v tomto sloupci.

```html
<table>
  <colgroup>
    <col />
    <col style="width: 50%;">
  </colgroup>
</table>
```

3.15.5 Postupná struktura tabulek

Tabulky mají jasně definovanou pozici pro každý z uvedených elementů. Vždy musí následovat následující pořadí:

1. `caption` (nemusí být přítomen),
2. `colgroup` (nemusí být přítomen),
3. `thead` (nemusí být přítomen),
4. `tbody` nebo případně jednotlivé řádky `tr`,
5. `tfoot` (nemusí být přítomen).

3.15.6 Stylování tabulek

Pro stylování můžeme používat již nám známe vlastnosti. Prvky v tabulce vybíráme[33] pomocí tříd a identifikátorů, případně používáme pseudotřídy výběru dle struktury, například `table tr:nth-child(odd)` pro výběr lichých řádků tabulky.

Pro tabulky ale potřebujeme řešit i další nové styly. Pokud na tabulku aplikujeme rámeček, tak se aplikuje jen na okolí celé tabulky. Často tedy aplikujeme rámečky na jednotlivé buňky, řádky a i tabulku samotnou. To ale způsobí, že se rámečky mezi buňkami zdvojí.

border-collapse: Slučování okrajů tabulky

Určuje, zda se mají okraje buněk tabulky slučovat do jednoho okraje nebo zůstávat oddělené. Používáme pro zlepšení vzhledu tabulek a odstranění zdvojených okrajů mezi buňkami.

Definice hodnoty:

`separate` výchozí hodnota, okraje buněk zůstávají oddělené,
`collapse` okraje buněk se slučují do jednoho okraje.

css
```
1 table {
2   border-collapse: collapse;
3 }
```

border-spacing: Vzdálenost mezi buňkami tabulky

Určuje vzdálenost mezi buňkami tabulky, když je `border-collapse` nastaven na `separate`. Používáme pro úpravu vzhledu tabulek a vytvoření prostoru mezi buňkami.

Definice hodnoty:

- Dvě číselné hodnoty s jednotkami, první určuje horizontální vzdálenost a druhá vertikální vzdálenost mezi buňkamim,
- Pokud je uvedena pouze jedna hodnota, platí pro obě osy.

css
```
1 table {
2   border-collapse: separate;
3   border-spacing: 10px 5px;
4 }
```

3.16 Základní stránky

V této kapitole si ukážeme několik základních konceptů, které nám pomohou tvořit jednoduché webové stránky.

[33] Pozor na tbody – pokud ho nevložíte do HTML, vloží se automaticky, a proto například selektor `table > tr` **nemusí** fungovat.

3.16.1 Zarovnání doprostřed

Nejklasičtější problém webové neznalce je jednoduchý – jak zarovnat obsah doprostřed stránky (zařízení). Nejjednodušší způsob je ale ten omezený. Pokud je prvek zobrazen v blokovém režimu (což je výchozí režim pro většinu elementů, například `div`, `p`, `h1` až `h6` a další), můžeme použít vlastnost `margin` s hodnotou `auto` na levé a pravé straně. Daný prvek pak musí mít stanovenou šířku (např. v pixelech nebo procentech). Prvek se poté zarovná doprostřed svého rodičovského elementu.

css

Zarovnání blokového prvku doprostřed

```
1 .container {
2   width: 50%;
3   margin: 0 auto;
4 }
```

Ihned v ukázce vidíme koncept kontejneru. To je obalovací prvek, který nám zjednoduše zarovnání stránky do určité šířky. To se nám velmi bude hodit v responzivním návrhu stránek, viz Kapitola 4.4.

3.16.2 Celostránková stránka

Celostránková stránka je stránka, která zabírá celou výšku a šířku okna prohlížeče. Stránka nepřetéká, tedy není větší než samotné vykreslovací okno prohlížeče. Takové stránky se nejčatěji používají pro jednoduché prezentace, vstupní stránky (landing pages) nebo portfolia. Na stránce tedy nemáme scrollbar.

Na to, abychom zabrali celé místo okna, potřebujeme nastavit výšku a šířku na 100% šířky, reps. výšky okna. Šířka je dle nastavení metatagu `viewport` vždy 100% šířky okna. Výšku ale musíme řešit speciálně. Na to máme již ukázané jednotky `vh` a `vw`, viz Kapitola 3.5.6.2.

css

Základní nastavení celostránkové stránky

```
1 html, body {
2   height: 100vh;
3   width: 100vw;
4 }
```

Nepřesnost jednotky vw

Často se stává, že studenti začnou používat jednotku `100vw` pro jakékoliv nastavení prvku na stránce na to, že zabere celou šířku okna. Problém jednotky `100vw` je, že nezohledňuje šířku scrollbarů – pokud je stránka delší než okno (tedy není celostránková!), tak se zobrazí scrollbar a tím se zmenší viditelná šířka okna, což způsobí přeteční stránky i na šířku – horizontálně.

Pokud si otevřeme stránku v prohlížeči na telefonu, tak zjistíme, že stránka není celostránková. To je způsobeno tím, že prohlížeče na mobilních zařízeních mají různé panely (adresní řádek, navigační tlačítka a další), které se mohou skrývat a zobrazovat. Tyto panely mění velikost viditelné části okna, což způsobí, že stránka přeteče.

Proto existují speciální jednotky `lvh`, `lvw`, `svh`, `svw`, `dvh` a `dvw`, které zohledňují tyto panely. Jednotky s předponou `l` (large) zohledňují panely, které jsou vždy viditelné. Jednotky s předponou `s` (small) zohledňují panely, které jsou vždy skryté. Jednotky s předponou `d` (dynamic) zohledňují aktuální stav panelů (zda jsou viditelné nebo skryté).

Správně by tedy celostránková stránka měla vypadat takto:

css

Základní nastavení celostránkové stránky s dynamickými jednotkami

```css
html, body {
  height: 100dvh;
  width: 100dvw;
}
```

3.16.3 Navigace

Navigace je důležitou součástí každé webové stránky. Navigační menu nám umožňuje uživatelům snadno se pohybovat mezi různými částmi webu. Nejčastěji se navigační menu umisťuje do hlavičky stránky, ale může být i na jiných místech, například v postranním panelu nebo v patičce.

Pro navigační menu se často používají seznamy, protože navigační položky jsou logicky uspořádány do skupin. Níže naleznete krátkou ukázku tvorby jednoduchého navigačního menu.

html

Jednoduché navigační menu[34] v HTML

```html
<div class="navigation">
  <ul>
    <li><a href="#home">Domů</a></li>
    <li><a href="#about">O nás</a></li>
    <li><a href="#contact">Kontakt</a></li>
  </ul>
</div>
```

css

Stylování navigačního menu v CSS

```css
.navigation ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
  display: block;
  text-align: center;
  background-color: #212121;
}
.navigation li {
  display: inline-block;
  margin: 0 15px;
}
.navigation a {
  text-decoration: none;
```

[34] Tag `div` se zde nehodí, bohužel zatím neznáme lepší způsob.

```
15   color: black;
16   font-weight: bold;
17   padding: 10px 15px;
18   border-radius: 5px;
19   background-color: #f0f0f0;
20 }
21 .navigation a:hover {
22   background-color: #e0e0e0;
23 }
```

3.17 Správný kód

Správný kód je důležitý pro zajištění přístupnosti, výkonu a správného fungování aplikace. Pro webové stránky je důležité dodržovat standardy a doporučení stanovená organizací W3C (World Wide Web Consortium). Další aspekty správného kódu jsou velmi subjektivní a liší se podle týmu, projektu nebo osobních preferencí.

Pro tento předmět si ale stanovíme několik základních pravidel, která nám pomohou psát lepší kód:

- Vždy budeme psát validní HTML a CSS kód, který odpovídá standardům W3C,
- Vždy budeme používat nějakou formu resetu nebo normalizace stylů, abychom zajistili konzistentní vzhled napříč různými prohlížeči,
- Kód bude vždy formátován správně a přehledně. Pro odsazení používáme dle své preference mezery nebo tabulátory, ale vždy konzistentně,
- Budeme používat sémantické HTML tagy, které odpovídají obsahu, který chceme zobrazit,
- Obecně budeme dělat stránky přístupné, tedy budeme využívat atributy jako `alt` u obrázků, popisky u formulářových prvků a další,
- Identifikátory (myšleno jakékoliv ID, třídy, datové atributy a podobně) budou pojmenovány smysluplně a konzistentně,
- Identifikátory budou vždy v anglickém jazyce. Jedinou výjimkou jsou případy kotev, které odkazují na části stránky v jiném jazyce (například v češtině),
- Pro identifikátory můžete používat jakoukoliv konvenci pojmenování, ale vždy konzistentně v celém projektu. Možné jsou například:
  - `kebab-case` (například `main-header`),
  - `camelCase` (například `mainHeader`),
  - `snake_case` (například `main_header`).
  - `PascalCase` (například `MainHeader`).
- Třídy budou popisovat sémantické účely tagů, nikoliv jejich vzhled. Například třída `button-primary-red` je špatná, protože popisuje vzhled tlačítka. Místo toho použijeme třídu `submit-button`, která popisuje účel tlačítka,
- O webových stránkách přemýšlíme pro budoucnost (i když se jedná o malý úkol). Proto se snažíme například vyhnout použití inline stylů, identifikátorům a podobně,
- Selektory píšeme co nejvíce volné, aby se daly znovu použít. Tedy nepoužíváme příliš mnoho pevných přímých cest (například `div > ul > li > a`), ale raději použijeme třídy nebo ID,
- Obecně prvkům nastavujeme zejména jednu dimenzi (šířku nebo výšku), aby se mohly přizpůsobit obsahu,

Bonusově můžete přidat i další pravidla dle svého uvážení, já doporučuji například:
- Použití komentářů pro oddělení částí kódu,
- Použití proměnných v CSS (pokud je to možné),
- Použití stejných jednotek v celém projektu (například vždy `rem` místo `px`),
- Použití lepšího způsobu zápis tříd, například BEM nebo např. utility-first přístup.

3.18 Nástroje pro vývojáře

V každém moderním prohlížeči máme k dispozici nástroje pro vývojáře (Developer Tools, zkráceně DevTools). Tyto nástroje nám umožňují ladit a analyzovat naše webové stránky přímo v prohlížeči.

To se nám hodí zejména v pokročilejších částech kurzu, kdy budeme ladit JavaScript a podobně. Pro nás začnou být ale důležité pro debuging pravidel a vlastností v CSS.

Při otevření nástrojů pro vývojáře (obvykle klávesou F12 nebo pravým klikem a volbou „Inspect“ či „Prozkoumat“) můžeme vidět strukturu HTML dokumentu, aplikovaná CSS pravidla, konzoli pro výpis chyb a další užitečné nástroje.

Tyto nástroje nám umožňují:
- Prohlížet a upravovat HTML a CSS v reálném čase,
- Ladit JavaScript kód,
- Analyzovat výkon stránky,
- Testovat responzivní design,
- A mnoho dalšího.

Doporučuji si s těmito nástroji pohrát a seznámit se s jejich funkcemi, protože nám velmi usnadní práci při vývoji webových stránek.

Rozložení stránky

4.1 Tagy pro rozložení stránky

V rámci webových stránek potřebujeme seskupovat prvky do logických celků. Celky poté reprezentují části stránky, které mají společný význam. Prozatím jsme si představili jen pár z nich – zejména seznamy a tabulky.

Pro seskupení bez dalšího významu používáme element `div`. Tento ale neříká nic o tom, jaký význam má obsah uvnitř něj.

Proto si nyní představíme několik dalších elementů, které mají význam pro rozložení stránky.

<header>: Hlavička

Slouží k označení hlavičky stránky nebo hlavičku určité sekce. Obvykle obsahuje nadpis, logo, navigaci nebo jiné úvodní informace. Tyto informaace uvádějí další obsah stránky nebo sekce. Pokud se jedná o hlavičku celé stránky, obvykle jí například čtečky či vyhledávače věnují menší pozornost – protože se jedná o stejný obsah na všech stránkách.

html
```
<header>
  <h1>Nadpis stránky</h1>
  <!-- ... -->
</header>
```

<nav>: Navigace

Slouží k označení navigačního bloku na stránce. Obvykle obsahuje odkazy na další části webu (či na jiné weby). Není omezeno kde se může nacházet – může být v hlavičce, patičce nebo i jinde na stránce.

html
```
<nav>
  <ul>
    <li><a href="/">Domů</a></li>
    <li><a href="/about">O nás</a></li>
    <li><a href="/contact">Kontakt</a></li>
  </ul>
</nav>
```

<main>: Hlavní obsah

Označuje hlavní obsah dokumentu. Měl by být použit pouze jednou na stránce. Obsah uvnitř tohoto elementu by měl být unikátní pro danou stránku a neměl by se opakovat na jiných stránkách (na rozdíl od hlavičky a patičky).

html
```
<main>
  <h2>Nadpis hlavního obsahu</h2>
  <p>Obsah této sekce je specifický pro tuto stránku.</p>
```

```
4 </main>
```

<article>: Článek

Reprezentuje samostatný, nezávislý obsah, který může být distribuován nebo znovu použit samostatně (například v RSS kanálech, e-mailech nebo tiskových verzích). Může obsahovat nadpisy, odstavce, obrázky, videa a další média. Příklady použití zahrnují blogové příspěvky, novinové články, uživatelské komentáře nebo jiné samostatné části obsahu. Často uvnitř používáme i další sémantické elementy jako `header`, `footer` nebo `section` pro lepší strukturování obsahu článku.

```html
1 <article>
2   <h2>Nadpis článku</h2>
3   <p>Obsah článku je nezávislý a může být znovu použit.</p>
4 </article>
```

<section>: Sekce

Používá se k označení tematické sekce v dokumentu. Sekce by měla být použita pro seskupení souvisejícího obsahu dohromady. Samostatně ale sekce nemusí dávat smysl – na rozdíl od článku, který by měl být samostatně pochopitelný. Příklady použití zahrnují kapitoly v knize, sekce na webové stránce nebo tematické bloky v článku.

```html
1 <section>
2   <h2>Nadpis sekce</h2>
3   <p>Obsah této sekce je tematicky související.</p>
4 </section>
```

<aside>: Postranní obsah

Reprezentuje obsah, který je pouze okrajově související s hlavním obsahem stránky. Může obsahovat doplňující informace, jako jsou poznámky, odkazy na související články, reklamy nebo jiné vedlejší informace. Obsah uvnitř tohoto elementu by měl být relevantní pro hlavní obsah, ale neměl by být jeho nedílnou součástí.

```html
1 <aside>
2   <h2>Postranní informace</h2>
3   <p>Tento obsah je okrajově související s hlavním obsahem.</p>
4 </aside>
```

<footer>: Patička

Slouží k označení patičky stránky nebo patičky určité sekce. Obvykle obsahuje informace jako jsou autorská práva, odkazy na zásady ochrany osobních údajů, kontaktní informace nebo navigační odkazy. Tyto informace obvykle uzavírají obsah stránky nebo sekce. Pokud se jedná o patičku celé stránky, obvykle jí například čtečky či vyhledávače věnují menší pozornost – protože se jedná o stejný obsah na všech stránkách.

```html
<footer>
  <p>&copy; 2025 Moje Společnost. Všechna práva vyhrazena.</p>
</footer>
```

Použití těchto sémantických tagů zlepšuje přístupnost a zlepšuje i možné využití obsahu (například pro vyhledávače). Jejich kombinace nám dovolují tvořit v podstatě libovolné rozložení stránky.

4.2 Flexbox

Flexbox je jedna z nejdůležitějších technologií pro tvorbu webových stránek vůbec. Jeho dosah sahá i mimo samotné webové stránky – například i do aplikací, které se z tohoto velmi dobrého návrhu učí.

Dříve webové stránky byly velmi omezené na to, jak se prvky dají rozložit.

Rozložení, jak již víme, je, jak jsou prvky na stránce uspořádány. Jestli jsou vedle sebe, pod sebou, jaké mají mezery, jak se chovají při změně velikosti okna a podobně.

Flexbox nám dovoluje vytvářet velmi flexibilní a přizpůsobivá rozložení. Flexbox je založen na kontejneru, který obsahuje položky.

Kontejneru a položkám můžeme nastavovat různé vlastnosti, které určují, jak se dané prvky uspořádají na stránce.

Flexbox je určen zejména na jednorozměrná rozložení – tedy buď řádky, nebo sloupce. Sice umí i „více dimenzionální“ rozložení, ale to je spíše rozšíření jednoho rozměru zalomením na další řádek nebo sloupec.

4.2.1 Kontejner

Kontejner je element, který obsahuje položky. Kontejneru musíme nastavit speciální typ zobrazení pomocí CSS vlastnosti `display`. Hodnota, kterou použijeme, je `flex`[35]. Tím se z běžného elementu stane flex kontejner.

Nastavení flex kontejneru

```css
.container {
  display: flex;
}
```

[35] Nikoliv `flexbox`. Jediná další hodnota, která se používá, je `inline-flex`, která je ale nad rámec WBA.

V rámci kontejneru poté můžeme všem prvkům, které jsou přímo uvnitř něj, nastavovat rozložení. V základu máme rozložení na hlavní ose a na vedlejší ose. Hlavní osa je ta, ve které se prvky řadí – tedy buď horizontálně (řádky) nebo vertikálně (sloupce). Vedlejší osa je ta, která je na hlavní osu kolmá.

justify-content: Zarovnání na hlavní ose

Zarovná na hlavní ose (podle směru řazení) položky uvnitř kontejneru.

Je možné používat i na zobrazení grid.

Definice hodnoty:

– `flex-start`: Zarovná položky na začátek hlavní osy, – `flex-end`: Zarovná položky na konec hlavní osy, – `center`: Zarovná položky na střed hlavní osy, – `space-between`: Rozprostře položky rovnoměrně mezi začátek a konec hlavní osy, přičemž první položka je na začátku a poslední na konci, v případě jednoho prvku je umístěn na začátek, – `space-around`: Rozprostře položky rovnoměrně podél hlavní osy s rovnoměrným prostorem kolem každé položky, – `space-evenly`: Rozprostře položky rovnoměrně podél hlavní osy s rovnoměrným prostorem mezi všemi položkami a okraji kontejneru.

```css
.container {
  display: flex;
  justify-content: center;
}
```

align-items: Zarovnání na vedlejší ose

Zarovná na vedlejší ose (kolmá na směr řazení) položky uvnitř kontejneru.

Je možné používat i na zobrazení grid.

Definice hodnoty:

– `flex-start`: Zarovná položky na začátek vedlejší osy, – `flex-end`: Zarovná položky na konec vedlejší osy, – `center`: Zarovná položky na střed vedlejší osy, – `stretch`: Roztáhne položky tak, aby vyplnily celý prostor podél vedlejší osy (výchozí hodnota), – `baseline`: Zarovná položky podle jejich základní linie textu.

```css
.container {
  display: flex;
  align-items: center;
}
```

Překvapivě je hodnota `stretch` nejpoužívanější v rámci pokročilých rozložení. Tím, že položky vyplní celý prostor, často odpadá potřeba nastavovat výšku či šířku položek. To je velmi užitečné zejména u dynamického obsahu, kde nevíme předem, jak vysoký či široký bude, o což se budeme právě nadále velmi často starat.

flex-direction: Směr řazení

Určuje směr, ve kterém jsou položky uvnitř kontejneru řazeny.

Definice hodnoty:

– `row`: Položky jsou řazeny horizontálně zleva doprava (výchozí hodnota), – `row-reverse`: Položky jsou řazeny horizontálně zprava doleva, – `column`: Položky jsou řazeny vertikálně shora dolů, – `column-reverse`: Položky jsou řazeny vertikálně zdola nahoru.

```css
.container {
  display: flex;
  flex-direction: column;
}
```

Vlastnost `flex-direction` je velmi důležitá zejména pro responzivní design. Jednoduše nám dovoluje zcela prohodit řazení položek podle velikosti obrazovky. Například máme na stránce tlačítka. Ty obalíme do kontejneru s flexem a prvky se řadí vedle sebe. Na menším zařízení ale chceme, aby se tlačítka řadila pod sebe. Toho docílíme právě změnou `flex-direction` na `column` v rámci responzivního designu.

flex-wrap: Zalomení směru řazení

Určuje, zda se položky uvnitř kontejneru mohou zalamovat na další řádky nebo sloupce, pokud není dostatek místa na hlavní ose.

Definice hodnoty:

– `nowrap`: Položky zůstanou na jedné řádce nebo sloupci a mohou přetékat (výchozí hodnota), – `wrap`: Položky se zalamují na další řádky nebo sloupce, pokud není dostatek místa, – `wrap-reverse`: Položky se zalamují na další řádky nebo sloupce v opačném směru.

```css
.container {
  display: flex;
  flex-wrap: wrap;
}
```

Stejný příklad jako výše u `flex-direction` můžeme použít i zde. Nemusíme vůbec prohazovat řazení, ale můžeme povolit zalamování. To znamená, že se rozložení, kontejner, vždy adaptuje na dostupný prostor. Pokud nebude místo, tak tlačítko zařadí na nový řádek. U řádků (resp. sloupců) je pro nás důležité pochopit, jak se budou chovat vlastnosti `justify-content` a `align-items`. Vlastnost `justify-content` se vždy vztahuje k hlavní ose. Prvky se tedy na každém řádku (sloupci) zarovnají podle této vlastnosti. Vlastnost `align-items` se vždy vztahuje k vedlejší ose. To znamená, že všechny řádky (sloupce) se zarovnají podle této vlastnosti v rámci svého řádku (sloupce).

gap: Mezery mezi položkami

Určuje mezery mezi položkami uvnitř kontejneru. Tato vlastnost je velmi užitečná, protože nám dovoluje nastavit mezery bez potřeby nastavovat okraje jednotlivým položkám[36].

Je možné používat i na zobrazení grid.

Definice hodnoty:

Uvádíme jednu nebo dvě číselné hodnoty s jednotkami.

Toto je složená vlastnost, jednotlivé vlastnosti pro dané hodnoty jsou `row-gap` a `column-gap`.

```css
.container {
  display: flex;
  gap: 10px;
}
```

4.2.2 Položky

Položky jsou prvky, které jsou přímo uvnitř kontejneru. Na položky můžeme nastavovat různé vlastnosti, které určují, jak se dané prvky chovají v rámci kontejneru.

Důležitá vlastnost, které jste si mohli všimnout je znovu to, že prohlížeče prvků dávají výchozí hodnoty následujících vlastností. Je dobré si uvědomit, jestli i tyto vlastnosti nepotřebujete zrovna vy nastavit do svého restového stylu.

order: Pořadí položky

Určuje pořadí, ve kterém jsou položky uvnitř kontejneru zobrazeny. Položky s nižším číslem se zobrazí dříve než položky s vyšším číslem. Výchozí hodnota je `0`, takže všechny položky mají stejné pořadí, pokud není tato vlastnost nastavena. Pokud mají dvě nebo více položek stejné hodnoty, zobrazí se v pořadí, v jakém jsou uvedeny v HTML kódu.

Definice hodnoty:

Uvádíme celé číslo (kladné, záporné nebo nula).

```css
.item {
  order: 1;
}
```

Vlastnost `order` je velmi užitečná pro změnu pořadí položek bez potřeby měnit HTML kód. Znovu se to hodí u responzivního designu, kdy na menších obrazovkách chceme mít jiný obsah na začátku než na větších obrazovkách.

[36] Nezapomeňte na vlastnost `margin`, která spojuje okraje položek a může způsobit nečekané výsledky.

flex-basis: Základní velikost položky

Určuje základní velikost položky před tím, než je prostor rozdělen mezi položky. Tato vlastnost určuje, jak velká by měla být položka na hlavní ose, než se aplikuje jakékoli další přizpůsobení (jako je růst nebo smrštění). Výchozí hodnota je `auto`, což znamená, že velikost položky je určena jejím obsahem. Některé hodnoty ale způsobují jiné chování, například `0` znamená, že položka nemá žádnou základní velikost a veškerý prostor je rozdělen podle hodnot růstu a smrštění.

V podstatě je ekvivalentní k nastavení `width` nebo `height` podle směru řazení.

Definice hodnoty:

Uvádíme hodnotu s jednotkami nebo klíčová slova jako `auto`.

```css
.item {
  flex-basis: 200px;
}
```

flex-shrink: Smrštění položky

Určuje, jak moc se může položka zmenšit, pokud není dostatek místa na hlavní ose. Výchozí hodnota je `1`, což znamená, že položka se může zmenšit podle potřeby. To je viditelné například při zmenšení kontejneru – položky se taktéž zmenší aby se vešly do dostupného prostoru. Ale jen tehdy, pokud není nastavené zalomení.

Pokud je nastavena na `0`, položka se nebude zmenšovat a zachová si svou základní velikost.

Definice hodnoty:

Uvádíme číslo (kladné nebo nula).

Dané číslo určuje poměr, ve kterém se položky zmenšují. Například pokud máme dvě položky, jedna s `flex-shrink: 1` a druhá s `flex-shrink: 2`, druhá položka se zmenší dvakrát více než první, pokud není dostatek místa.

Pokud ale obě mají stejnou jakoukoliv hodnotu, zmenší se obě stejně.

```css
.item {
  flex-shrink: 0;
}
```

flex-grow: Růst položky

Určuje, jak moc se může položka zvětšit, pokud je k dispozici volné místo na hlavní ose. Výchozí hodnota je `0`, což znamená, že položka se nebude zvětšovat nad svou základní velikost. Pokud je nastavena na `1` nebo vyšší, položka se může zvětšit podle potřeby, aby vyplnila dostupný prostor.

Pokud je více položek s nastaveným růstem, dostupný prostor se rozdělí mezi ně podle jejich hodnot růstu.

Definice hodnoty:

Uvádíme číslo (kladné nebo nula).

Dané číslo určuje poměr, ve kterém se položky zvětšují. Například pokud máme dvě položky, jedna s `flex-grow: 1` a druhá s `flex-grow: 2`, druhá položka se zvětší dvakrát více než první, pokud je k dispozici volné místo.

Pokud ale obě mají stejnou jakoukoliv hodnotu, zvětší se obě stejně.

```css
.item {
  flex-grow: 1;
}
```

align-self: Zarovnání položky na vedlejší ose

Umožňuje jednotlivým položkám přepsat zarovnání na vedlejší ose, které je nastaveno na kontejneru pomocí `align-items`. Tímto způsobem můžeme mít různé zarovnání pro různé položky uvnitř stejného kontejneru.

Lze používat i na zobrazení grid.

Definice hodnoty:

– `auto`: Položka dědí zarovnání z kontejneru (výchozí hodnota), – `flex-start`: Zarovná položku na začátek vedlejší osy, – `flex-end`: Zarovná položku na konec vedlejší osy, – `center`: Zarovná položku na střed vedlejší osy, – `stretch`: Roztáhne položku tak, aby vyplnila celý prostor podél vedlejší osy, – `baseline`: Zarovná položku podle její základní linie textu.

```css
.item {
  align-self: center;
}
```

4.2.3 Jak se Flex naučit

Flexbox je velmi rozsáhlý a mocný nástroj. Jeho možnosti jsou opravdu široké a je potřeba si je vyzkoušet v praxi. Existuje mnoho zdrojů a návodů, které vám mohou pomoci se naučit Flexbox. Osobně doporučuji zkusit dvě stránky:

- Flexbox Froggy, hra, kde pomáháte žábě dostat se na leknín pomocí Flexbox vlastností,
- A Complete Guide to Flexbox, velmi obsáhlý a podrobný průvodce všemi vlastnostmi Flexboxu.

V rámci hodin WBA si společně vyřešíme nespočet úkolů, které vám pomohou pochopit, jak Flexbox funguje a jak ho můžete využít ve svých projektech.

4.3 Grid

Grid je další velmi mocný nástroj pro rozložení webových stránek. Je určen zejména pro dvourozměrná rozložení – tedy jak řádky, tak sloupce. Můžeme si to tedy představit jako tabulku. Oproti HTML tabulce můžeme jakékoliv prvky rozmístit do libovolných buněk, řádků a sloupců.

Hlavní rozdíl oproti Flexboxu je tedy v tom, že můžeme pracovat s více rozměry najednou. Můžeme tedy prvnímu sloupci nastavit jasnou velikost – aby se dobře vyhledávano v rozložení.

Znovu definujeme kontejner i položku stejným způsobem jako u Flexboxu. V rámci HTML máme všechny prvky vloženy jako přímé děti kontejneru. Na kontejneru zapneme zobrazení pomocí vlastnosti `display` na hodnotu `grid`.

4.3.1 Kontejner

V rámci kontejnerů můžeme nastavovat různé vlastnosti, které určují, jak se dané prvky uspořádají v rámci mřížky.

Důležitý koncept je, že jednotlivým řádkům či sloupcům můžeme nastavit mimo přesných velikostí taky relativní velikost, která počítá s dostupným prostorem. Tato jednotka se nazývá `fr` (fractional unit) a představuje zlomek dostupného prostoru. Například, pokud máme tři sloupce, každý s velikostí `1fr`, tak se dostupný prostor rozdělí na tři stejné části. Pokud ale první sloupec nastavíme na `2fr` a na ostatní dva na `1fr`, tak první sloupec zabere polovinu dostupného prostoru a zbylé dva čtvrtinu.

grid-template-columns: Definice sloupců

Definuje počet a velikost sloupců v mřížce.

Definice hodnoty:

Uvádíme jednu nebo více hodnot oddělených mezerami. Každá hodnota představuje velikost jednoho sloupce. Můžeme používat číselné hodnoty s jednotkami nebo klíčová slova jako `auto` nebo `min-content`[37].

```css
.container {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

Uvnitř zápisů nějakých vlastností často opakujeme stejné hodnoty. Proto v CSS vznikla funkce `repeat`, která přijímá dva argumenty. Prvním je počet opakování a druhým je hodnota, která se má opakovat. Tímto způsobem můžeme zkrátit zápis. V rámci jedného hodnoty můžeme používat i více hodnot, které se opakují, včetně možnosti jiných hodnot.

Uvnitř sloupců a řádku taktéž můžeme použít funkci `minmax`, která přijímá dva argumenty. Prvním je minimální velikost a druhým maximální velikost. Tímto způsobem můžeme určit,

[37] `min-content` je šířka nejširšího obsahu v daném sloupci.

že sloupec či řádek může mít minimální velikost, ale může se i zvětšit až na určitou maximální velikost podle dostupného prostoru.

grid-template-rows: Definice řádků

Definuje počet a velikost řádků v mřížce.

Definice hodnoty:

Uvádíme jednu nebo více hodnot oddělených mezerami. Každá hodnota představuje velikost jednoho řádku. Můžeme používat číselné hodnoty s jednotkami nebo klíčová slova jako `auto` nebo `min-content`.

```css
.container {
  display: grid;
  grid-template-rows: repeat(3, 1fr) 100px;
}
```

V rámci sloupců a řádků můžeme používat i vlastnost `gap` (resp. `row-gap` a `column-gap`), kterou známe z Flexboxu.

Pokud nastavíme gridu uvnitř HTML 10 položek, ale v CSS máme definované tři sloupce a tři řádky, tak se položka, která je na více (10.) řádku, zobrazí. Zobrazí se ale již automatickou velikosti – tedy podle obsahu.

To, jakou má řádek či sloupec velikost po přidání dalších položek nad rámec definovaných používáme vlastnost `grid-auto-rows` a `grid-auto-columns`.

grid-auto-rows: Automatické řádky

Určuje velikost řádků, které jsou přidány automaticky, pokud není dostatek definovaných řádků.

Ekvivaltně můžeme definovat i `grid-auto-columns` pro sloupce.

Definice hodnoty:

Uvádíme číselné hodnoty s jednotkami nebo klíčová slova jako `auto` nebo `min-content`.

```css
.container {
  display: grid;
  grid-auto-rows: 100px;
}
```

Uvnitř gridu pak můžeme používat vlastnosti pro rozmístění sloupců, řádků a položek. Na to máme vlastnosti `justify-content` a `align-content`. V rámci flexboxu známe pouze `justify-content`, který na hlavní ose určuje zarovnání položek. Avšak i tam lze `align-content` použít, pokud máme více řádků (při zalomení) – tam určuje zarovnání řádků na vedlejší ose v rámci kontejneru. V rámci gridu vlastnosti `*-content` určují zarovnání celé mřížky – sloupců a řádků – v rámci kontejneru. To se hodí právě tehdy, když jednotlivé sloupce a řádky se ne-expandují na celou velikost kontejneru. Tím tedy můžeme dát umístění na hlavní ose (horizontálně, `justify-content`) a na vedlejší ose (vertikálně, `align-content`). Možné hodnoty jsou: `start`,

`end`, `center`, `stretch`, `space-between`, `space-around`, `space-evenly`, viz popis vlastnosti, Kapitola 4.2.1.

Další vlastnosti, které určují zarovnání, jsou `justify-items` a `align-items`. Ty zarovnávají jednotlivé položky uvnitř mřížky. Určením velikosti sloupce a řádku na nějakou hodnotu může nastat, že reálný prvek je menší než velikost buňky. Pomocí těchto vlastností můžeme určit, jak se má položka uvnitř buňky zarovnat. Pokud chceme, aby prvek vyplnil celou buňku, použijeme hodnotu `stretch` nebo nastavíme velikost položky na `100%` v obou osách. Možné hodnoty jsou: `start`, `end`, `center`, `stretch`, viz popis vlastnosti, Kapitola 4.2.1.

4.3.2 Položky

Položkám uvnitř gridu můžeme nastavovat vlastnosti, které umožní přesné umístění v rámci mřížky.

grid-column: Umístění sloupců

Určuje, ve kterých sloupcích se má položka zobrazit. Můžeme určit počáteční a koncový sloupec, ve kterém se má položka zobrazit.

Obdobně můžeme definovat i `grid-row` pro řádky.

Definice hodnoty:

Uvádíme dvě čísla oddělená lomítkem `/`. První číslo je počáteční sloupec a druhé číslo je koncový sloupec. Sloupce se počítají od `1`. Pokud chceme prvek přes jeden (první) sloupec, ale chceme aby zabral dva sloupce, tak použijeme `1 / 3`.

Můžeme také použít klíčová slova jako `span` pro určení, kolik sloupců má položka zabrat. Tím se neurčuje specifický sloupec, ale použije se ten, který bude indikován z pozice položky v HTML.

Toto je složená vlastnost, jednotlivé vlastnosti pro dané hodnoty jsou `grid-column-start` a `grid-column-end`.

```css
.container {
  grid-column: 1 / 3;
}
```

)

4.3.3 Další možnosti gridu

Grid má připravené další vlastnosti, které nám umožní ještě více přizpůsobit rozložení. Mezi tyto vlastnosti patří například `grid-template-areas`, která nám dovoluje pojmenovat oblasti v rámci mřížky a poté je přiřadit položkám. To je velmi užitečné pro složitější rozložení, kde chceme mít jasné pojmenování oblastí. Dále je tu vlastnost `grid-auto-flow`, která určuje, jakým způsobem se mají automaticky přidávat nové položky do mřížky[38]. Můžeme určit, zda se mají přidávat řádek po řádku nebo sloupec po sloupci.

[38]Na toto jste si mohli sami všimnout u vlastnosti `grid-auto-columns`, protože bez této vlastnosti se položky přidávají řádek po řádku.

Tyto vlastnosti avšak nejsou důležité pro většinu běžných použití gridu. Proto je v rámci WBA nebudeme detailně probírat.

4.3.4 Jak se Grid naučit

Stejně jako u Flexboxu je potřeba si Grid vyzkoušet v praxi. Doporučuji vám zkusit následující stránky:

- CSS Grid Garden, hra, kde pomáháte rostlině růst pomocí Grid vlastností,
- A Complete Guide to Grid, velmi obsáhlý a podrobný průvodce všemi vlastnostmi Gridu.

4.4 Responzivní design

Responzivitu si lze představit jako schopnost webové stránky přizpůsobit se různým velikostem obrazovek a zařízením. Často této věci říkáme obdobnými pojmy, jako je „adaptivní design“ nebo „fluidní design“. Zjednodušeně to znamená, že naše webová stránka bude fungovat a vypadat dobře na různých typech zařízení, ať už je to desktopový počítač, tablet nebo mobilní telefon.

Podporované zařízení velmi záleží na tom, pro koho webovou stránku vytváříme, respektive kdo ji bude používat. Pokud děláme např. interní komerční aplikaci pro firmu, která běží pouze na kokrétním hardwarovém zařízení (např. tablet s Window a konkrétním rozlišením), tak nám stačí se zaměřit pouze na toto zařízení. Naopak pokud vytváříme veřejný web, tak musíme počítat s tím, že uživatelé mohou používat různá zařízení s různými velikostmi obrazovek.

Prozatím všechny stránky, co jsme tvořili, byly statické a neměnily se podle velikosti obrazovky. Při přibližování či čistém otevřením jinde jasně vidíme, že stránka není přizpůsobená. Podle nastavení v tagu `<meta>` s hodnotou `viewport` se stránka může buď zmenšit, nebo zvětšit, aby se vešla do obrazovky. To ale nedělá z designu vhodnou formu pro dané zařízení. Pokud jsme například na telefonu, vždycky očekáváme, že obsah bude zejména pod sebou, aby se dobře četl. Taktéž očekáváme, že bude dostatečně velký, aby s ním šlo interagovat a aby se dal dobře přečíst bez nutnosti přibližování. Naše webové stránky často začnou přetékat na druhé dimenzi, a to je většinou něco, co určitě nechceme.

Moderní weby jsou vždy vázané na jednu dimenzi – nejčastěji na šířku obrazovky. Druhá dimenze je potom určena pro skrolování.

> 📺 **Proč jenom jedna dimenze?**
>
> Obecně se předpokládá pohyb ve více dimenzích jako špatný uživatelský zážitek. Uživatelé jsou zvyklí rolovat stránku vertikálně (nahoru a dolů) pro čtení obsahu.
>
> Při zapnutí obou dimenzí je ovládání krkolomné a uživatelé mohou snadno ztratit přehled o tom, kde se na stránce nacházejí. Navíc, mnoho zařízení, zejména mobilních, má omezený prostor na obrazovce a přidání horizontálního posouvání by mohlo zhoršit použitelnost a čitelnost obsahu.

Klíčové pro vytvoření dobré responzivní stránky je to, mít již velmi kvalitní základní rozložení. Často se na webu setkáváme s tím, že se webová stránka vypracuje na jednom typu zařízení (často desktop, nebo naopak mobil) a pak se snažíme přidat responzivitu dodatečně na ostatní zařízení. Často začínáme na zařízení, které je primární pro učel. Například vyplňování daní skoro nikdo nebude dělat na mobilu, takže začneme na desktopu a pak přidáme podporu pro

tablet. Naopak u sociálních sítí je primární zařízení mobil, takže začneme na mobilu a pak přidáme desktop.

Pokud nebudeme mít kvalitní základní rozložení, tak nám bude dělat problémy i samotná responzivita. Respektive bude hrozně moc práce přizpůsobit rozložení na různá zařízení. Níže si zopakujeme, co očekáváme, že už naše webová stránka bude umět:

- Používat relativní jednotky pro velikosti, mezery a podobně.
- Mít flexibilní rozložení pomocí Flexboxu nebo Gridu.
- Mít nastavenou pouze jednu dimenzi (nejčastěji šířku) pro rozložení, ostatní se dopočítává automaticky.
- Mít dobře nastavený viewport v tagu `<meta>`.

Pokud je vše v pořádku, očekáváme, že responzivita vzhledem k celkovému času vývoje bude pouze zlomek[39].

Mezi relativní (často tzv. responzivní jednotky) se řadí procenta (`%`), jednotky `vw` a `vh` (viewport width/height), jednotky `em` a `rem` (relativní k velikosti písma) a jednotka `fr` (fractional unit) u gridu. Pokud přemnoho nastavujeme jiné jednotky, tak tuto jednotku budeme muset přepočítávat pro různá zařízení. To je moc hodnot!

4.4.1 Podpora zařízení

Jak jste se dozvěděli dříve, to, jaká zařízení chceme podporovat, záleží na tom, pro koho webovou stránku vytváříme. Od teď budeme počítat, že vytváříme zejména prezentační webové stránky, které mají fungovat na běžných zařízeních.

Běžné zařízení jsou taková zařízení, které se přespříliš neodlišují od toho, co očekáváte od běžného uživatele. Podporovat totiž to, že webové stránky můžeme otevřít na chytrých hodinkách, na chytrých ledničkách a dalších, je možná cool, ale v praxi to není potřeba. Stačí nám tedy počítat s tím, že uživatelé budou používat běžné počítače, tablety a mobilní telefony.

Praxe nám říká, že je vhodné podporovat všechna zařízení, které mají šířku obrazovky alespoň 320 pixelů. To zahrnuje většinu mobilních telefonů na trhu. Druhá strana problému je, že řešíme šířku obrazovky do přibližně 1920 pixelů na šířku.

Šířka a výška obrazovky taktéž není dobrý ukazatel toho, co vlastně velikostí myslíme. Anglicky se často definuje pojem „viewport“[40], což je ta část obrazovky, ve které se zobrazuje webová stránka. Viewport může být menší než skutečná velikost obrazovky, protože prohlížeče mají různé panely, lišty a další prvky uživatelského rozhraní, které zabírají místo[41].

Většina responzivity se právě řeší dle šířky viewportu. Další problém je, že sice zařízení může být široké, ale může být velmi nízké. To znamená, že musíme počítat i s výškou viewportu. Pokud se ale na výšku zařízení neváženě (a tedy, jak víme, nenastavujeme vlastnost `height`, protože je automatická), často vůbec nemusíme řešit responzivitu na výšku. Problémy tvoří

[39] Osobně to odhaduji na maximálně 15 % celkového času vývoje, pokud máme kvalitní základní rozložení. Naopak pokud musíme řešit např. responzivní chování i u interaktivních částí, tak se toto může opravdu protáhnout.

[40] Nikdy se mi nepodařilo hezky přeložit tento pojem do češtiny. Doslovně je to „zobrazovací okno“, ale to mi přijde divné. Často se používá „viditelná oblast“, ale to taky není úplně ono. Takže budu používat anglický termín.

[41] Vzpomeňte si na to, jak jsem vám u jednotky `vh` zmiňoval, že na mobilních telefonech tato jednotka nefunguje úplně podle očekávání kvůli tomu, že prohlížeč může mít skryté panely a proto používáme jiné jednotky jako `dvh` (dynamic viewport height).

třeba přilepené hlavičky, které mohou zabrat příliš mnoho místa na menších obrazovkách. Znovu ale počítáme s minimální výškou viewportu 320px[42].

Pokud je viewport mimo tento rozsah, stránku nesmíme rozbít – co uděláme, je, že stránka bude mít stejné nastavení, jako na nejbližší podporované zařízení. Z pohledu zmenšení je to očividné. U zvětšení je potřeba řešit více věcí. Představte si, že máte širokoúhlý monitor. Pokud na takovém monitoru zobrazíte webovou stránku, která je optimalizovaná pro 1920px na šířku (a je poměrně dobře vytvořená), tak se stránka umístí na střed a zbylé místo po stranách zůstane prázdné.

Tomuto systému říkáme kontejnery (containers). To znamená, že máme hlavní kontejner, který má maximální šířku (např. 1920px) a je vycentrovaný na stránce. Tento kontejner měníme v závislosti na šířce viewportu. Vždycky má nastavený tento kontejnér zarovnání na střed (flexbox nebo pomocí vlastnosti `margin` s hodnotou `auto`). Tímto způsobem docílíme toho, že na velkých obrazovkách bude stránka vypadat dobře a nebude se roztahovat do nekonečna.

> **Roztažení do nekonečna**
>
> Rozatžení do maxima znamená, že i další obsah na stránce bude roztahován. To se stane nejspíše i s textem a co se může stát, je to, že místo několik čitelných řádek textu se stane jeden dlouhý řádek, který bude těžko čitelný.
>
> Vědecké studie (a zkuste si to sami) ukazují, že ideální délka řádku pro čtení na obrazovce je mezi 50 až 75 znaky na řádek. Po tomto limitu je velmi jednoduché se ztratit v místě textu a ztratit přehled o tom, kde vlastně jsme.

4.4.2 Jiné způsoby responzivity

Mimo celého systému přizpůsobení se konkrétnímu zařízení můžeme používat ještě další techniky, které nám pomohou přizpůsobit obsah na zařízení.

Některé stránky vznikli například ještě v době, kdy nebyla responzivita tak běžná. O historii rozložení jsme se bavili dříve, viz Kapitola 3.15.1. Tyto stránky často používali pevné šířky pro různé části stránky.

Po zavedení přenosných zařízení a mobilních telefonů se začaly používat techniky, by znamenalo že se musí celá tato stránka předělat od začátku. To není moc dobrý manažerský nápad.

Proto vznikl nápad toho, že se vytvoří dvě různé verze stránky. Jedna se bude zobrazovat na velkých obrazovkách (desktop) a druhá na malých obrazovkách (mobil). Tento přístup se nazývá „adaptive design“ (adaptivní design).

Extrémní způsob tohoto přístupu je to, že se vytvoří zcela samostatné webové stránky pro mobil a pro desktop. To má například známý český e-shop Alza.cz. Při přístupu na desktop se načte desktopová verze. Při přístupu na stejnou stránku z telefonu se detekuje, že se jedná o mobilní zařízení a načte se mobilní verze stránky, resp. se uživatel přesměruje na jinou doménu (konkrétně m.alza.cz).

Tento přístup se stále používá, pokud je např. desktopová verze příliš složitá a obsahuje takové funkcionality, které by třeba na mobilu nebyly použitelné.

[42] Vemte si telefon, šířku má 320px, otočte ho do módu „landscape“ (na šířku) a výška bude stále minimálně 320px.

4.4.3 Body zlomu

Body zlomu (angl. „breakpoints“) jsou specifické šířky viewportu, při kterých se mění rozložení stránky. Tato myšlenka vychází z adaptivního designu, kde máme předdefinované rozložení pro různé typy (šířky viewportu) zařízení (např. mobil, tablet, desktop). Tyto rozložení garantují, že se pro danou šířku viewportu bude používat konkrétní stylování a rozložení. Poté se podle šířky viewportu vybere takové rozložení, které se nejlépe hodí. Pokud máme např. nějakou šířku mezi „tabletem“ a „desktopem“, tak se použije rozložení pro „tablet“, protože je určitě šířka viewportu větší než pro „tablet“ ale zároveň je určitě menší než pro „desktop“.

Tyto body zlomu se často nastavují právě zmíněným kontejnerem, který má maximální šířku pro dané zařízení. Například pro mobilní zařízení může mít maximální šířku 320px a to se bude zobrazovat pro všechny zařízení do šířky viewportu 700px, kde se začne používat kontejner pro tablet s maximální šířkou 768px.

Tento způsob nám velmi zjednoduší práci s responzivitou. Místo nekonečného množství různých šířek viewportu můžeme počítat pouze s několika málo rozloženími. Nejčastěji se používají tři až šest různých bodů zlomu, které pokrývají nejběžnější zařízení. Nám bude stačit používat tři, často s těmito hodnotami:

- Mobil, do 700px, šířka kontejneru 320px,
- Tablet, do 1200px, šířka kontejneru 700px,
- Desktop, nad 1200px, šířka kontejneru 1200px.

Kontejner bude stále veprostřed stránky a bude mít nastavenou maximální šířku podle daného bodu zlomu.

4.4.4 Media queries

Nyní můžeme přestoupit k tomu, jak se reálně responzivita dělá. Po nastavení zmíněného `<meta>` tagu s viewportem a vytvoření kvalitního základu pro rozložení můžeme přistoupit k samotnému přizpůsobení obsahu.

V CSS proto máme speciální at-pravidla nazvané `@media`. Pokud již umíte programovat[43], tak si můžete představit, že `@media` je něco jako podmínka.

Pravidlo `@media` nám dovoluje zapnout (aplikovat) určitý blok CSS kódu (několik pravidel) pouze tehdy, pokud je splněna určitá podmínka. Obsahuje dvě části: podmínku a blok kódu. Podmínka určuje, kdy se má blok kódu aplikovat.

Do podmínky vkládáme vlastně klíčová slova a případně vlastnosti s konktrétnimi hodnotami. Mezi nejčastější patří:

- `max-width`: Určuje maximální šířku viewportu, při které se má blok kódu aplikovat. Tato (a některé další) vlastnost je nová a je možné ji využívat k více věcím, viz níže,
- `min-width`: Určuje minimální šířku viewportu, při které se má blok kódu aplikovat,
- `max-height`: Určuje maximální výšku viewportu, při které se má blok kódu aplikovat,
- `min-height`: Určuje minimální výšku viewportu, při které se má blok kódu aplikovat,
- `orientation`: Určuje orientaci zařízení (např. `portrait` pro na výšku, `landscape` pro na šířku),
- `resolution`: Určuje rozlišení obrazovky zařízení (např. `300dpi` pro tiskárny),
- `print`: Specifikuje, že se blok kódu má aplikovat pouze při tisku stránky,
- `screen`: Specifikuje, že se blok kódu má aplikovat pouze při zobrazení na obrazovce,

[43] Což tak trošku doufám, že umíte.

- a další, viz později.

css
Media query s minimální šířkou viewportu
```
@media (min-width: 300px) {
  .container {
    width: 100%;
  }
}
```

Pro jednoduší zápis šířek a výšek můžeme používat i zkrácené zápisy pomocí oprátor <=, >=, < a >. Například místo `max-width: 700px` můžeme napsat `width <= 700px`.

css
Media query s minimální šířkou viewportu (zkrácený zápis)
```
@media (width <= 700px) {
  .container {
    width: 100%;
  }
}
```

Pro zápis podmínek můžeme používat různé logické operátory jako `and`, `,` a `not`. Mezi zvláštní operátor patří `only`, který zajišťuje, že se blok kódu aplikuje pouze tehdy, pokud je zařízení schopné danou podmínku vyhodnotit. Toto se používá zejména pro starší prohlížeče, které nemusí některé vlastnosti podporovat. Logické nebo se zapisuje pomocí čárky `,`, v moderních prohlížečích můžeme používat i klíčové slovo `or`.

css
Media query s logickým operátorem AND
```
@media (min-width: 700px) and (max-width: 1200px) {
  .container {
    width: 80%;
  }
}
```

css
Media query s logickým operátorem OR
```
@media (min-width: 1200px, orientation: landscape) {
  .container {
    width: 70%;
  }
}
```

min-width: Minimální šířka

Určuje minimální šířku prvku, která se nesmí překročit.

Pokud je obsah prvku větší než tato hodnota, prvek se zvětší, aby obsah pojmul – ale záleží na nastavení vlastnosti `width` a dalších (např. `aspect-ratio`).

Používá se taktéž v media queries pro určení minimální šířky viewportu, při které se má blok kódu aplikovat.

Stejná vlastnost existuje i pro výšku – `min-height`.

Definice hodnoty:

Uvádíme číselné hodnoty s jednotkami (např. `px`, `%`, `vw`, `em`, apod.).

```css
.container {
  min-width: 300px;
}
```

max-width: Maximální šířka

Určuje maximální šířku prvku, která se nesmí překročit.

Pokud je obsah prvku větší než tato hodnota, prvek se zmenší a obsah se přizpůsobí (např. zalomením textu nebo přizpůsobením obrázku) – ale záleží na nastavení vlastnosti `width` a dalších (např. `aspect-ratio`).

Používá se taktéž v media queries pro určení maximální šířky viewportu, při které se má blok kódu aplikovat.

Stejná vlastnost existuje i pro výšku – `max-height`.

Definice hodnoty:

Uvádíme číselné hodnoty s jednotkami (např. `px`, `%`, `vw`, `em`, apod.).

```css
.container {
  max-width: 800px;
}
```

4.4.5 Dobré praktiky

Při používání `@media` je dobré dodržovat několik zásad, které nám pomohou udržet kód přehledný a efektivní. Často se mezi studenty stává, že při psaní responzivity začnou vytvářet selektorky prvků „od nuly“. Tedy nenavazují na to, co je již v CSS napsáno. Jak jsme si říkali dříve, viz Kapitola 3.5.7, pravidla defivané později, mají přednost před pravidly dříve definovanými. Navíce konkrétní selektory mají různou prioritu mezi sebou[44]. To znamená, že pokud budeme mít v obyčejném CSS (v našem „základu“ webové stránky, tj. desktopové nebo mobilní verzi) pravidlo se selektorem `.container .item`, a v rámci media query napíšeme pravidlo s selektorem `.item`, tak toto pravidlo bude mít nižší prioritu než to první. To může vést k tomu, že naše změny v rámci media query nebudou fungovat, protože budou přepsány pravidly z „základního“ CSS.

To byste samozřejmě jednoduše zjistili pomoci např. DevTools v prohlížeči, ale je to zbytečná komplikace. Proto se vždy budeme snažit kopírovat selektory z „základního“ CSS do media query. A taktéž budeme psát media query na konci CSS souboru, aby měly vyšší prioritu –

[44] Pro fajnšmekry doporučuji materiály pro MVOP Webového inženýrství ve třetím ročníku.

nebo alespoň hned za pravidla pro daný selektor. Pravidel `@media` může být v CSS souboru více.

Co spousta lidí dělá, je, že si responzivitu představuje jako způsob, jak se obsah „vléva“ do různých velikostí obrazovek. To je dobré zejména pro základní pochopení responzivity. My se totiž snažíme, aby se prvky dle své velikosti přizpůsobily velikosti obrazovky. Často musíme něco obětovat, například to, že prvek bude vyšší.

Tohle osobně beru jako nedostatečné pochopení responzivity. Responzivita totiž není o tom, že se prvky „roztahují“ nebo „smršťují“. Responzivita je o tom, že se prvky přizpůsobují různým velikostem obrazovek tak, aby byly použitelné a čitelné. To znamená, že někdy prvek může být menší, jindy větší, jindy může být úplně jiný. Například na mobilu může být navigační menu skryté za ikonou „hamburger menu“, zatímco na desktopu je viditelné celé. To je také responzivita.

Aby byla responzivita co nejjednodušší, používáme flexbox nebo grid pro rozložení. Často u flexboxu ve responzivitě děláme jednu z těchto dvou věcí:

- Měníme směr osy z řádku na sloupec (pomocí `flex-direction`),
- Zapneme zalamování položek (pomocí `flex-wrap`).

Jen tyhle dvě akce nám dovolí, že se obsah přizpůsobí různým velikostem obrazovek velmi jednoduše. Potom designujeme změnu responzivity jen pro samotné prvky ve flexu (kde často měníme, jen velikost písma, mezery a podobně).

Na ukázku toho, jak by měla být responzivita jednoduchá přikládám ukázku studijního portálu (který zajisté znáte). Portál (v době psaní tohoto textu) používá na všechny (i interní stránky) 3110 pravidel. Na responzivitu pomocí queries existuje 116 pravidel. Na poměr je to tedy přibližně 4 % všech pravidel.

4.4.6 Container queries

Container queries jsou nová funkcionalita v CSS, která nám umožňuje aplikovat styly na základě velikosti rodičovského kontejneru, nikoli pouze na základě velikosti viewportu. To je užitečné zejména v případech, kdy máme komponenty, které mohou být umístěny v různých kontextech a velikost jejich rodičovského kontejneru může ovlivnit jejich vzhled a chování. Obecně použití container queries by mělo být upřednostňováno před media queries, protože to vede k více modulárnímu a znovupoužitelnému kódu. Navíce kód bude přehlednější, protože styly budou přímo navázány na komponenty, nikoli na celou stránku.

Fungují v podstatě stejně jako media queries, ale místo toho, abychom kontrolovali velikost viewportu, kontrolujeme velikost rodičovského kontejneru.

Container query s minimální šířkou kontejneru

```css
@container (min-width: 400px) {
  .item {
    font-size: 1.5rem;
  }
}
```

Mimo jiné dovolují container queries používat dvě **klíčové funkcionality**:

- Definování na **jaký kontejner** se mají container query pravidla kontrolovat,
- Použití proměnných a funkcí pro validaci platnosti query.

Na konkrétní prvky v rámci struktury komponenty (ale i stránky) můžeme přidat vlastnost `container-name`, která určuje jméno kontejneru a na ten později můžeme navázat pravidla v rámci `@container`. Taktéž můžeme definovat vlastnost `container-type`, která určuje, jaký typ kontejneru to je, což dovoluje pravidlům se ptát na konkrétní vlastnosti kontejneru (např. velikost, stav scrollování, apod.).

container-name: Název kontejneru

Určuje jméno kontejneru pro použití v container queries.

Tímto způsobem můžeme definovat více kontejnerů na stránce a aplikovat styly na základě velikosti konkrétního kontejneru.

Definice hodnoty:

Uvádíme jméno kontejneru jako identifikátor (bez mezer a speciálních znaků).

Můžeme uvést více jmen oddělených mezerou, čímž definujeme více kontejnerů na jednom prvku.

Nesmí se jednat o klíčová slova, jako je např. `or`, `and`, `not`, apod.

css

```
.container {
  container-name: main-container;
}
```

container-type: Typ kontejneru

Určuje typ kontejneru pro použití v container queries.

Tento typ určuje, jaké vlastnosti kontejneru mohou být použity v pravidlech container queries.

Definice hodnoty:

Možné hodnoty jsou:

- `size`: Umožňuje kontrolovat velikost kontejneru (šířka a výška),
- `inline-size`: Umožňuje kontrolovat pouze šířku kontejneru,
- `block-size`: Umožňuje kontrolovat pouze výšku kontejneru,
- `normal`: Výchozí hodnota, která neumožňuje žádné kontroly.

css

```
.container {
  container-type: inline-size;
}
```

css

Container query s názvem kontejneru a minimální šířkou

```
@container main-container (min-width: 500px) {
  .item {
    background-color: lightblue;
  }
}
```

Stejně, jako u media queries můžeme používat logické operátory.

Container query s logickými operátory

```css
@container card (min-width: 400px), style(--responsive: true), scroll-state(stuck: top)
  .item {
    font-size: 1.2rem;
  }
}
```

V ukázce výše můžeme vidět, že kromě klasické podmínky s minimální šířkou kontejneru můžeme používat i další podmínky. Prvním z nich je funkce `style()`, která dovoluje kontrolovat hodnoty CSS vlastností (včetně vlastních proměnných). Druhou funkcí je `scroll-state()`, která dovoluje kontrolovat stav scrollování kontejneru (např. zda je scrollovací oblast přilepená na vrchu nebo dole).

Do funkce `scroll-state()` můžeme vložit:

- `scrollable: none | top | right | bottom | left | x | y | block-start | block-end | inline-start | inline-end | block | inline`, zda je scrollovací oblast scrollovatelná na nějaké straně,
- `snapped: none | x | y | block | inline | both`, zda je scrollovací oblast zarovnaná[45],
- `stuck: none | top | right | bottom | left | block-start | block-end | inline-start | inline-end`, zda je scrollovací oblast přilepená na nějaké straně (pro pozicování typu `sticky`, viz Kapitola 5.5).

Obecně nám ale stačí práce se základními hodnotami:

- `min-width` a `max-width` pro šířku kontejneru,
- `min-height` a `max-height` pro výšku kontejneru,
- `aspect-ratio` pro poměr stran kontejneru,
- `orientation` pro orientaci kontejneru (na výšku nebo na šířku).

Veškeré hodnoty se počítají z velikosti daného kontejneru, resp. z jeho box modelu, viz Kapitola 3.7.1.

Pro kontejnery existují taktéž nové jednotky, které slouží právě pro práci s kontejnery:

- `cqw` (container query width): 1 % šířky kontejneru,
- `cqh` (container query height): 1 % výšky kontejneru,
- A další, jako `cqi`, `cqb`, `cqmin`, `cqmax`, viz MDN Web Docs.

4.4.7 Další media funkcionality

Konsorcium webových stránek (W3C) definuje následující tzv. media features, které dovolují CSS (resp. celému webu) se ptát zařízení na konkrétní vlastnosti. Již zmíněné je `orientation`.

Do tzv. „typů médií“ (media types) patří „screen“ (obrazovka), „print“ (tisk) a „all“ (všechny typy).

Představíme si další media features, které se často na webových stránkách používají:

[45] CSS dovoluje definovat body, na kterých se scrollování musí „zastavit“. Toto se používá zejména u karuselů a podobných prvků.

- `prefers-color-scheme`: Určuje, zda uživatel preferuje tmavý nebo světlý režim. Hodnoty jsou `light`, `dark` nebo `no-preference`,
- `prefers-reduced-motion`: Určuje, zda uživatel preferuje snížené množství animací a pohybů na stránce. Hodnoty jsou `reduce` nebo `no-preference`,
- `hover`: Určuje, zda zařízení podporuje hover efekty (např. myš). Hodnoty jsou `hover` nebo `none`,
- `display-mode`: Určuje, jakým způsobem je webová stránka zobrazena (např. v režimu celé obrazovky nebo v režimu prohlížeče). Hodnoty jsou `fullscreen`, `standalone`, `minimal-ui` nebo `browser`,
- `resolution`: Určuje rozlišení obrazovky zařízení v DPI (dots per inch) nebo DPCM (dots per centimeter),
- `aspect-ratio`: Určuje poměr stran viewportu nebo kontejneru (např. `16/9` pro širokoúhlý formát),
- A další, viz MDN Web Docs - @media.

css

Media query pro tmavý režim

```css
@media (prefers-color-scheme: dark) {
  body {
    background-color: black;
    color: white;
  }
}
```

Design webových stránek

5.1 Písma

Dříve, viz Kapitola 3.5.8, jsme si ukázali, jak se na webové stránky, resp. na konkrétní text aplikuje písmo pomocí vlastnosti `font-family`. Nyní si ukážeme, jak se na webové stránky přidávají nová písma.

Písma jsou jeden z nejdůležitějších aspektů webového designu. Hlavním důvodem je, že písma mají velký vliv na čitelnost textu a celkový vzhled webové stránky. Je to taktéž velmi dobrý způsob, jak se odlišit od konkurence a vytvořit si vlastní značku.

Taky jsme si řekli, že na to, abychom na webu mohli použít nějaké písmo, musí být toto písmo nainstalováno na zařízení uživatele. To byla lež. Můžeme totiž importovat jakékoliv písmo pomocí CSS.

Pomocí pravidla `@font-face` můžeme definovat vlastní písmo, které bude použito na webové stránce.

Definice vlastního písma

```css
@font-face {
  font-family: "MyCustomFont";
  src: url("fonts/MyCustomFont.woff2") format("woff2"),
       url("fonts/MyCustomFont.woff") format("woff");
  font-weight: 400;
  font-style: normal;
}
```

Do definice určujeme název písma, zdroj písma a další vlastnosti. Jako zdroj můžeme uvést různé formáty písma, například `.woff2`, `.woff`, `.ttf` nebo `.otf`. To, jaké se použije, záleží na prohlížeči a jeho podpoře.

Pokud písmo poté chceme použít, používáme dané jméno ve vlastnosti `font-family`. Pokud chceme použít více tlouštěk písma, můžeme definovat více pravidel `@font-face` s různými hodnotami `font-weight`.

Obecně pro fonty platí, že se jedná o poměrně velké množství dat, hlavně kvůli různým variantám písma (např. tučné, kurzíva). Proto nechceme mít příliš mnoho různých fontů, které by zatěžovaly načítání stránky. Obecné pravidlo je mít maximálně 2-3 různá písma na stránku. A pro každé písmo mít maximálně 3-4 různé varianty (např. normální, tučné, kurzíva, tučná kurzíva). Pokud používáme omezené znaky, je taktéž dobré optimalizovat fonty na tyto znaky, aby se zmenšila velikost souboru. Některé fonty jsou totiž plné znaků, které třeba vůbec nepoužíváme. CSS dovoluje omezit a rozdělit fonty podle znaků, ale to je pokročilejší téma[46].

Fonty se ale často ukládají pouze jako vektorové obrázky, což znamená, že jsou škálovatelné a vypadají dobře na různých zařízeních a rozlišeních obrazovky. Výhodou fontů je ale to, že se nejedná o čisté obrázky, ale jsou velmi optimalizované právě na vykreslování textu.

Fonty nemusíme používat pouze na zobrazení obyčejného textu, ale používáme je i na jiné prvky, jako třeba ikony, viz Kapitola 5.2.

[46] Pro pokročilé, podívejte se na `unicode-range` ve specifikaci CSS Fonts.

5.2 Ikony

Ikony jsou velmi důležitou součástí webového designu. Dovolují totiž jednoduše, často s použitím méně místa než text, vyjádřit určitou funkci nebo informaci. Ikony se na webu používají různými způsoby. Nejběžnější způsob je použití ikon jako obrázků – buď jako rastrové obrázky (PNG, JPG) nebo jako vektorové obrázky (SVG).

Je taktéž možné použít ikony jako fonty, což je velmi populární způsob díky knihovnám jako Font Awesome nebo Material Icons. Tyto knihovny nám zadarmo dávají k dispozici řadu ikon, které můžeme použít na našich webových stránkách tím, že importujeme jejich CSS soubory, které importují font, viz Kapitola 5.1. Problém s webovými fonty a ikonami je takový, že sémanticky ikony začnou být považovány za text – každý ikona je totiž právě znak v písmu. To může vést k nejasnostem a například v rámci čteček obrazovky nemusí být ikony správně interpretovány. Pro nejlepší sémantiku se doporučuje používat ikony jako SVG obrázky a k němu umisťovat popisky pro čtečky obrazovky, např. pomocí atributu `aria-label`.

Při importu poslední verze Font Awesome (verze 6) můžeme použít ikony takto:

Použití ikony z Font Awesome

```html
<i class="fa-solid fa-camera"></i>
```

Kde `fa-solid` určuje styl ikony (plný) a `fa-camera` určuje konkrétní ikonu (fotoaparát). Ikony můžeme samozřejmě stylovat pomocí CSS, například měnit jejich velikost, barvu nebo přidávat efekty.

5.3 Designové vlastnosti

Představíme si další vlastnosti, které se zejména používají pro zlepšení designu určitých prvků bez nutnosti použití obrázků.

box-shadow: Stín prvku

Přidá stín k prvku. Můžeme nastavit, zda stín bude uvnitř či vně prvku, jeho rozmazání, posun a barvu.

Definice hodnoty:

Povinně určujeme:

- horizontální posun (např. `2px`),
- vertikální posun (např. `2px`),
- rozmazání (např. `5px`), což určuje, jak moc bude stín rozmazaný,
- barvu (např. `rgba(0, 0, 0, 0.3)`).

Nepovinně můžeme přidat:

- rozšíření stínu (např. `3px`), což způsobí, že stín bude větší základní barvou okolo prvku,
- zda je stín uvnitř prvku (`inset`).

Navíce můžeme hodnoty kombinovat tím, že vytvoříme pro jeden prvek více stínů oddělených čárkou.

```css
.box {
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3), inset 1px 1px 3px rgba(0, 0, 0, 0.2);
}
```

text-shadow: Stín textu

Přidá stín k textu. Můžeme nastavit posun, rozmazání a barvu stínu.

Definice hodnoty:

Povinně určujeme:

- horizontální posun (např. `1px`),
- vertikální posun (např. `1px`),
- rozmazání (např. `2px`), což určuje, jak moc bude stín rozmazaný,
- barvu (např. `rgba(0, 0, 0, 0.5)`).

Navíce můžeme hodnoty kombinovat tím, že vytvoříme pro jeden text více stínů oddělených čárkou.

```css
.text {
  text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.5);
}
```

filter: Filtry pro prvky

Umožňuje aplikovat různé vizuální efekty na prvky, jako je rozmazání, změna jasu, kontrastu, odstínu a další.

Definice hodnoty:

Kombinujeme různé funkce. Záleží na pořadí jednotlivých volání funkcí. Mají různé parametry.

- `blur(px)`: Rozmazání prvku o zadaný počet pixelů,
- `brightness(%)`: Úprava jasu prvku. Hodnota 100% je výchozí, vyšší hodnoty zvyšují jas, nižší snižují,
- `contrast(%)`: Úprava kontrastu prvku. Hodnota 100% je výchozí,
- `grayscale(%)`: Převod prvku na odstíny šedi. Hodnota 100% znamená úplně šedý prvek,
- `sepia(%)`: Aplikace sépiového efektu na prvek. Hodnota 100% znamená plný sépiový efekt,
- `hue-rotate(deg)`: Otočení odstínu barev o zadaný počet stupňů v rámci barevného spektra,
- `invert(%)`: Inverze barev prvku. Hodnota 100% znamená úplnou inverzi,
- `saturate(%)`: Úprava sytosti barev prvku. Hodnota 100% je výchozí,
- `opacity(%)`: Nastavení průhlednosti prvku. Hodnota 100% je plně neprůhledný, 0% je plně průhledný.

```css
.image {
  filter: grayscale(100%) blur(5px) brightness(150%);
}
```

transform: Transformace prvků

Umožňuje aplikovat různé transformace na prvky, jako je posun, otočení, škálování a zkosení. Změny nemění původní tok dokumentu, což znamená, že ostatní prvky kolem nebudou ovlivněny.

Definice hodnoty:

Kombinujeme různé funkce. Záleží na pořadí jednotlivých volání funkcí.

- `translate(x, y)`: Posun prvku o zadané hodnoty v osách X a Y,
- `rotate(deg)`: Otočení prvku o zadaný počet stupňů,
- `scale(x, y)`: Škálování prvku v osách X a Y. Hodnota 1 znamená původní velikost,
- `skew(x-deg, y-deg)`: Zkosení prvku o zadané úhly v osách X a Y,
- `matrix(a, b, c, d, e, f)`: Aplikace obecné transformační matice na prvek.

```css
.box:hover {
  transform: rotate(45deg) scale(1.5);
}
```

Důležitá myšlenka za transformacemi je to, že stejně jako například vlastnost `filter` se jedná pouze o grafickou úpravu prvku. Pokud prvek nahneme, otočíme, nebo zvětšíme, jeho původní velikost a pozice zůstávají stejné. Tyto velikosti tedy neovlivňují ostatní prvky na stránce. Při vykreslování se vezme původní velikost prvku (automatická, nebo z vlastností `width` a `height`), včetně dalších hodnot dle box modelu (padding, border, margin). Z této velikosti se vypočítá rozložení stránky. Až poté se při vykreslení řeší, jak bude prvek vypadat, včetně aplikace transformací a filtrů.

Pro transformace, resp. rotace je nutné si definovat nové jednotky:

- `deg`: Stupně (degrees). Úhel v rozsahu 0 až 360 stupňů[47],
- `rad`: Radiány (radians). Alternativní jednotka pro úhly, kde 2πrad je 360°,
- `turn`: Otáčky (turns). Jedna otáčka je 360°, tedy 1turn = 360° = 2πrad.

Jako bod pro transformace se používá výchoze střed prvku. To můžeme změnit pomocí vlastnosti `transform-origin`, ta nám dovolí nastavit jiný bod, kolem kterého se transformace provádí.

transform-origin: Bod transformace prvku

Určuje bod, kolem kterého se aplikují transformace prvku.

Výchozí hodnota je `50% 50%`, což znamená střed prvku.

[47] Často zadáváme i větší stupně, a to zejména, když chceme například animovat „více otáček“. Pokud ale animace neděláme, tak je vhodné držet se v rozsahu 0-360°.

Definice hodnoty:

Hodnoty mohou být číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2) nebo klíčová slova (`top`, `bottom`, `left`, `right`, `center`). Můžeme zadat dvě hodnoty – první pro horizontální osu a druhou pro vertikální osu.

```css
.element {
  transform-origin: top left;
}
```

backdrop-filter: Filtry na pozadí prvků

Umožňuje aplikovat vizuální efekty na oblast za prvkem, jako je rozmazání nebo změna jasu. Samotný prvek zůstává nezměněn, ale oblast za ním je upravena. Upravení počítá i s pozadím samotného prvku (např. poloprůhledného).

Definice hodnoty:

Hodnoty jsou ekvivalentní jako u vlastnosti `filter`, viz vlastnost filter. Je možné kombinovat různé funkce, které se aplikují na pozadí prvku.

```css
.modal {
  backdrop-filter: blur(10px) brightness(80%);
}
```

5.4 Animace a přechody

Pokud někoho požádáte o to, aby vám vysvětlil, co jsou animace, pravděpodobně Vám řekne, že animace je iluze. Ve skutečnosti je animace sekvence obrázků, které se rychle mění za sebou, čímž vytvářejí dojem pohybu. Pokud něco zobrazujeme mimo reálný život, například na obrazovce počítače, tak musíme tento pohyb nějakým způsobem simulovat, protože nemáme[48] displej, který by uměl zobrazit reálný svět.

Simulace docílíme tím, že rychle měníme obrázky na obrazovce. Důležite je právě, jak často se tyto obrázky mění. Lidské oko totiž vnímá změny přibližně od 24 snímků za sekundu (fps) výše jako plynulý pohyb. Pro nejlepší zážitek se snažíme dosáhnout alespoň 60 fps, což je standardní obnovovací frekvence většiny monitorů. Oko je ale i schopné vnímat i vyšší frekvence, například 120 fps nebo 240 fps, což je běžné u moderních herních monitorů a VR zařízení. U větších frekvencí ale už není skoro rozdíl vnímán, protože lidské oko má své limity.

Definovat ale animace na všech, např. 120 snímcích za sekundu, by bylo velmi náročné. Proto se animace často definují pomocí klíčových snímků (angl. keyframes), které budeme v rámci těchto skript uvažovat. Klíčový snímek je určitý stav animace v konkrétním čase. Můžeme si to představit, jako kdybychom v daný moment „zmrazili“ animaci a podívali se, jak vypadá. Mezi klíčovými snímky pak dochází k interpolaci, která zajišťuje, že se vypočítá pohyb mezi těmito stavy.

[48] Alespoň v roce 2025.

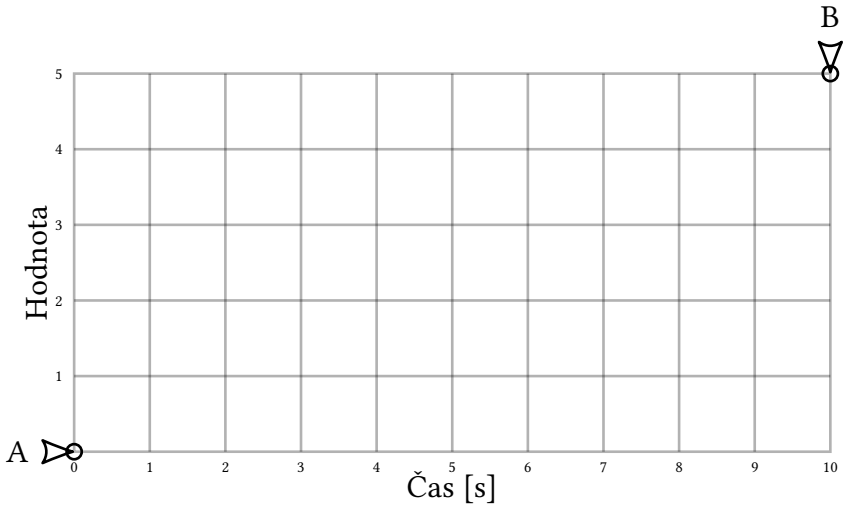
Prohlížeč poté vypočítané hodnoty mezi klíčovými snímky vykresluje v dané frekvenci, čímž vytváří dojem plynulého pohybu. Použitím snímků taktéž zajistíme to, že můžeme jednoduše měnit délku animace, aniž bychom museli měnit samotné klíčové snímky.

V rámci webů máme desítky způsobů, jak animace definovat a používat. Pro tyto skripta budeme v rámci CSS uvažovat dva hlavní způsoby:

- Přechody (transitions): Jednoduché animace mezi dvěma stavy prvku, které se spouští při určité události, například při najetí myší nebo kliknutí,
- Animace (animations): Složitější animace, které mohou mít více klíčových snímků, opakování a další vlastnosti.

5.4.1 Interpolace

Interpolace je proces výpočtu mezilehlých hodnot mezi dvěma známými hodnotami. Máme definované dva body, například počáteční a koncový stav nějaké vlastnosti, a chceme zjistit, jaké hodnoty by měly být mezi těmito dvěma body v různých časech.



Obrázek 3: Graf znazorňující nutné dopočítání hodnot mezi dvěma hodnotami.

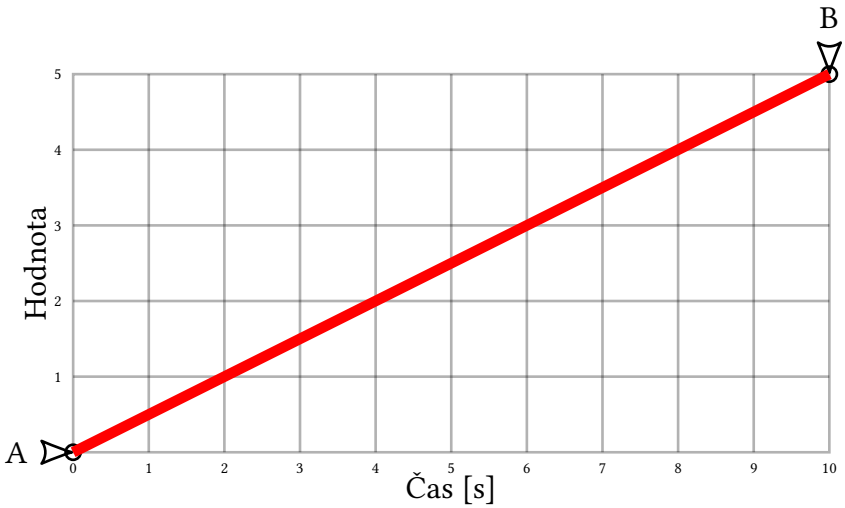
Obrázek 3 obsahuje základ, ze kterého si vysvětlíme interpolaci. Obsahuje dva body A a B, které mají určité hodnoty v čase (0 a 10 sekund). Jako hodnotu vypočítáme například výšku bodu na ose Y (z 0 do 5).

Mezi těmito dvěma body potřebujeme vypočítat „přechod“ hodnoty z bodu A do bodu B během daného času. Nejjednodušší způsob je lineární interpolace, která jednoduše rovnoměrně mění hodnotu mezi dvěma body. To znamená, že v každém čase vypočítáme hodnotu podle vzorce[49]:

$$\text{hodnota}(t) = \text{hodnota}_A + (\text{hodnota}_B - \text{hodnota}_A) * \frac{t - \text{čas}_A}{\text{čas}_B - \text{čas}_A}$$

U jiných typů interpolace není dopočítávání takto jednoduché.

[49] Zkuste si uvědomit, proč tento vzorec dává smysl. Je nutné vědět, co dělá první člen a proč je „zjištění aktuálního času“ tak důležité.



Obrázek 4: Lineární interpolace mezi dvěma body A a B.

Obrázek 4 ukazuje výsledek této lineární interpolace mezi body A a B. Můžeme ověřit, zda náš vzorec funguje. Vybereme např. čas $t = 4s$:

$$\text{hodnota}(4) = 0 + (5 - 0) * \frac{4 - 0}{10 - 0} = 2$$

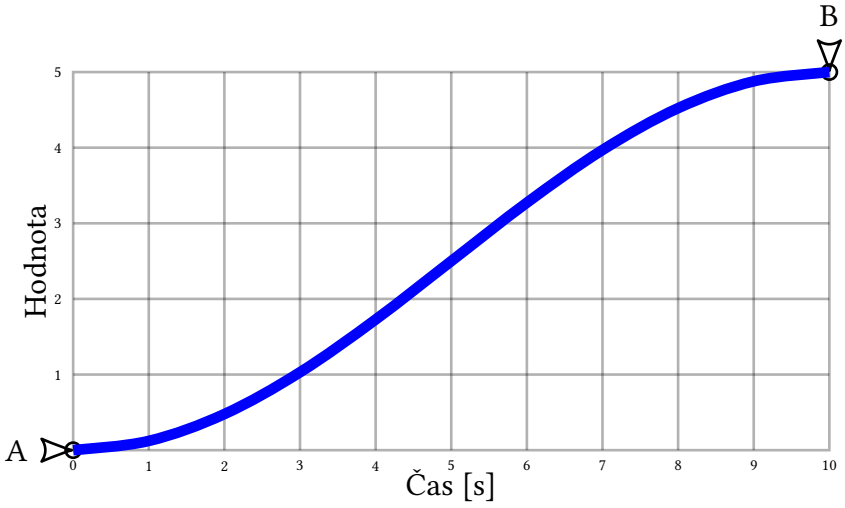
Což pomocí grafu můžeme ověřit. Je důležité si všimnout, že naše vybrané hodnoty jsou „pěkné“. Kdyby např. bod A nezačínal v čase 0 a na hodnotě 0, začala by hrát roli i tato posunutí.

V případě že bychom chtěli vypočítat hodnotu začínající v čase $t = 3s$ a začínal by na hodnotě 4 a končil stejně, tak by vzorec po dosazení vypadal takto:

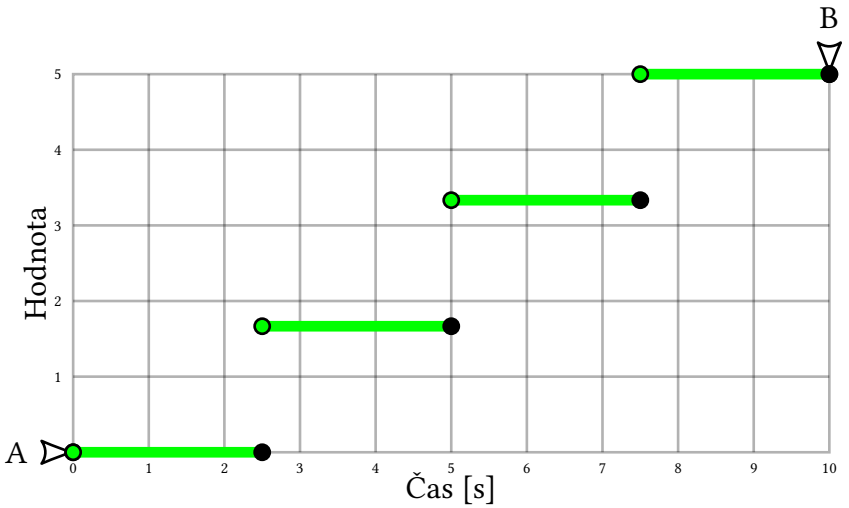
$$\text{hodnota}(4) = 4 + (5 - 4) * \frac{4 - 3}{10 - 3} = 4 + 1 * \frac{1}{7} \approxeq 4 + 0.142857 \approxeq 4.142857$$

Linární interpolace ale není jediný způsob, jak mezi dvěma body počítat hodnoty. Existují i jiné metody. Ukážeme si dvě další běžné metody:

- Vyhlazené funkce (sinusoida, cosinusoida),
- Schodové funkce.



Obrázek 5: Sinusoida jako vyhlazená interpolace mezi dvěma body A a B.



Obrázek 6: Schodová funkce jako interpolace mezi dvěma body A a B.

Konkrétní typy interpolačních funkcí jsou popsány níže, viz Kapitola 5.4.3.

Mimo interpolační typy je důležité zmínit, že nějaké hodnoty prostě pomocí interpolace vypočítat nejdou. Například barvy se dají interpolovat, protože mezi dvěma barvami můžeme vypočítat mezilehlé barvy. To se počítá dle *barevného prostoru* (viz skripta MVOP Webového inženýrství pro 3. ročník), který určuje, jak se barvy reprezentují. V rámci RGB si ale můžeme představit, že můžeme interpolovat každou ze tří složek (červená, zelená, modrá) zvlášť.

Naopak například u vlastnosti `display` nemá smysl počítat mezilehlé hodnoty. Oni totiž neexistují. Buď je hodnota `block`, nebo `inline`, nebo `none`, není totiž zádná mezní hodnota. Nedává totiž ani smysl animovat mezi flexboxem a jiným stylem. I když si můžeme představit, jak by to asi vypadalo, ale to v praxi není možné generalizovat. Těmto vlastnostním říkáme, že mají diskrétní hodnoty (diskrétní, nebo také skokové hodnoty).

Nejčastěji tyto vlastnosti neanimujeme, nebo používáme moderní vlastnost `transition-behavior`, která nám dovolí určit, jak se má přechod chovat při animaci těchto vlastností, ať už je ten skokový krok správně časově umístěn.

Pokud chceme pozici prvků animovat více plynuje a s jasným postupem, je lepší použít jiné způsoby animace, a to zejména pomocí JS, viz Kapitola 6.3.5.

5.4.2 Přechody

Přechody nám dovolují definovat jednoduché animace mezi dvěma stavy prvku. Prvek se na stránce mění mezi stavy:

- Pomocí uživatelské interakce, například najetím myší, kliknutím nebo zaostřením,
- Programově, například pomocí JavaScriptu tím, že změníme vlastnost či např. třídu, a tím se zaktivuje nové pravidlo.

Pokud vykreslovací jádro zaregistruje, že aktuální stav neodpovídá stav neodpovídá tomu, který má přejít, začne se podle vlastnosti `transition` animovat daná vlastnost. Přechod se tedy vždy dívá na cílové nastavení `transition` a to používá pro aktuální animaci.

transition-property: Vlastnosti pro přechod

Určuje, které vlastnosti prvku budou animovány při přechodu mezi stavy.

Definice hodnoty:

Můžeme zadat konkrétní vlastnosti (např. `background-color`, `transform`) nebo použít hodnotu `all`, která znamená, že budou animovány všechny změněné vlastnosti. Při zadání více vlastností je oddělujeme čárkou.

```css
.element {
  transition-property: background-color, transform;
}
```

transition-duration: Doba trvání přechodu

Určuje, jak dlouho bude přechod trvat.

Definice hodnoty:

Hodnotu zadáváme v sekundách (`s`) nebo milisekundách (`ms`). Při zadání více hodnot je oddělujeme čárkou. Pokud je méně hodnot než vlastností v `transition-property`, poslední hodnota se použije pro zbývající vlastnosti.

```css
.element {
  transition-duration: 0.5s;
}
```

transition-timing-function: Funkce časování přechodu

Určuje rychlostní křivku přechodu, tedy jak se bude rychlost animace měnit v čase. Viz Kapitola 5.4.1 a Kapitola 5.4.3.

Definice hodnoty:

Existují předdefinované hodnoty (např. `linear`, `ease`) nebo lze použít funkce jako `cubic-bezier(...)` nebo `steps(...)`. Viz Kapitola 5.4.3.

```css
.element {
  transition-timing-function: ease-in-out;
}
```

transition-delay: Zpoždění přechodu

Určuje, jak dlouho bude trvat, než přechod začne po změně stavu.

Definice hodnoty:

Hodnotu zadáváme v sekundách (`s`) nebo milisekundách (`ms`). Při zadání více hodnot je oddělujeme čárkou. Pokud je méně hodnot než vlastností v `transition-property`, poslední hodnota se použije pro zbývající vlastnosti.

```css
.element {
  transition-delay: 0.2s;
}
```

Častěji se používá zkrácený zápis vlastnosti `transition`, která kombinuje všechny čtyři předchozí vlastnosti do jednoho zápisu. V tomto zápisu můžeme taktéž přehledně zapisovat více přechodu najednou pro různé vlastnosti.

Zkrácený zápis přechodů

```css
.element {
  transition: background-color 0.5s ease-in-out 0s, transform 1s linear 0.2s;
}
```

transition-behavior: Chování přechodu

Určuje, jak se bude přechod chovat při animaci vlastností, které animovat nelze pomocí interpolace. Viz Kapitola 5.4.1.

Definice hodnoty:

Možné hodnoty jsou:
- `normal`: Výchozí chování. Pokud vlastnost nelze interpolovat, přechod se provede ihned na novou hodnotu,
- `allow-discrete`: Povolit diskrétní přechody. Pokud vlastnost nelze interpolovat, přechod se provede až během animace (obyčejně v „prostředku“).

```css
.element {
  transition-behavior: allow-discrete
}
```

Pro definici základního přechodu můžeme použít at-pravidlo `@starting-style`, které nám dovolí definovat výchozí styl prvku, ze kterého se bude přecházet při načtení stránky.

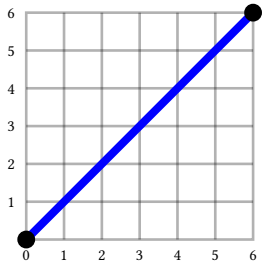
5.4.3 Funkce časování

V rámci animací a přechodů často potřebujeme určit, jakým způsobem se bude hodnota měnit v čase. V CSS máme předdefinované následující funkce časování (easing functions):
- `linear`: Lineární změna hodnoty v čase. Rychlost je konstantní,
- `ease`: Výchozí funkce časování. Začíná pomalu, zrychluje a pak opět zpomaluje na konci,
- `ease-in`: Začíná pomalu a pak zrychluje,
- `ease-out`: Začíná rychle a pak zpomaluje,
- `ease-in-out`: Kombinace `ease-in` a `ease-out`. Začíná a končí pomalu, uprostřed je rychlejší.

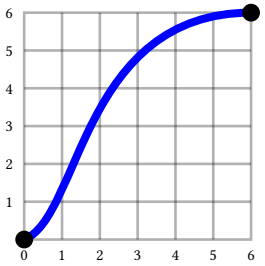
Kromě těchto předdefinovaných funkcí můžeme použít i vlastní funkce:
- `cubic-bezier(x1, y1, x2, y2)`: Definice vlastní kubické Bézierovy křivky pomocí čtyř kontrolních bodů. Hodnoty `x1`, `y1`, `x2`, `y2` jsou v rozsahu od 0 do 1,

- `steps(n, direction)`: Definice schodové funkce s `n` kroky. Volitelný parametr `direction` může být `start` (skok na začátku kroku) nebo `end` (skok na konci kroku).

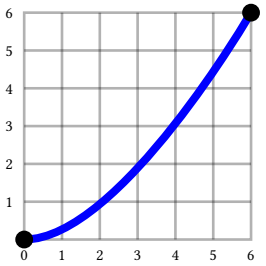
Níže naleznete vizuální znázornění jednotlivých funkcí časování.



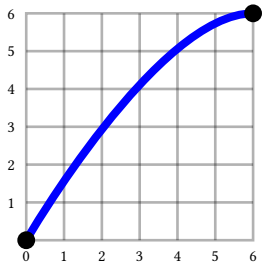
Obrázek 7: linear



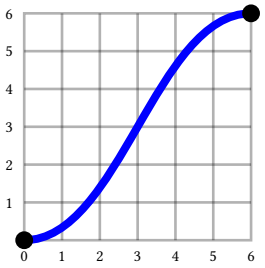
Obrázek 8: ease



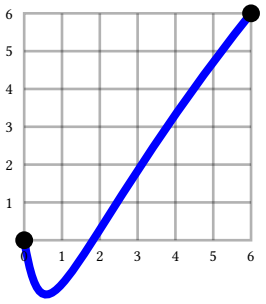
Obrázek 9: ease-in



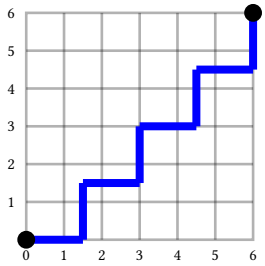
Obrázek 10: ease-out



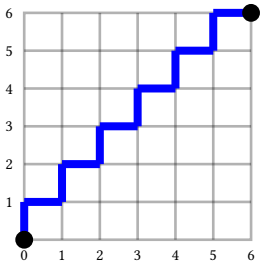
Obrázek 11: ease-in-out



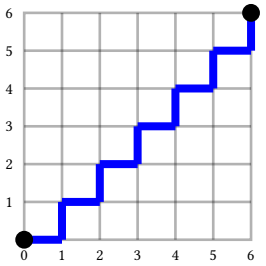
Obrázek 12: cubic-bezier(0.1, -0.6, 0.2, 0)



Obrázek 13: steps(4)



Obrázek 14: steps(6, start)



Obrázek 15: steps(6, end)

V různých jiných systémech existují další předdefinované (resp. známé) funkce časování, například `easeInSine`, `easeOutCubic` apod. Některé však z nich nelze vyjádřit přímo v CSS kvůli tomu, že CSS podporuje pouze `cubic-bezier` a `steps`. Na seznam neznámějších funkcí časování se můžete podívat například na easings.net.

5.4.4 Čas

Již dříve jsme implicitně zmínili časové jednotky, které se používají pro definici délky trvání animací a přechodů. V CSS máme dvě hlavní časové jednotky:

- `s`: Sekundy. Základní jednotka času. 1s = 1000ms,
- `ms`: Milisekundy. Menší jednotka času. 1000ms = 1s.

5.4.5 Animace

Dalším typem animací jsou klíčové snímky (keyframes). Ty nám místo přechodu mezi dvěma stavy dovolují definovat více mezistavů, mezi kterými se má animace plynule měnit. Tyto stavy musíme ale přesně definovat pomocí at-pravidla `@keyframes`, kde určujeme jednotlivé procentuální kroky animace a jejich vlastnosti. Oproti stavovým animacím lze tyto nastavovat nehledě na uživatelskou interakci, tedy mohou běžet samy od sebe. Můžeme definovat počet opakování, zpoždění, směr animace a další vlastnosti.

Nevýhodou je, že musíme přesně definovat všechny mezistavy, což může být pracné pro složitější animace. Další je to, že konkrétní snímky vždy zabírají stejný čas, resp. mezi snímky není možné definovat různé časování. Jediné časování znovu nastavuje časová funkce pro *celou animaci*.

animation-name: Název animace

Definuje, jaká animace (definovaná pomocí `@keyframes`) se má použít pro prvek.

Definice hodnoty:

Hodnotou je název animace, kterou jsme definovali pomocí `@keyframes`.

Při více animacích jejich názvy oddělujeme čárkou.

```css
.element {
  animation-name: slide-in;
}
```

animation-duration: Délka animace

Určuje, jak dlouho bude trvat jedna iterace animace.

Definice hodnoty:

Hodnotu zadáváme v sekundách (`s`) nebo milisekundách (`ms`).

Při více animacích jejich délky oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-duration: 2s;
}
```

animation-timing-function: Časovací funkce animace

Určuje rychlostní křivku animace, tedy jak se bude rychlost animace měnit v čase. Viz Kapitola 5.4.1 a Kapitola 5.4.3.

Definice hodnoty:

Existují předdefinované hodnoty (např. `linear`, `ease`) nebo lze použít funkce jako `cubic-bezier(...)` nebo `steps(...)`. Viz Kapitola 5.4.3.

Při více animacích jejich časovací funkce oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-timing-function: ease-in-out;
}
```

animation-delay: Zpoždění animace

Určuje, jak dlouho bude trvat, než animace začne po načtení stránky nebo po spuštění animace.

Definice hodnoty:

Hodnotu zadáváme v sekundách (`s`) nebo milisekundách (`ms`).

Při více animacích jejich zpoždění oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-delay: 1s;
}
```

animation-iteration-count: Počet opakování animace

Určuje, kolikrát se má animace opakovat.

Definice hodnoty:

Hodnotou může být číslo (např. `3`) nebo klíčové slovo `infinite` pro nekonečné opakování.

Při více animacích jejich počty opakování oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-iteration-count: 3;
}
```

animation-direction: Směr animace

Určuje, jakým směrem se má animace přehrávat.

Definice hodnoty:

Možné hodnoty jsou:
- `normal`: Výchozí hodnota. Animace se přehrává od začátku do konce,
- `reverse`: Animace se přehrává od konce k začátku,
- `alternate`: Animace se střídavě přehrává od začátku do konce a pak od konce k začátku,
- `alternate-reverse`: Animace se střídavě přehrává od konce k začátku a pak od začátku do konce.

Při více animacích jejich směry oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-direction: alternate;
}
```

animation-fill-mode: Zachování animace

Určuje, jak se má prvek chovat před začátkem a po skončení animace. Respektive, jaké styly mají být aplikován na prvek mimo dobu trvání animace.

Definice hodnoty:

Možné hodnoty jsou:
- `none`: Výchozí hodnota. Po skončení animace se prvek vrátí do původního stavu před animací,
- `forwards`: Po skončení animace zůstane prvek ve stavu, který měl na konci animace,
- `backwards`: Před začátkem animace se prvek nastaví do stavu, který měl na začátku animace,
- `both`: Kombinace `forwards` a `backwards`. Prvek se nastaví do stavu na začátku animace před jejím spuštěním a zůstane ve stavu na konci animace po jejím skončení.

Při více animacích jejich režimy zachování oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-fill-mode: forwards;
}
```

animation-play-state: Stav přehrávání animace

Určuje, zda je animace spuštěna nebo pozastavena.

Definice hodnoty:

Možné hodnoty jsou:
- `running`: Výchozí hodnota. Animace je spuštěna a přehrává se,
- `paused`: Animace je pozastavena a nepřehrává se.

> Při více animacích jejich stavy přehrávání oddělujeme čárkou. Pokud je méně hodnot než názvů v `animation-name`, poslední hodnota se použije pro zbývající animace.

```css
.element {
  animation-play-state: paused;
}
```

Animace často nastavujeme pomocí složené vlastnosti `animation`, která kombinuje všechny předchozí vlastnosti do jednoho zápisu.

Zkrácený zápis animace

```css
.element {
  animation: slide-in 2s ease-in-out 1s infinite alternate forwards;
}
```

Animace definujeme pomocí at-pravidla `@keyframes`. Do pravidla píšeme jednotlivé kroky, které se vždy váží na nějaké procento průběhu animace (0% až 100%). V každém kroku pak definujeme vlastnosti, které se mají v daném kroku nastavit.

Definice animace pomocí @keyframes

```css
@keyframes slide-in {
  0% {
    transform: translateX(-100%);
    opacity: 0;
  }
  100% {
    transform: translateX(0);
    opacity: 1;
  }
}
```

Jednotlivých kroků může být libovolný počet. Můžeme je definovat i na základě klíčových slov `from` (ekvivalent 0%) a `to` (ekvivalent 100%). Je možné definovat i necelé procenta, například 25.5%.

Animace s více kroky a vlastnostmi

```css
@keyframes color-change {
  from {
    background-color: red;
  }
  50% {
    background-color: yellow;
  }
  60% {
    background-color: blue;
    transform: rotate(45deg);
```

```
11      font-size: 1.5em;
12    }
13    to {
14      background-color: green;
15    }
16  }
```

5.5 Pozicování

Prvky na stránce rozmisťujeme pomocí různých metod. V základním „flow“ stránky se jedná o rozložení (layout) podle box modelu. Nejčastěji ho kombinujeme s jinými způsoby, jako jsou flexbox nebo grid, viz Kapitola 4.2 a Kapitola 4.3.

Někdy ale potřebujeme prvky umístit na konkrétní místo na stránce, bez ohledu (či s ohledem) na ostatní prvky. Toto rozmístění ale potřebujeme řešit v jiném kontextu, než je běžné rozložení stránky.

Tomu říkáme pozicování (positioning). Pozicování ve většině případů způsobí, že prvek „vypadne“ z běžného toku dokumentu a ostatní prvky se podle něj neřídí. Máme několik typů pozicování, které si nyní představíme.

position: Pozicování prvku

Určuje způsob, jakým je prvek pozicován na stránce.

Můžeme určit, že například prvek nebude vázan na své místo v běžném toku dokumentu, ale bude umístěn relativně k jinému prvku nebo například k oknu prohlížeče.

Definice hodnoty:

Možnými hodnotami jsou:

- `static`: Výchozí hodnota. Prvek je umístěn podle běžného toku dokumentu,
- `relative`: Prvek je umístěn relativně ke své původní pozici. Můžeme použít vlastnosti `top`, `right`, `bottom` a `left` k posunu prvku od jeho původní pozice,
- `absolute`: Prvek je umístěn relativně k nejbližšímu předkovi, který má pozicování jiné než `static`. Pokud takový předek neexistuje, je prvek umístěn relativně k počátku dokumentu (html element), nebo dle kotvy, viz Kapitola 5.5.7,
- `fixed`: Prvek je umístěn relativně k oknu prohlížeče. Zůstává na stejném místě i při posouvání stránky,
- `sticky`: Prvek je umístěn relativně k běžnému toku dokumentu, ale může se „přilepit“ k určité pozici při posouvání stránky. Kombinace `relative` a `fixed`.

css

```
1 body > header {
2   position: fixed;
3   top: 0;
4   left: 0;
5   width: 100%;
6 }
```

top: Pozice prvku od okrajů

Určuje vzdálenost prvku od horního okraje jeho obsahujícího bloku (parent elementu nebo okna prohlížeče, dle typu pozicování).

Podobně fungují i vlastnosti `bottom`, `left` a `right`.

Definice hodnoty:

Jakékoliv číselné hodnoty s jednotkami (viz Kapitola 3.5.6.2).

```css
.element {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

5.5.1 Statické pozicování

Statické pozicování je výchozí způsob, jakým jsou prvky umístěny na stránce. Tedy, až je rozloží obyčejný tok dokumentu, tak poté jsou prvky umístěny na své místo. Nelze je posunout pomocí vlastností `top`, `right`, `bottom` nebo `left`. Toto pozicování je vhodné pro většinu případů, kdy chceme, aby prvky byly umístěny podle běžného toku dokumentu.

Další typy pozicování slouží k tomu, abychom umístili prvky mimo tento běžný tok dokumentu. Důležité pro nás je, že spoustu případů, kdy chcete použít pozicování vede k tomu, že děláte chybu v návrhu stránky. Místo toho, abyste se snažili prvky umístit na konkrétní místo pomocí pozicování, je lepší se zaměřit na správné rozložení stránky pomocí box modelu, flexboxu nebo gridu. Přepoklad je, že pozicování použijete pouze v případech, kdy je to opravdu nutné pro konkrétní efekt nebo funkci.

Nejhorší hřích

Co studenti často dělají, je, že když jim někdo představí pozicování, tak ho začnou používat na všechno možné, jako kdyby flexbox, grid a i čistý box model vůbec neexistoval. To je ale špatně. Pozicování je totiž velmi náchylné na budoucí změny obsahu stránky. Když totiž změníte obsah stránky, například přidáte další text nebo obrázky, tak se může stát, že prvky, které jsou pozicovány absolutně nebo fixně, se začnou překrývat nebo budou mimo obrazovku. To vede k tomu, že stránka přestane fungovat správně a uživatelé budou mít špatný zážitek.

Navíce někteří studenti nepřemýšlí nad responzivitou. Pokud v pozicování budeme používat absolutní hodnoty (např. `px`), tak se stane, že prvky budou v případě změny velikosti obrazovky špatně umístěny. Například pokud chceme dát něco doprostřed, tak nemůžeme vypočítat prostě aktuální šířku prvku a dát ho na polovinu. Místo toho musíme používat relativní jednotky (např. `%`, `vw`, `vh`), nebo využít jiné metody.

5.5.2 Relativní pozicování

Po rozložení prvku podle běžného toku dokumentu můžeme tento prvek posunout relativně k jeho původní pozici pomocí vlastností `top`, `right`, `bottom` a `left`. To znamená, že když nastavíme například `top: 10px`, tak prvek se posune o 10 pixelů dolů od své původní pozice. Ostatní prvky na stránce zůstávají na svém místě, jako kdyby se prvek vůbec neposunul – stejně jako u vlastnosti `transform` či `filter`, viz vlastnost transform a vlastnost filter.

To, že se ostatní prvky nepřizpůsobují je zároveň dobré i špatné. Špatné to může být tím, že nám na stránce vznikne prázdné místo, kde původně prvek byl. Dobré to je tím, že můžeme vytvářet různé efekty. Nejčastěji se ale používá relativní pozicování pro malé posuny, např. pro ikony, které nesedí přesně na svém místě (např. v „řádku textu“). Často ho ale používáme pro definování tzv. zakotvení (anchor), viz Kapitola 5.5.7 a absolutní pozicování.

5.5.3 Absolutní pozicování

Absolutní pozicování nám dovoluje umístit prvek na konkrétní místo na stránce, relativně k nejbližšímu předkovi, který má pozicování jiné než `static`. Nejblíže znamená, že hledáme v rodičovských prvcích směrem nahoru. Pokud takový předek neexistuje, je prvek umístěn relativně k počátku dokumentu (html element), nebo dle kotvy, viz Kapitola 5.5.7.

Původní místo prvku v běžném toku dokumentu je ignorováno a ostatní prvky se podle něj neřídí, resp. **prvek žádné místo nezabírá**. Pozici určujeme pomocí vlastností `top`, `right`, `bottom` a `left`. Toto pozicování je velmi užitečné pro vytváření překryvných prvků, jako jsou modální okna, tooltipy nebo dropdown menu.

Např. pro tzv. dialogy (modální okna) často používáme absolutní pozicování, abychom je umístili doprostřed obrazovky. Máme-li na stránce „kartu“, ve které se okno zobrazuje, tak můžeme jako předka použít tuto kartu s pozicováním `relative`, a okno pak umístit absolutně uvnitř této karty.

html

Struktura karty s modálním oknem

```html
<div class="card">
  <div class="content>
        <!-- obsah karty -->
    </div>
    <div class="modal">
    <div class="content">
      <p>Jste si jistí?</p>
      <button>Ano</button>
      <button>Ne</button>
    </div>
  </div>
</div>
```

css

Styly karty s modálním oknem

```css
.card {
  /* Relativní pozicování, aby .modal mohl být absolutně pozicován
  vůči .card */
  position: relative;
```

```
  width: 15em;
  border: 1px solid #ccc;
  padding: 1em;
}
.modal {
  /* Absolutní pozicování relativně ke .card */
  position: absolute;
  top: 0;
  left: 0;

  /* Vyplnění "backdrop" */
  width: 100%;
  height: 100%;

  background: rgba(0, 0, 0, 0.5);

  /* Zarovnání obsahu */
  display: flex;
  align-items: center;
  justify-content: center;
}
.modal .content {
  background: white;
  padding: 1em;
  border-radius: 5px;
}
```

V předešlých ukázkách (HTML + CSS) jsme vytvořili kartu s modálním oknem, které je absolutně pozicováno relativně ke kartě. Všimněte si, že nikde v pozicování nepoužíváme absolutní hodnoty a vše se počítá z aktuálních rozměrů karty.

Pokud byste chtěli pomocí absolutního pozicování umístit doprostřed, nestačí nám nastavit vlastnosti `top` a `left` na 50%, protože by se prvek zarovnal sice o 50% od levého horního rohu, ale to je více, než je střed. Proto musíme ještě od této hodnoty „odečíst“ polovinu šířky a výšky prvku pomocí vlastnosti `transform`, viz vlastnost transform. Tedy nastavíme `transform: translate(-50%, -50%)`, čímž prvek posuneme zpět o polovinu své šířky a výšky. Tím dosáhneme dokonalého zarovnání na střed.

5.5.4 Fixní pozicování

Fixní pozicování nám dovoluje umístit prvek na konkrétní místo na obrazovce, relativně k oknu prohlížeče. Odborně se jedná o „viewport“[50]. To znamená, že prvek zůstává na stejném místě i při posouvání (scrollování) stránky.

Toto pozicování je velmi užitečné pro vytváření prvků, které mají zůstat viditelné i při posouvání stránky, jako jsou navigační menu, tlačítka pro návrat na začátek stránky nebo různé notifikace. Při nastavení fixního pozicování můžeme opět použít vlastnosti `top`, `right`, `bottom` a `left` k určení přesné pozice prvku na obrazovce, např. `bottom` správně interpretuje spodní okraj okna prohlížeče. Prvek znovu v rámci obyčejného toku nezabírá žádné místo.

[50] Vzpomeňte si na kapitolu responzivity, viz Kapitola 4.4.

To, že nezabírá místo může být velký problém, protože např. pokud budeme dělat „navigační panel“ (header), který bude vždy nahoře na stránce, tak samotný obsah stránky může být skryt pod tímto panelem. Proto je často nutné přidat „padding“ nebo „margin“ na začátek obsahu stránky, aby se zajistilo, že obsah nebude skryt pod fixně pozicovaným prvkem.

5.5.5 Lepivé pozicování

Lepivé pozicování je kombinací relativního a fixního pozicování. Prvek je nejprve umístěn podle běžného toku dokumentu (relativní je v tomto případně myšleno tak, že se na něj mohou ostatní prvky (absolutní) vázat). Když ale uživatel posouvá stránku a prvek dosáhne určité pozice (určené vlastnostmi `top`, `right`, `bottom` nebo `left`), tak se „přilepí“ k této pozici a zůstane tam i při dalším posouvání stránky (fixní chování). Toto pozicování je užitečné pro vytváření prvků, které mají zůstat viditelné na určité pozici při posouvání stránky, jako jsou navigační menu, záhlaví tabulek nebo vedlejší panely ale pouze „do určité míry“.

Vázání sticky je vždy ke svému rodiči. Pokud se dostaneme na konec rodiče, sticky prvek se přestane chovat jako fixní a zůstane na svém místě. Prvek tedy z rodiče nemůže „vyskočit“ ven.

5.5.6 Vrstvy

Již během vlastnosti `transform` jsme si všimli, že stránky nejsou jednoduché 2D plochy, ale že prvky mohou být na sobě navrstveny. To znamená, že některé prvky mohou být vykresleny nad jinými prvky. Mimo zmíněné vlastnosti máme k dispozici i vlastnost `z-index`, která nám dovoluje určit pořadí těchto vrstev.

z-index: Vrstva prvku

Určuje pořadí vrstev prvků na stránce. Prvky s vyšším `z-index` jsou vykresleny nad prvky s nižším `z-index`.

Tato vlastnost funguje pouze pro prvky s pozicováním jiným než `static`.

Definice hodnoty:

Celá čísla (kladná i záporná).

```css
.modal {
  position: fixed;
  z-index: 1000;
}
```

5.5.7 Kotvy

Kotvy představují moderní způsob, jak propojit dva prvky (referenční „kotvu“ a plovoucí prvek) bez nutnosti, aby byly v HTML vnořeny do sebe. Tradiční absolutní pozicování vyžaduje, aby byl rodič nastaven jako `relative`. S kotvami může být referenční prvek (například tlačítko) hluboko ve struktuře stránky, zatímco plovoucí prvek (například tooltip nebo menu) může být umístěn přímo v `<body>`, aby se vyhnul oříznutí pomocí `overflow: hidden`, a přesto zůstane vizuálně „přilepený“ k tlačítku.

Propojení funguje na principu identifikátorů. Prvek, ke kterému se chceme vztahovat, označíme jménem (kotvou) a cílový prvek na toto jméno navážeme. Samotné umístění se pak definuje pomocí funkce `anchor()`, která nahrazuje pevné hodnoty pixelů (např. `top: anchor(bottom)` znamená „moje horní hrana začíná tam, kde končí spodní hrana kotvy“).

anchor-name: Určení kotvy

Tato vlastnost na daném prvku definuje jméno kotvy, na který se mohou ostatní vlastnosti a prvky vázat. Tímto se prvek stane referenčním bodem pro ostatní absolutně pozicované prvky.

Definice hodnoty:

- `none` (výchozí),
- `<dashed-ident>` (název kotvy).

Název kotvy musí začínat dvěma pomlčkami (`--`), například `--moje-kotva`.

```css
.tlacitko {
  anchor-name: --moje-kotva;
}
```

position-anchor: Určení referenční kotvy

Tato vlastnost na daném prvku určí, že tento prvek bude pozicován vůči konkrétní kotvě definované pomocí `anchor-name`.

Tato vlastnost se nastavuje na plovoucím prvku, který má na `position` hodnotu `absolute` nebo `fixed`.

Poté lze použít funkci `anchor()` ve vlastnostech `top`, `left`, `right` nebo `bottom` pro přesné umístění vůči hranám kotvy.

Definice hodnoty:

- `none` (výchozí),
- `<dashed-ident>` (název kotvy).

```css
.tooltip {
  position: absolute;
  position-anchor: --moje-kotva;
  top: anchor(bottom);
  left: anchor(center);
}
```

5.6 Barevné přechody

Barevné přechody (gradients) nám dovolují vytvářet plynulé přechody mezi dvěma nebo více barvami. Tyto přechody můžeme použít jako pozadí prvků pomocí vlastnosti `background` nebo `background-image`, ale i v jiných kontextech, například jako výplň tvarů v SVG.

Přechody využívají interpolaci, o které jsme si povídali v animacích, viz Kapitola 5.4. Klasická interpolace avšak často nestačí, je nutné dělat tzv. korekce barev (např. pomocí gamma korekce), aby přechody vypadaly přirozeně. Nebo je nutné využívat speciální barevné prostory (např. LAB nebo LCH), které lépe odpovídají lidskému vnímání barev, o nich si bonusově můžete přečíst ve skriptech MVOP Webového inženýrství pro 3. ročník.

V základu existuji tří funkce, které nám dovolují vytvářet různé typy přechodů:

- `linear-gradient()`: Vytváří lineární přechod mezi barvami podél přímky. Můžeme určit směr přechodu (např. `to right`, `45deg`) a jednotlivé barvy s jejich pozicemi,
- `radial-gradient()`: Vytváří radiální (kruhový) přechod mezi barvami vycházející z určitého bodu. Můžeme určit střed přechodu, velikost a tvar (např. `circle`, `ellipse`) a jednotlivé barvy s jejich pozicemi,
- `conic-gradient()`: Vytváří kuželový přechod mezi barvami kolem určitého bodu. Můžeme určit střed přechodu a jednotlivé barvy s jejich pozicemi.

Mezi další varianty patří opakující se přechody (`repeating-linear-gradient()`, `repeating-radial-gradient()`, `repeating-conic-gradient()`), které nám dovolují vytvářet vzory s opakujícími se přechody.

Níže jsou ukázky použití všech typů přechodů se zápisem v CSS.

Obrázek 16: Lineární přechod

css

Lineární přechod

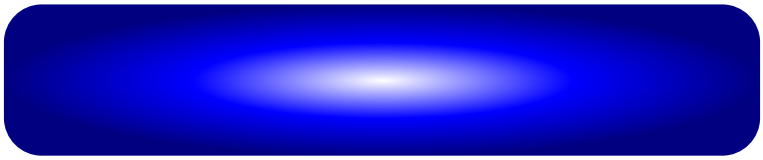
```
1 background: linear-gradient(to right, red, orange, yellow, green, blue,
  indigo, violet);
```

Obrázek 17: Lineární přechod pod úhlem

css

Lineární přechod pod úhlem

```
1 background: linear-gradient(40deg, blue, indigo, violet);
```

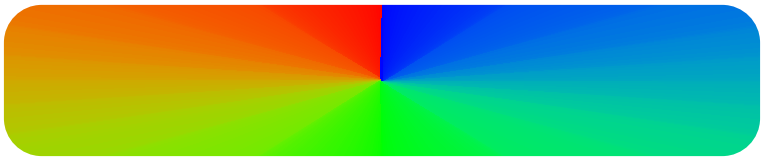


Obrázek 18: Radiální přechod

css

Radiální přechod

```
1 background: radial-gradient(white, blue, navy);
```



Obrázek 19: Kuželový přechod

Kuželový přechod

```css
background: conic-gradient(red, green, blue);
```

5.7 Frameworky

Pojem framework (v češtině někdy rámec) označuje v softwarovém inženýrství sadu knihoven, nástrojů a best-practices, které poskytují základní strukturu pro vývoj aplikací. Zatímco knihovna (library) je soubor funkcí, které vy voláte ve svém kódu, framework naopak často volá váš kód. Tento princip se nazývá „Inversion of Control“. Framework nám dává „lešení“, na kterém stavíme, a nutí nás dodržovat určitá pravidla a architekturu.

Mimo webový vývoj se s frameworky setkáváme například při vývoji desktopových aplikací (.NET, Qt) nebo backendových systémů (Spring, Django, ASP.NET Core).

Hlavním cílem frameworku je zrychlit vývoj tím, že řeší opakující se problémy (např. připojení k databázi, routování požadavků nebo v případě CSS konzistentní vzhled) a vývojář se tak může soustředit na unikátní byznys logiku aplikace.

5.7.1 Webové CSS frameworky

Ve světě CSS slouží frameworky k tomu, aby vývojář nemusel psát styly pro každou komponentu od nuly. Poskytují předpřipravené třídy a komponenty (tlačítka, formuláře, navigační lišty, modální okna), které vypadají dobře „out-of-the-box“ a jsou testovány na různých prohlížečích. Historicky frameworky řešily především nekonzistenci mezi prohlížeči a složité vytváření layoutů (v dobách před Flexboxem a Gridem, viz Kapitola 4.2 a Kapitola 4.3). Dnes se zaměřují spíše na efektivitu psaní kódu, udržitelnost designového systému a responzivitu.

Designový framework je sada předdefinovaných stylů a komponent, které zajišťují konzistentní vzhled a chování webových aplikací. Pokud by designové systémy neexistovali, každý vývojář by musel vytvářet vlastní styly a komponenty od nuly, což by vedlo k nekonzistentnímu vzhledu a chování aplikací. Ao to by vedlo k nespokojeným uživatelům, protože by vše fungovalo a vypadalo zcela jinak.

Jednotlivé brandy si definují své designové příručky, které popisují barvy, typografii, ikonografii, rozložení a interakce. Tyto příručky pak implementují do designových systému a frameworků, které vývojáři používají při tvorbě aplikací.

Mezi nejznámější designové systémy patří:

- Material Design od Google,
- Carbon Design od IBM,
- Fluent Design od Microsoftu,
- Human Interface od Apple.

5.7.2 JavaScriptové frameworky

Velmi krátce se dotkneme i JavaScriptových frameworků, ačkoli ty řeší logiku a interaktivitu, nikoliv primárně stylování. Moderní webový vývoj je dominován frameworky (nebo knihovnami s ekosystémem frameworku) jako jsou **React**, **Vue**, **Angular** či **Svelte**. Tyto nástroje slouží k vytváření tzv. Single Page Applications (SPA) a řeší správu stavu aplikace, vykreslování DOMu a komunikaci se serverem. Často se CSS frameworky integrují přímo do těchto JS frameworků (např. pomocí komponentních knihoven jako Material UI nebo Vuetify). O JavaScriptových frameworcích se více dozvíte v skriptech předmětu MVOP Webového inženýrství. O JS samotném v další kapitole, viz Kapitola 6.2.

5.7.3 Kdy frameworky používat a kdy ne

Použití frameworku není automaticky správná volba pro každý projekt. Je nutné zvážit výhody a nevýhody.

Kdy framework použít:

- **Rychlý prototyping:** Potřebujete rychle vytvořit funkční ukázku (MVP) a nezáleží tolik na unikátním designu,
- **Týmová spolupráce:** Framework zavádí standardy. Když přijde nový vývojář k projektu postaveném na Bootstrapu nebo Tailwindu, okamžitě ví, jak psát styly, aniž by musel studovat stovky řádků vlastního CSS,
- **Administrační rozhraní:** U interních systémů je důležitější funkčnost a konzistence než unikátní grafika,
- **Termíny:** Frameworky šetří čas, protože nemusíte řešit „triviální“ problémy jako stylování formulářových prvků napříč prohlížeči.

Kdy framework nepoužívat:

- **Výuka:** Pokud se učíte CSS, frameworky vám jen uškodí. Skryjí před vámi principy fungování (Box model, specificita, kaskáda). Naučte se základy, až pak nástroje.
- **Jednoduché weby:** Pro malou vizitku nebo jednostránkový web je importování celého frameworku zbytečnou zátěží (bloat), která zpomaluje načítání.
- **Unikátní design:** Pokud máte od grafika velmi specifický design, který neodpovídá standardním šablonám, strávíte více času přepisováním frameworku než psaním vlastního CSS.

Můžete si ale vytvořit vlastní framework, komponenty, které budete znovu používat napříč projekty. To je ideální cesta, jak mít výhody frameworku (rychlost, konzistence) bez jeho nevýhod (zbytečný kód, omezení designu).

5.7.4 Tailwind CSS a utility-first přístup

V posledních letech dominuje světu CSS frameworků **Tailwind CSS**. Přišel s přístupem zvaným **Utility-First**. Namísto toho, aby nabízel hotové komponenty jako `.btn-primary` (semantické CSS), nabízí tisíce malých tříd, které dělají jednu konkrétní věc.

html

Srovnání tradičního a utility-first přístupu

```
1 <button class="btn btn-primary">Klikni</button>
2
```

```
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2
  px-4 rounded">
  Klikni
</button>
```

Tyto třídy se často sestavují až po tom, co speciální program přečte HTML soubor a vygeneruje výsledné CSS. Případně můžete přímo říci, jaké typy tříd chcete generovat (např. pouze barvy a mezery).

Výhody:

- **Rychlost:** Nemusíte vymýšlet názvy tříd (což je jedna z nejtěžších věcí v programování) a neustále přeskakovat mezi HTML a CSS souborem,
- **Malá velikost ve finále:** Pomocí nástrojů se z výsledného CSS odstraní všechny nepoužité třídy, takže výsledný soubor je malý,
- **Konzistence:** Jste nuceni vybírat z předdefinované palety barev a mezer, což drží design vizuálně pohromadě.

Nevýhody a kritika:

- **„Class soup" (Polévka tříd):** HTML kód se stává nepřehledným, dlouhým a spojujete technologie do jednoho celku, který by měl být oddělený (strukturální HTML vs. prezentační CSS),
- **Znovupoužitelnost:** Pokud máte tlačítko na 10 místech, musíte buď kopírovat dlouhý řetězec tříd, nebo použít komponentní přístup (v JS frameworku) či direktivu `@apply` (což jde proti filozofii Tailwindu),
- **Přenos dat**: Sic se ušetří místo na CSS, ale HTML soubor je větší, což může zpomalit načítání stránky,
- **Nutnost build processu:** Tailwind nelze efektivně používat bez preprocesorů a Node.js prostředí.

Častá kritika zní, že je to jen „inline stylování s extra kroky". Není to tak jednoduché, protože Tailwind umí řešit věci jako responzivitu, pseudo-třídy (hover, focus) – to v inline stylech nejde.

Použití Tailwindu je nutné pro každý projekt zvážit. Já jej pro studenty doporučuji až po zvládnutí základů CSS (což může přijít až za několik *let*).

5.7.5 Bootstrap a historie

Nelze mluvit o frameworcích a nezmínit Bootstrap. Vytvořený v Twitteru, byl to první framework, který masivně zpopularizoval responzivní webdesign a systém 12sloupcového gridu (ještě před nativním CSS Gridem). Dlouhou dobu byl standardem pro tvorbu webů.

Dnes je považován za poněkud zastaralý („relikt"), a to hlavně kvůli své mohutnosti a faktu, že weby postavené na Bootstrapu vypadají „všechny stejně". Přesto je stále velmi používaný v enterprise sféře a pro rychlé prototypy.

12sloupcový grid Bootstrapu byl revoluční v době, kdy CSS Grid a Flexbox ještě nebyly široce podporovány. Tento grid způsobuje, že na stránkách můžeme tvořit řádky, které se automaticky přizpůsobují velikosti obrazovky. Do daného řádku poté vkládáme jednotlivé prvky, které vždy zabírají určitý počet sloupců (např. 6 z 12 pro polovinu šířky). Dané prvky pak můžeme jednoduše (např. pomocí utility-tříd) změnit, resp. změnit jejich šířku (počet sloupců) pro různé velikosti obrazovky. Například `col-12 col-md-6` znamená, že prvek zabírá 12 sloupců (celou šířku) na malých obrazovkách a 6 sloupců (polovinu šířky) na středních a

větších obrazovkách. Klíčové znaky `md` odpovídají jednotlivým bodům zlomu (breakpoints), stejně jak jsme si ukazovali v obyčejném CSS, viz Kapitola 4.4.

5.7.6 Další moderní frameworky

Kromě Tailwindu a Bootstrapu existuje řada dalších zajímavých nástrojů:

- **Bulma:** Čistě CSS framework (bez JS) založený na Flexboxu. Velmi čitelný a snadný na naučení.
- **Foundation:** Pokročilý framework pro profesionální a přístupné weby, historický konkurent Bootstrapu.
- **Pico.css / MVP.css:** Minimalistické frameworky, které stylovaly pouze základní HTML elementy bez nutnosti přidávat třídy. Ideální pro dokumentace nebo velmi jednoduché textové weby.
- **Open Props:** Moderní knihovna postavená na CSS proměnných (Variables), která nedává hotové komponenty, ale sadu proměnných pro barvy, stíny a animace. Je to jakýsi „střední krok“ mezi vlastním CSS a Tailwindem.

5.8 Vlastní design

V ideálním světě by každý projekt měl mít svůj vlastní design, vytvořený na míru podle potřeb uživatelů a byznysových cílů. To zahrnuje spolupráci s UX/UI designéry, kteří vytvářejí wireframy, prototypy a finální vizuály.

Cílem těchto skript a předmětu však není naučit vás být designéry, ale spíše dát vám nástroje a znalosti, jak implementovat design vytvořený profesionály. Přesto je důležité, abyste měli určité povědomí o designových principech, abyste mohli efektivně komunikovat s designéry a rozumět jejich potřebám.

5.8.1 Figma

Figma je populární moderní nástroj pro UI/UX design, který běží přímo v prohlížeči. Umožňuje týmovou spolupráci v reálném čase, vytváření prototypů a export designů do různých formátů. Ve Figmě je možné vytvářet design, webové stránky, mobilní aplikace a další digitální produkty.

Mezi netriviální výhody patří právě implementace designových systémů. Protože si můžeme definovat tzv. „styles“ (barvy, typografie, efekty) a „components“ (tlačítka, formuláře, karty), které lze opakovaně používat napříč projekty. To zajišťuje konzistenci designu a urychluje práci designérů.

V rámci studia na Smíchovské střední průmyslové škole se s Figmou pravděpodobně setkáte v předmětech zaměřených grafické systémy, tedy GRS, případně později na specifičtějších MVOP kurzech. I v rámci MVOP Webového inženýrství se jde do designu více do hloubky.

5.8.2 Uživatelský zážitek

Design dělíme do dvou hlavních oblastí:

- Uživatelské rozhraní (UI, User Interface): Jak web vypadá, jak jsou uspořádány prvky na stránce, jaké barvy a typografie jsou použity,

- Uživatelská zkušenost (UX, User Experience): Jak uživatel interaguje s webem, jak snadné je najít informace, jak rychle se načítají stránky, jak intuitivní je navigace.

Pro ně definujeme několik základních principů, které byste měli mít na paměti při tvorbě webových stránek:

- Jednoduchost: Udržujte design čistý a jednoduchý. Příliš mnoho prvků může uživatele zmást,
- Konzistence: Používejte konzistentní barvy, typografii a rozložení napříč celým webem,
- Responzivita: Zajistěte, aby web fungoval dobře na různých zařízeních a velikostech obrazovky,
- Přístupnost: Ujistěte se, že váš web je přístupný pro všechny uživatele, včetně těch s hendikepy (např. pomocí kontrastru barev, alternativních textů pro obrázky a dalších technik),
- Rychlost: Optimalizujte načítání stránek, aby uživatelé nemuseli dlouho čekat,
- Intuitivní navigace: Ujistěte se, že uživatelé mohou snadno najít to, co hledají.

Zásad existuje samozřejmě mnohem více. V rámci WBA, resp. pozdějších úkolů, ve kterých tvoříte vlastní design, je důležité mít tyto principy na paměti. Nejlepší tipy pro tvorbu jsou:

- Přemýšlejte jako uživatel, ne jako vývojář,
- Testujte svůj design s reálnými uživateli a sbírejte zpětnou vazbu,
- Přemýšlejte nad designem a netvořte ho jen „protože to tak vypadá dobře“.

Interaktivita

6.1 Formuláře

Formuláře jsou nezákladnějším způsobem, jak mohou uživatelé komunikovat s webovými stránkami. Pomocí formulářů mohou uživatelé zadávat data, která jsou následně odeslána na server k dalšímu zpracování. Formuláře se skládají z různých prvků, jako jsou textová pole, tlačítka, zaškrtávací políčka, přepínače a další. Sémanticky se správně definují jako vstup uživatele a nemusí nutně sloužit jen jako formuláře pro odeslání dat na server.

V rámci HTML máme k dispozici ruzné prvky pro tvorbu formulářů, které si postupně představíme.

<form>: Formulář

Definuje formulář a obaluje různé prvky formuláře.

Atributy:

method Určuje jakým způsobem se budou data předávat další „vrstvě“ stránky. Možné hodnoty jsou `GET`, `POST` a `dialog`. Výchozí hodnota je `GET`.

action Určuje URL, kam budou data odeslána po odeslání formuláře. Může být relativní nebo absolutní URL. Pokud není atribut `action` uveden, data se odešlou na stejnou URL jako je aktuální stránka. Samotná stránka pak musí být schopna tato data zpracovat.

```html
<form action="/submit" method="POST">
  <!-- Prvky formuláře zde -->
</form>
```

Většinou prvků formuláře je potřeba umístit do značky `<form>`, která definuje samotný formulář. Tento prvek může obsahovat různé atributy, jako je `action`, který určuje URL, kam budou data odeslána, a `method`, který určuje HTTP metodu (např. GET nebo POST) použitou při odesílání dat. Data můžeme odesílat i jinými způsoby, a to, jak se liší tyto způsoby si ukážeme později, viz Kapitola 6.1.5.

6.1.1 Prvky formuláře

Nyní si představíme jednotlivé prvky, které můžeme v rámci formulářů používat.

6.1.1.1 Všeobecná nastavení

Skoro každému prvku můžeme nastavovat speciální atributy a abychom je zbytečně neopakovali u každého prvku, představíme si je zde.

Atribut `name` slouží k pojmenování prvku formuláře. Toto jméno je důležité, protože se používá jako klíč při odesílání dat na server. Takže když uživatel vyplní jedno pole, které atribut `name` má hodnotu `email`, server obdrží tyto data právě pod klíčem `email`. Klíče se tedy v jednom formuláři nesmí opakovat.

Atribut `value` určuje výchozí hodnotu prvku formuláře. Například u textového pole může být tato hodnota předvyplněný text, který se zobrazí uživateli. To děláme např. při úpravě již

existujících dat – nechceme, aby si uživatel musel vzpomínat jak předtím data byly, když chce změnit např. jedno pole.

Atribut `placeholder` slouží k zobrazení nápovědného textu uvnitř vstupního pole, když je prázdné. Tento text obvykle poskytuje uživateli informaci o tom, jaký typ dat by měl do pole zadat. Například v textovém poli pro e-mail může být placeholder „Zadejte svůj e-mail“. Resp. by spíše mělo být „pepa@test.com“, protože to je konkrétní příklad. To co se udává do prvku by mělo být naznačeno popisek, viz Kapitola 6.1.2.

Atribut `id` slouží k jednoznačné identifikaci prvku na stránce, jak již známe z předchozích kapitol. Ve formulářích je často používán ve spojení s prvkem `<label>`, který umožňuje přiřadit popisek k určitému prvku formuláře. Tímto způsobem můžeme zajistit lepší přístupnost a uživatelskou přívětivost formulářů.

Atribut `disabled` označuje, že prvek formuláře je zakázán a uživatel s ním nemůže interagovat.

Atribut `readonly` označuje, že prvek formuláře je pouze pro čtení a uživatel nemůže měnit jeho hodnotu.

6.1.1.2 Základní typy

`<input>`: Vstupní pole

Definuje různé typy vstupních polí v závislosti na hodnotě atributu `type`. Může být použit pro textová pole, zaškrtávací políčka, přepínače, tlačítka a další.

Atributy:

type Určuje typ vstupního pole. Mezi možné hodnoty patří:

- `text`: Jednořádkové textové pole,
- `password`: Pole pro zadání hesla (text je skrytý),
- `checkbox`: Zaškrtávací políčko,
- `radio`: Přepínač (výběr z více možností),
- `submit`: Tlačítko pro odeslání formuláře,
- `reset`: Tlačítko pro resetování formuláře,
- `button`: Obecné tlačítko,
- `email`: Pole pro zadání e-mailové adresy,
- `number`: Pole pro zadání číselné hodnoty,
- `date`: Pole pro zadání data,
- a další.

```html
<input type="text" name="username" placeholder="Uživatelské jméno">
```

Tag `<input>` je velmi univerzální a pomocí atributu `type` můžeme definovat různé druhy vstupních polí. Nyní si ukážeme jak jednotlivé typy fungují.

Textová pole

Při vytvoření `<input type="text">` získáme jednoduché jednořádkové textové pole, do kterého může uživatel zadat libovolný text. Uživatel může kliknout do pole a začít psát.

Číslo

Pomocí `<input type="number">` vytvoříme pole, které umožňuje zadávat pouze číselné hodnoty. Pro usnadnění zadávání čísel můžeme přidat šipky pro zvýšení nebo snížení hodnoty. Je možné také nastavit minimální a maximální hodnoty pomocí atributů `min` a `max`. Nebo můžeme určit krok změny hodnoty pomocí atributu `step`.

E-mail

Pomocí `<input type="email">` vytvoříme pole určené pro zadání e-mailové adresy. Tento typ pole může také provádět základní validaci formátu e-mailové adresy, což pomáhá zajistit, že uživatel zadá platnou e-mailovou adresu. Validace ale dovoluje jakoukoliv hodnotu, která vypadá jako e-mail (obsahuje zavináč apod.). E-maily ale mohou mít (pro některé studenty) nečekané formáty, např. `a@localhost` je validní e-mailová adresa.

Heslo

Pomocí `<input type="password">` vytvoříme pole pro zadání hesla. Text zadaný do tohoto pole je skrytý (zobrazen jako tečky nebo hvězdičky), aby se zabránilo jeho viditelnosti pro ostatní osoby. Zadané heslo nejde kopírovat uživatelem. Na server se ale posílá v čisté podobě[51].

Datum

Máme k dispozici různé typy pro zadávání data a času, např. `<input type="date">`, `<input type="time">`, `<input type="datetime-local">` či `<input type="month">`. Tyto typy poskytují uživateli speciální rozhraní pro výběr data a času, což usnadňuje zadávání těchto hodnot. Například `<input type="date">` zobrazí kalendář pro výběr data.

Vyskakované okno se na různých zařízeních mění dle konkrétního prohlížeče. Na mobilních zařízení se otevře nativní výběr data, který je optimalizovaný pro dotykové ovládání přímo v daném operačním systému. Na destopu je to většinou jednoduchý kalendář. Tyto prvky avšak nemůžeme stylovat pomocí CSS, proto je nějaké stránky nepoužívají a nahrazují je vlastními implementacemi. Na pozadí přesto tento prvek existuje, aby „data“ uchovával.

Skryté pole

Pomocí `<input type="hidden">` vytvoříme skryté pole, které není viditelné pro uživatele. Tento typ pole se často používá k ukládání dat, která nejsou určenaa přímou interakci uživatele, ale jsou potřebná pro odeslání formuláře. Například můžeme uložit ID uživatele nebo token pro ověření.

Zaškrtávací políčko

Pomocí `<input type="checkbox">` vytvoříme zaškrtávací políčko, které umožňuje uživateli vybrat nebo zrušit výběr jedné nebo více možností. Uživatel může kliknout na políčko pro změnu jeho stavu (zaškrtnuto/nezaskrtnuto).

[51] HTTPS posílá data zašifrovaně, ale server je přijme v čisté podobě.

Pro určení stavu zaškrnutí můžeme použít atribut `checked`. Pokud je tento atribut přítomen, políčko bude ve výchozím stavu zaškrtnuté.

Přepínač

Pomocí `<input type="radio">` vytvoříme přepínač, který umožňuje uživateli vybrat jednu možnost z předdefinované skupiny možností. Všechny přepínače, které patří do jedné skupiny, musí mít stejný atribut `name`.

Příklad přepínačů ve formuláři

```html
<input type="radio" name="color" value="red"> Červená
<input type="radio" name="color" value="green"> Zelená
<input type="radio" name="color" value="blue"> Modrá
```

Výše uvedený kód vytvoří tři přepínače pro výběr barvy. Po zakliknutí jednoho z přepínačů se ostatní automaticky odkliknou, protože patří do stejné skupiny (mají stejný `name`). Navíce vybrání nelze být zrušeno kliknutím na již vybraný přepínač[52].

Soubor

Pomocí `<input type="file">` vytvoříme pole pro výběr souboru z počítače uživatele. Uživatel může kliknout na tlačítko pro otevření dialogu pro výběr souboru.

Dovnitř pole se uloží obsah souboru, který uživatel vybral. To může používat samotná stránka nebo se mohou soubory odeslat na server jako součást formuláře.

Je možné přidat atribut `multiple`, který umožní uživateli vybrat více souborů najednou. Dalším možným atributem je `accept`, který určuje typy souborů, které jsou povoleny pro výběr (např. obrázky, dokumenty apod.). Zadáváme se pomocí MIME typů nebo přípon souborů. Například `accept="image/*"` povolí všechny typy obrázků, zatímco `accept=".pdf,.docx"` povolí pouze PDF a DOCX soubory[53].

Ne všechy způsoby odesílání formulářů podporují odesílání souborů.

Odeslání

Pomocí `<input type="submit">` vytvoříme tlačítko pro odeslání formuláře. Když uživatel klikne na toto tlačítko, formulář se odešle na server podle nastavení atributů `action` a `method` ve značce `<form>`.

Reset

Pomocí `<input type="reset">` vytvoříme tlačítko pro resetování formuláře. Když uživatel klikne na toto tlačítko, všechna pole formuláře se vrátí do výchozího stavu (hodnoty před vyplněním uživatelem).

[52] Je to možné ale např. pomocí reset tlačítka.

[53] Přípony souboru ale nejsou spolehlivá metoda, protože uživatel může změnit příponu souboru ručně. Doporučuje se vždy validovat typ souboru na serveru.

Další typy

Mezi další typy tagu `<input>` řadíme např.:

- `color`: Pole pro výběr barvy,
- `range`: Posuvník pro výběr hodnoty z určitého rozsahu,
- `tel`: Pole pro zadání telefonního čísla,
- `url`: Pole pro zadání URL adresy,
- `search`: Pole pro zadání vyhledávacího dotazu,
- `datetime-local`: Pole pro zadání data a času bez časového pásma,
- `week`: Pole pro zadání týdne v roce,
- A další.

6.1.1.3 Víceřádkové textové pole

<textarea>: Víceřádkový vstup

Definuje víceřádkové textové pole, do kterého může uživatel zadat delší text. Na rozdíl od `<input type="text">`, které je omezeno na jeden řádek, `<textarea>` umožňuje zadávat text přes více řádků.

Atributy:

rows Určuje počet viditelných řádků v textovém poli.

cols Určuje počet viditelných sloupců (šířku) v textovém poli.

```html
<textarea name="message" rows="4" cols="50" placeholder="Zadejte svou zprávu zde..."></textarea>
```

Základní hodnotu pro `<textarea>` můžeme zadat mezi otevírací a zavírací značku. Pozor ale na nové řádky a mezery – ty se v rámci HTML berou jako součást obsahu.

resize: Možnost změny velikosti prvku

Dovolí uživateli měnit velikost prvku (např. textového pole).

Nejčastěji používáme u prvků formuláře. Lze použít ale např. i na panely, obrázky apod.

Definice hodnoty:

- none (výchozí),
- both,
- horizontal,
- vertical,
- block,
- inline.

```css
textarea {
  resize: vertical;
}
```

6.1.1.4 Tlačítka

<button>: Tlačítko

Definuje tlačítko, které může spouštět různé akce, jako je odeslání formuláře nebo spuštění JavaScriptové funkce.

Atributy:

type Určuje typ tlačítka. Mezi možné hodnoty patří:
- `button`: Obecné tlačítko (výchozí),
- `submit`: Tlačítko pro odeslání formuláře,
- `reset`: Tlačítko pro resetování formuláře.

```html
<button type="button">Klikni na mě</button>
```

Tlačítka v rámci HTML definujeme buď pomocí prvku `<button>`, nebo pomocí `<input type="button">`. V rámci formuláře ale má smysl používat spíše `<input type="submit">` a `<input type="reset">`, protože tyto prvky mají implicitní chování pro odeslání a resetování formuláře. Klasický tag `<button>` je vhodný pro obecná tlačítka, která neodesílají formulář, navíce do nich můžeme vložit i jiné HTML prvky (např. ikony).

6.1.2 Popisky

Popisky jsou nejdůležitější částí formulářů, protože poskytují uživateli informace o tom, co má do jednotlivých polí zadat. Pro přiřazení popisku k určitému prvku formuláře používáme prvek `<label>`. Pokud je popisek správně přiřazen k prvku, kliknutí na popisek se přenese na odpovídající prvek (např. zaškrtávací políčko nebo přepínač), což zlepšuje uživatelskou přívětivost a přístupnost formulářů. Přiřazení popisku k prvku provedeme pomocí atributu `for`, který obsahuje hodnotu atributu `id` odpovídajícího prvku formuláře.

<label>: Popisek

Definuje popisek pro prvek formuláře.

Atributy:

for Určuje ID prvku formuláře, ke kterému je popisek přiřazen.

```html
<label for="username">Uživatelské jméno:</label>
<input type="text" id="username" name="jmeno">
```

6.1.3 Pokročilé prvky

Dále si ukážeme další pokročilé prvky, které můžeme ve formulářích používat. Tyto prvky nám umožňují vytvářet složitější a interaktivnější formuláře.

Pro formuláře toho existuje hodně, a to je nad rámec povinných znalostí, a proto je naleznete později, viz Kapitola 7.1.

6.1.3.1 Výběrové pole

Tag `<select>` slouží k vytvoření rozbalovacího výběrového pole, které umožňuje uživateli vybrat jednu nebo více možností ze seznamu. Uvnitř tagu `<select>` používáme tag `<option>` pro definování jednotlivých možností.

<select>: Výběrové pole

Definuje rozbalovací výběrové pole s různými možnostmi.

Atributy:

multiple Umožňuje uživateli vybrat více možností najednou.

```html
<select name="cars" id="car-select">
  <option value="">--Vyberte auto--</option>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

<option>: Možnost výběru

Definuje jednotlivou možnost v rámci výběrového pole `<select>`.

Atributy:

value Určuje hodnotu, která bude odeslána na server, když je tato možnost vybrána.

```html
<option value="volvo">Volvo</option>
```

Toto výběrové okno je znovu závislé na prohlížeči a operačním systému. V moderním CSS je ale možné jej částečně stylovat pomocí pseudoelementu `::picker`.

Dalším výběrem je prvek `<datalist>`, který umožňuje definovat seznam předdefinovaných možností pro vstupní pole. Tento prvek je spojen s `<input>` pomocí atributu `list`, což umožňuje uživateli vybírat z předdefinovaných možností při zadávání textu. Oproti `<select>` je ale možné i zadat vlastní hodnotu, která není v seznamu. Takže si lze představit `<datalist>` spíše jako doporučení, možné hodnoty, než jako pevně daný výběr.

<datalist>: Seznam možností

Definuje seznam předdefinovaných možností pro vstupní pole.

```html
<input list="browsers" name="browser" id="browser">
<datalist id="browsers">
```

```
  <option value="Chrome">
  <option value="Firefox">
  <option value="Safari">
</datalist>
```

6.1.3.2 Zobrazení výsledků

Poslední prvek, který si ukážeme, je `<output>`, který slouží k zobrazení výsledků výpočtů nebo interakcí na stránce. Tento prvek je užitečný pro zobrazování dynamicky generovaných hodnot, které se mění na základě uživatelského vstupu nebo jiných akcí na stránce.

<output>: Zobrazení výsledků

Definuje oblast pro zobrazení výsledků výpočtů nebo interakcí na stránce.

Atributy:

for Určuje ID prvků, jejichž hodnoty jsou použity pro výpočet výsledku.

html

```
<output name="result" for="input1 input2">Výsledek se zobrazí zde</
output>
```

6.1.4 Validace formulářů

HTML poskytuje vestavěné mechanismy pro validaci formulářů, které nám umožňují zajistit, že uživatel zadá správná data před odesláním formuláře. Můžeme použít různé atributy pro validaci, jako jsou `required`, `minlength`, `maxlength`, `pattern` a další. Samozřejmě jsme si již uvedli nějaké ukázky dříve – např. u čísla atributy `min` a `max`.

Atribut `required` označuje, že pole musí být vyplněno před odesláním formuláře. Při odeslání (pomocí tlačítka `submit`) prohlížeč zkontroluje, zda jsou všechna povinná pole vyplněna. Pokud nejsou, zobrazí se uživateli chybová zpráva u daného pole a formulář se neodešle.

Atributy `minlength` a `maxlength` určují minimální a maximální délku textu, který může uživatel zadat do textového pole. Pokud uživatel zadá text, který je kratší než `minlength` nebo delší než `maxlength`, prohlížeč zobrazí chybovou zprávu a formulář se neodešle.

Atribut `pattern` umožňuje definovat regulární výraz, který musí zadaný text odpovídat. Pokud uživatel zadá text, který neodpovídá zadanému vzoru, prohlížeč zobrazí chybovou zprávu a formulář se neodešle. Například můžeme použít `pattern="[A-Za-z]{3,}"` pro povolení pouze písmen s minimální délkou 3 znaky. Pro validaci se používá JavaScriptový regulární výraz, takže je potřeba dodržovat jeho syntaxi.

Některé tyto validace avšak nejsou dostatečné pro konkrétní použití. Proto se často používá i vlastní validace pomocí JavaScriptu, která nám umožňuje provádět složitější kontroly a poskytovat uživateli lepší zpětnou vazbu.

Důležité ale je, že vstupu od uživatele, jakkoliv je validovaný na straně klienta (v prohlížeči), nemůžeme věřit. Na serveru je vždy potřeba data znovu validovat, protože uživatel může validaci obejít (např. pomocí nástrojů pro vývojáře v prohlížeči).

6.1.5 Způsob odeslání dat

Při odesílání formulářů máme k dispozici různé způsoby, jak mohou být data odeslána na server. Řešíme dva hlavní problémy:

- Jakým způsobem se data přenesou,
- Jaký formát dat se použije pro přenos.

Pro způsoby přenosu máme dvě hlavní metody[54]:

- `GET`: Data jsou připojena k URL jako dotazovací řetězec,
- `POST`: Data jsou odeslána v těle HTTP požadavku.

Způsob přenosu pomocí `GET` pro nás znamená, že zadaná data se přenášejí uvnitř URL. Přesněji v části parametrů, tedy za otazníkem (`?`). Jednotlivé hodnoty se pojují pomocí ampersandu (`&`). Jednu hodnotu dělíme pomocí rovnítka (`=`) na dvě části, kde vlevo je jméno (klič, viz `name` atribut) a vpravo je hodnota (viz `value` atribut nebo zadaný text). Tento způsob je vhodný pro jednoduché formuláře, kde nejsou přenášena citlivá data, protože URL může být viditelná v historii prohlížeče nebo v logách serveru.

Při použití metody `POST` jsou data odeslána v těle HTTP požadavku, což znamená, že nejsou viditelná v URL. Tento způsob je vhodný pro formuláře, které přenášejí citlivá data, jako jsou hesla nebo osobní informace.

Pro formát dat máme tři hlavní možnosti, které určujeme pomocí atributu `enctype` ve značce `<form>`:

- `application/x-www-form-urlencoded`: Výchozí formát, kde jsou data kódována jako páry klíč-hodnota,
- `multipart/form-data`: Stále se ukládají jako páry klíč-hodnota, ale umožňuje odesílání souborů,
- `text/plain`: Jako prostý text, často jen pro ladění.

Ihned z tohoto vidíme, že primární způsob odesílání formulářů je pomocí `multipart/form-data`, protože často chceme odesílat i soubory.

V URL adrese při použití `GET` je potřeba speciální znaky kódovat pomocí procentového kódování (URL encoding). Například mezera se kóduje jako `%20`, znak `@` jako `%40` atd.

> **i Časté využití GET**
>
> Metoda `GET` se často používá pro vyhledávací formuláře, kde uživatel zadává dotaz a očekává výsledky. Výhodou použití `GET` je, že výsledky vyhledávání mohou být snadno sdíleny pomocí URL. Například vyhledávací formulář na webu může mít URL jako `https://example.com/search?query=html+forms`, což umožňuje uživateli sdílet tento odkaz s ostatními.
>
> Query parametry je taktéž jednoduše možné číst v rámci JavaScriptu na webové stránce, což může být užitečné pro dynamické načítání obsahu na základě zadaných parametrů.

[54] Zmíněný `dialog` je speciální režim pro modální dialogy a není běžně používaný pro odesílání dat na server.

6.1.6 Stylování formulářů

Stylování formulářů probíhá stejně, jako u jiných HTML prvků. Jedinou novou vlastností je zmíněná `resize`, kterou můžeme použít u víceřádkových textových polí.

Jednotlivé prvky seskupujeme často s jejich popisky do bloků pomocí prvku `<div>`[55]. To nám umožňuje lépe ovládat rozložení formuláře pomocí CSS.

Jako poslední část k formulářům si ukážeme, jak si můžeme nastylovat vlastní zaškrtávací políčka. V rámci obyčejného CSS totiž je nelze v podstatě vůbec stylovat.

6.1.6.1 Zaškrtávací políčko

Prvně se zamyslíme nad tím, jak bychom si vlastní takto interaktivní prvek vytvořili. První myšlenka by měla být „jak udělám klikací změnu stavu?“. To přeci neumíme – JavaScript následuje až v další kapitole.

Řešení je ale jednoduché – využijeme k tomu samotný HTML prvek. Musíme si vzpomenout na vlastnost `<label>`, která nám umožňuje přiřadit popisek k určitému prvku. Navíce ale způsobuje, že kliknutím na popisek se přenese kliknutí na samotný prvek – to je užitečné pro to, abychom nemuseli klikat pouze „na čtvereček“, ale i na text vedle něj.

Tento princip využijeme k vytvoření vlastního zaškrtávacího políčka. Další myšlenkou by měla být „jak zjistím v CSS, jestli je prvek zaškrtnutý?“. Na to máme řešení v CSS, ukážeme si totiž pseutořídu `:checked`, která se zaktivuje pokud je prvek typu checkbox nebo radio zaškrtnutý.

Selektor: Zaškrtnutí prvku

Umožňuje cílit na zaškrtnuté prvky formuláře, jako jsou zaškrtávací políčka a přepínače.

```css
input:checked {
  background-color: #4CAF50;
}
```

Implementujeme vlastní zaškrtávací políčko tak, že:

- Stále budeme mít prvek `<input type="checkbox">`, ale ten skryjeme např. pomocí `display: none;`,
- Vytvoříme si vlastntí prvek, který bude reprezentovat zaškrtávací políčko s vlastním designem (bez omezení prohlížeče),
- Pomocí `:checked` změníme vzhled našeho vlastního prvku, když je původní prvek zaškrtnutý,
- Obalíme vše do `<label>`, aby bylo možné kliknout na celý blok (čtvereček i text).

Začneme s HTML:

HTML pro vlastní zaškrtávací políčko

```html
<label class="custom-checkbox">
```

[55] V bonusové sekci, viz Kapitola 7.1, si ukážeme i speciální prvek `<fieldset>`, který je určený pro seskupování prvků formuláře.

```
  <input type="checkbox" name="subscribe" />
  <span class="checkmark">
    <span class="checkmark-inner"></span>
  </span>
  <span class="label-text">Přihlásit se k odběru novinek</span>
</label>
```

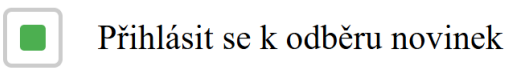
A nyní CSS pro stylování:

CSS pro vlastní zaškrtávací políčko

```css
.custom-checkbox {
  display: flex;
  align-items: center;
  cursor: pointer;
  gap: 1em;
}
.custom-checkbox input {
  display: none;
}
.custom-checkbox .checkmark {
  width: 1.5em;
  height: 1.5em;
  border: 0.125em solid #ccc;
  border-radius: 0.25em;

  display: flex;
  align-items: center;
  justify-content: center;
}
.custom-checkbox .checkmark-inner {
  width: 0.75em;
  height: 0.75em;
  background-color: transparent;
  transition: background-color 0.2s;
  border-radius: 0.125em;
}
.custom-checkbox:has(input:checked) .checkmark-inner {
  background-color: #4CAF50;
}
```

Pro výběr toho stavu, jestli je prvek zaškrtnutý, jsme použili novou pseudo-třídu `:has()`, která nám umožňuje vybírat rodičovské prvky na základě jejich potomků. Takže v našem případě vybíráme `.custom-checkbox`, pokud obsahuje `input:checked`.



Obrázek 20: Vlastní zaškrtávací políčko

6.2 JavaScript

V první řadě, než se do JavaScriptu pustíme, je nutné si uvědomit, že jsme v předmětu zaměřeném na webové technologie. A na JavaScript máme pouze omezený čas. Proto se zaměříme pouze na to nejdůležitější, co potřebujeme pro tvorbu interaktivních webových stránek. Místy si připomenem i základy programování, ale v části již nebudeme řešit obecné koncepty programování, jako jsou datové struktury, algoritmy apod. Pro tyto koncepty doporučuji samostudium nebo jiné předměty zaměřené na programování.

Spousta věcí je zde zjednodušena a pokud chcete komplexnější přehled o JavaScriptu, doporučuji se podívat na skripta z třetího ročníku MVOP Webového inženýrství.

JavaScript je divný. To je jediná věta, kterou v celých skriptech uslyšíte jako výtku vůči tomuto jazyku. Zvykněte si na to hned teď, po následujících kapitolách by jste si měli uvědomit proč tomu tak je. Stěžovat si na jeden z nejpoužívanějších jazyků na světě je ale zbytečné.

JavaScript je velmi silný jazyk, který lze používat skoro na cokoliv. V rámci následujících kapitol se zaměříme na použití JavaScriptu pro webové stránky, ale je dobré vědět, že JavaScript lze používat i mimo webové prohlížeče.

6.2.1 Historie jazyka

JavaScript byl vytvořen v roce 1995 Brendanem Eichem během pouhých deseti dnů, zatímco pracoval ve společnosti Netscape. Původně byl navržen jako jednoduchý skriptovací jazyk pro webové stránky, ale rychle se stal jedním z nejpopulárnějších programovacích jazyků na světě. V roce 1997 byl JavaScript standardizován organizací ECMA International pod názvem ECMAScript.

Nyní tedy nepoužíváte JavaScript, ale ECMAScript, i když se tento termín běžně nepoužívá. Postupem let se JavaScript vyvíjel a přidávaly se nové funkce a vlastnosti.

Kvůli zpětné kompatibilitě zůstaly některé starší a méně efektivní způsoby psaní kódu, což přispívá k „divnosti“ jazyka. Tyto starší koncepty, které standard vyřadil nebo radí proti jejich používání, nebudeme v našich skriptech používat. Vyhnutím se jim se vyhneme i mnoha problémům, které by mohly nastat.

6.2.2 Klíčové koncepty

Nyní se vrhneme již na to, jak se s JavaScriptem pracuje.

Prvné zodpovíme otázku, jak vůbec JS vložíme do webové stránky. Proto máme definován prvek `<script>`, který nám umožňuje vkládat JavaScriptový kód přímo do HTML dokumentu nebo načítat externí JavaScriptové soubory.

`<script>`: JavaScriptový kód

Definuje blok JavaScriptového kódu nebo odkaz na externí JavaScriptový soubor.

Atributy:

src Určuje URL externího JavaScriptového souboru, který má být načten a vykonán.

type Určuje typ skriptu. Výchozí hodnota je `text/javascript`, což označuje JavaScriptový kód.

module Pokud je tento atribut přítomen, skript je považován za JavaScriptový modul, což umožňuje použití importů a exportů mezi různými soubory. V rámci WBA se s moduly nebudeme zabývat.

```html
<script>
  console.log("Ahoj, světe!");
</script>
```

Tag s JS lze vkládat buď do hlavičky dokumentu (`<head>`), nebo do těla dokumentu (`<body>`). Tyto dva způsoby mají velké důsledky na to, kdy a s čím se začne JavaScriptový kód vykonávat. Pokud je skript umístěn v hlavičce, vykoná se dříve, než je načten zbytek obsahu stránky. To znamená, že kdybychom chtěli manipulovat s prvky na stránce, tak to nebude možné, protože ještě nebyly načteny. Proto je doporučeno umisťovat skripty na konec těla dokumentu, těsně před zavírací značku `</body>` nebo použít speciální atributy (`defer`) a nebo se správně spoléhat na události načtení stránky.

JS je pro nás skriptovaný interpretovaný a dynamicky typovaný jazyk. To znamená, že kód je vykonáván řádek po řádku a proměnné nemusí mít předem definovaný datový typ.

Jednotlivé řádky kódu jsou ukončeny středníkem (`;`), ale v mnoha případech je možné středníky vynechat – ale kvůli možným problémům je lepší je vždy používat.

Komentáře se píší buď pomocí `//` pro jednořádkové komentáře, nebo pomocí `/* ... */` pro víceřádkové komentáře.

6.2.2.1 Typy

V JavaScriptu máme několik základních datových typů:

- `number`: Číselné hodnoty (celá čísla i desetinná),
- `string`: Textové hodnoty (řetězce znaků),
- `boolean`: Logické hodnoty (`true` nebo `false`),
- `null`: Speciální hodnota představující „žádnou hodnotu“,
- `undefined`: Speciální hodnota představující „hodnotu, která nebyla definována“,
- `object`: Složené datové struktury, které mohou obsahovat více hodnot,
- `symbol`: Unikátní a neměnné hodnoty, často používané jako identifikátory.

Všimneme si, že JavaScript nemá speciální datový typ pro celá čísla a desetinná čísla – vše je reprezentováno jako `number`. Podobně neexistuje typ pro znakové řetězce (char).

Pole, o kterém si povíme později, je speciálním typem objektu.

To, jak se s těmito typy pracuje, si ukážeme v následujících kapitolách.

6.2.2.2 Proměnné

V JavaScriptu můžeme vytvářet proměnné pomocí klíčových slov `var`, `let` a `const`.

- `var`: Starší způsob deklarace proměnných, velmi zmatečné určení rozsahu (existuje před samotným řádkem deklarace),

- `let`: Moderní způsob deklarace proměnných s blokovým rozsahem,
- `const`: Deklarace konstantních proměnných, které nelze znovu přiřadit (také s blokovým rozsahem).

U `const` je důležité si uvědomit, že hodnota samotné proměnné nemůže být změněna (přiřazena nová hodnota), ale pokud je hodnota objekt nebo pole, můžeme měnit jeho obsah (vlastnosti nebo prvky). Protože se jedná o tzv. referenční typy, kde proměnná obsahuje odkaz na místo v paměti, kde je objekt nebo pole uložené.

Pokud se pokusíme `const` proměnnou znovu přiřadit, vyhodí runtime výjimku, která způsobí, že další příkazy se neprovedou.

6.2.2.3 Operátory

S jednotlivými datovými typy můžeme provádět různé operace pomocí operátorů. Mezi základní operátory patří:

- Aritmetické operátory: `+`, `-`, `*`, `/`, `%` (modulo),
- Přiřazovací operátory: `=`, `+=`, `-=`, `*=`, `/=`,
- Porovnávací operátory: `==`, `===`, `!=`, `!==`, `<`, `>`, `<=`, `>=`,
- Logické operátory: `&&` (AND), `||` (OR), `!` (NOT),
- Řetězcové operátory: `+` (konkatenace řetězců)[56].
- Ternární operátor: `condition ? valueIfTrue : valueIfFalse`,
- Binární operátory: `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`,
- Inkrementační a dekrementační operátory: `++`, `--`,
- A další speciální, jako např. `typeof`.

Modulo je v JS operátor typu, který vrací zbytek po dělení dvou čísel. Nejedná se však o zbytek jako v matematice, ale o tzv. „zbytek podle podílu“. To znamená, že výsledek může být záporný, pokud je první operand záporný. Pro „pravé modulo“ (vždy kladný výsledek) je potřeba použít speciální funkci:

Funkce pro pravé modulo v JavaScriptu

```js
function mod(a, b) {
  return ((a % b) + b) % b;
}
```

Mimo klasické programování vidíme, že existují dva typy porovnání, a to:

- Volné porovnání (`==`, `!=`): Při porovnání dochází k automatické konverzi typů, což může vést k neočekávaným výsledkům, např. `0 == '0'` je `true`,
- Striktní porovnání (`===`, `!==`): Porovnávají se jak hodnoty, tak i typy bez konverze, např. `0 === '0'` je `false`.

Chceme skoro vždy používat striktní porovnání, abychom se vyhnuli nejasnostem.

Další operátory, které jste možná v rámci předmětu Programování a vývoj aplikací neviděli, jsou binární operátory. Tyto operátory pracují na úrovni jednotlivých bitů čísel a umožňují provádět bitové operace, jako je AND, OR, XOR, NOT a posuny bitů. Tyto operace jsou užitečné pro nízkoúrovňové programování, optimalizace výkonu a práci s daty na úrovni bitů.

Jednotlivé operátory:

[56] Všimněte si, že v JS má operátor + dvě funkce – s čísly provádí sčítání, s řetězci konkatenaci. Pokud je jeden operand řetězec, druhý se převede na řetězec a provede se konkatenace.

- & (bitový AND): Provádí logickou operaci AND na každém bitu dvou čísel,
- | (bitový OR): Provádí logickou operaci OR na každém bitu dvou čísel,
- ^ (bitový XOR): Provádí logickou operaci XOR na každém bitu dvou čísel, tedy, že právě jedno (a jen jedno) z bitů je 1, tak výsledek je 1,
- ~ (bitový NOT): Inverzuje všechny bity čísla,
- << (levý posun): Posune bity čísla doleva o zadaný počet pozic,
- >> (pravý posun se znaménkem): Posune bity čísla doprava o zadaný počet pozic, zachovává znaménko.

Připomeneme si ještě význam inkrementace a dekrementace:

- ++: Zvýší hodnotu proměnné o 1,
- --: Sníží hodnotu proměnné o 1.

Důležité ale je, kde je tento operátor umístěn:

- Prefixová forma (`++variable` nebo `--variable`): Nejprve se hodnota proměnné změní, a pak se použije v daném výrazu,
- Postfixová forma (`variable++` nebo `variable--`): Nejprve se hodnota proměnné použije v daném výrazu, a pak se změní.

Příklad prefixové a postfixové inkrementace v JavaScriptu

```js
let a = 5;
console.log(++a); // 6
console.log(a++); // 6
console.log(a);   // 7
```

Ternární operátor slouží jako zkrácená forma podmíněného výrazu. Jeho syntaxe je následující: `condition ? valueIfTrue : valueIfFalse` Kde `condition` je výraz, který se vyhodnotí jako pravda (`true`) nebo nepravda (`false`). Pokud je `condition` pravda, vrátí se `valueIfTrue`, jinak se vrátí `valueIfFalse`.

Posledním k připomenutí jsou operátory přiřazení. Společně s klasickým přiřazením (=) máme i zkrácené formy, které kombinují přiřazení s jinou operací. Například:

- +=: Přičte hodnotu k proměnné a přiřadí výsledek zpět do proměnné,
- -=: Odečte hodnotu od proměnné a přiřadí výsledek zpět do proměnné,
- A tak dále pro ostatní aritmetické operátory.

V podstatě si to lze představit jako zkratku:

$$x\ op = y$$

Je ekvivalentní k:

$$x = x\ op\ y$$

Kde `op` je libovolný aritmetický operátor.

6.2.2.4 Řídicí struktury

Mezi řídicí struktury v JavaScriptu patří:

- Podmíněné příkazy: `if`, `else if`, `else`, `switch`,
- Smyčky: `for`, `while`, `do...while`,
- Výjimky: `try`, `catch`, `finally`, `throw`.

Větvení pomocí podmínek způsobí, že se určitý blok kódu vykoná pouze tehdy, pokud je splněna daná podmínka. Část podmínky která se porovnává se jmenuje „výraz“ a může

obsahovat jakékoliv platné výrazy v JavaScriptu, které vrací hodnotu `true` nebo `false`[57]. Po jednoduché podmínce může následovat libovolný počet `else if` větví, které umožňují testovat další podmínky, a nakonec může být přítomen jeden `else` blok, který se vykoná, pokud žádná z předchozích podmínek nebyla splněna. Výrazy podmínek se kontrolují postupně podle toho, jak byly definovány.

Příklad větvení pomocí if-else v JavaScriptu

```js
let score = 85;
if (score >= 90) {
  console.log("Výborně");
} else if (score >= 75) {
  console.log("Dobře");
} else if (score >= 50) {
  console.log("Uspokojivě");
} else {
  console.log("Neuspokojivě");
}
```

Speciální formou větvení je `switch`, který umožňuje porovnávat hodnotu výrazu s několika možnými případy (`case`). Pokud hodnota výrazu odpovídá hodnotě některého z případů, vykoná se odpovídající blok kódu. Každý `case` by měl být ukončen příkazem `break`, aby se zabránilo „propadnutí“ do dalších případů. Někdy se ale toto „propadnutí“ využívá záměrně pro sdílení kódu mezi více případy. Jednotlivé bloky `case` se mohou zaobalit do složených závorek `{}`. To děláme zejména, pokud v daném bloku definujeme proměnné, aby nedošlo k jejich kolizi s jinými bloky.

Taktéž je možné definovat `default` blok, který se vykoná, pokud žádný z případů neodpovídá hodnotě výrazu.

Příklad větvení pomocí switch v JavaScriptu

```js
let day = 3;
switch (day) {
  case 1: {
    console.log("Pondělí");
    break;
  }
  case 2: {
    console.log("Úterý");
    break;
  }
  case 3: {
    console.log("Středa");
    break;
  }
  default: {
    console.log("Neznámý den");
```

[57] Ve skutečnosti JavaScript používá koncept „truthy“ a „falsy“ hodnot, což znamená, že některé hodnoty jsou považovány za pravdivé nebo nepravdivé i bez explicitního porovnání s `true` nebo `false`. Například číslo `0`, prázdný řetězec `""`, `null`, `undefined` a `NaN` jsou považovány za „falsy“, zatímco všechny ostatní hodnoty jsou „truthy“.

```
17   }
18 }
```

V JS máme dva základní typy smyček: `for` a `while`.

Smyčka `for` se používá, když víme předem, kolikrát chceme opakovat určitý blok kódu. Skládá se ze tří částí: inicializace, podmínka a „úprava počítadla“. Inicializace se provede jednou na začátku smyčky, podmínka se kontroluje před každou iterací, a „úprava počítadla“ se provede na konci každé iterace.

js

Příklad smyčky for v JavaScriptu

```
1 for (let i = 0; i < 5; i++) {
2   console.log("Iterace číslo: " + i);
3 }
```

Ani jedna z částí smyčky `for` není povinná. Můžeme tedy vytvořit nekonečnou smyčku, pokud vynecháme podmínku:

js

Příklad nekonečné smyčky for v JavaScriptu

```
1 for (;;) {
2   console.log("Toto poběží navždy");
3 }
```

Taktéž je možné definovat více inicializací a „úprav počítadla“ pomocí čárek:

js

Příklad smyčky for s více proměnnými v JavaScriptu

```
1 for (let i = 0, j = 10; i < j; i++, j--) {
2   console.log("i: " + i + ", j: " + j);
3 }
```

Smyčka `while` se používá, když nevíme předem, kolikrát chceme opakovat určitý blok kódu. Skládá se pouze z podmínky, která se kontroluje před každou iterací. Pokud je podmínka pravdivá, vykoná se blok kódu uvnitř smyčky.

js

Příklad smyčky while v JavaScriptu

```
1 let count = 0;
2 while (count < 5) {
3   console.log("Počet: " + count);
4   count++;
5 }
```

Podobně existuje i smyčka `do...while`, která se liší od `while` tím, že blok kódu se vykoná alespoň jednou před kontrolou podmínky.

js

Příklad smyčky do...while v JavaScriptu

```
1 let count = 0;
2 do {
3   console.log("Počet: " + count);
```

```
4   count++;
5 } while (count < 5);
```

Pro iteraci existují dva speciální příkazy, které nám dovolují průběh smyčky ovlivnit:

- `break`: Okamžitě ukončí smyčku a pokračuje vykonávání kódu za smyčkou,
- `continue`: Přeskočí zbytek aktuální iterace a pokračuje další iterací, pokud je podmínka smyčky stále pravdivá.

Příklad použití break a continue v JavaScriptu

```js
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Ukončí smyčku, když i je 5
  }
  if (i % 2 === 0) {
    continue; // Přeskočí sudá čísla
  }
  console.log("Neparní číslo: " + i);
}
```

Pro průchod objekty a poli máme speciální smyčky, ale ty si ukážeme později, viz Kapitola 6.2.2.7.

6.2.2.5 Funkce

Funkce jsou bloky kódu, které lze opakovaně volat. Funkce se deklarují pomocí klíčového slova `function`, následovaného názvem funkce, seznamem parametrů v závorkách a tělem funkce v složených závorkách.

Deklarace funkce v JavaScriptu

```js
function greet(name) {
  console.log("Ahoj, " + name + "!");
}
```

Funkce se volají pomocí jejich názvu následovaného závorkami, ve kterých můžeme předat argumenty.

Volání funkce v JavaScriptu

```js
greet("Svět"); // Volání funkce s argumentem "Svět"
```

Funkce mohou vracet hodnoty pomocí klíčového slova `return`. Když je `return` vykonáno, funkce okamžitě skončí a vrátí zadanou hodnotu.

Funkce vracející hodnotu v JavaScriptu

```js
function add(a, b) {
  return a + b;
}
let sum = add(5, 3); // sum bude 8
```

Pokud funkce nedosáhne `return` příkazu, vrátí automaticky hodnotu `undefined`.

Mimo to existují další způsoby jak deklarovat funkce, například pomocí funkčních výrazů nebo šipkových funkcí (arrow functions), které jsou kratší a mají odlišné chování vůči `this`, ty jsou ale nad rámec těchto skript.

6.2.2.6 Objekty

Objekt je kolekce vlastností, kde každá vlastnost je dvojice klíč-hodnota. Objekty se vytvářejí pomocí složených závorek `{}` a vlastnosti se definují jako klíč následovaný dvojtečkou a hodnotou. Klíč musí být buď platný identifikátor, nebo řetězec v uvozovkách. Vlastnosti jsou odděleny čárkami.

Vytvoření objektu v JavaScriptu

```js
let person = {
  name: "Jan",
  age: 30,
  isStudent: false
};
```

K danému objektu, resp. jeho vlastnostem, můžeme přistupovat pomocí tečkové notace nebo hranatých závorek.

Přístup k vlastnostem objektu v JavaScriptu

```js
console.log(person.name); // Přístup pomocí tečkové notace
console.log(person["age"]); // Přístup pomocí hranatých závorek
```

Hranaté závorky jsou užitečné, když klíč obsahuje speciální znaky nebo mezery, nebo když klíč není znám předem (například je uložen v proměnné). Můžeme také přidávat, měnit nebo odstraňovat vlastnosti objektu.

Úprava vlastností objektu v JavaScriptu

```js
person.isEmployed = true; // Přidání nové vlastnosti
person.age = 31; // Změna hodnoty vlastnosti
```

Pod jednotlivými hodnotami lze ukládat jakékoliv další typy, včetně funkcí a dalších objektů.

Objekt s metodou v JavaScriptu

```js
let student = {
  name: "Eva",
  grades: [90, 85, 88],
  greet: function() {
    console.log("Ahoj, jmenuji se " + this.name);
  }
};
student.greet(); // Volání metody objektu
```

Objekty lze procházet pomocí smyčky `for...in`, která iteruje přes všechny vlastnosti objektu.

Procházení objektu pomocí for...in v JavaScriptu

```
for (let key in person) {
  console.log(key + ": " + person[key]);
}
```

6.2.2.7 Pole

Pole jsou speciální typ objektu, který slouží k ukládání uspořádaných kolekcí hodnot. Pole se vytvářejí pomocí hranatých závorek `[]`, a jednotlivé prvky jsou odděleny čárkami.

js

Vytvoření pole v JavaScriptu

```
let numbers = [1, 2, 3, 4, 5];
```

K prvkům pole můžeme přistupovat pomocí jejich indexu, který začíná na 0.

js

Přístup k prvkům pole v JavaScriptu

```
console.log(numbers[0]); // Přístup k prvnímu prvku (1)
console.log(numbers[2]); // Přístup k třetímu prvku (3)
```

Vzhledem k tomu, že jsou pole objekty, mají i své vlastnosti a metody. Máme k dispozici například:

- `length`: Vlastnost, která vrací počet prvků v poli,
- `push()`: Metoda pro přidání jednoho nebo více prvků na konec pole,
- `pop()`: Metoda pro odstranění posledního prvku z pole,
- `shift()`: Metoda pro odstranění prvního prvku z pole,
- `unshift()`: Metoda pro přidání jednoho nebo více prvků na začátek pole,
- `indexOf()`: Metoda pro nalezení indexu prvního výskytu určité hodnoty v poli,
- A další.

Pole můžeme procházet pomocí smyčky `for`, `for..in` nebo `for..of`. Cyklus typu `for..of` je speciálně navržen pro iteraci přes iterovatelné objekty[58], jako jsou pole, a poskytuje přímý přístup k hodnotám prvků bez nutnosti používat indexy.

js

Procházení pole v JavaScriptu

```
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
for (let index in numbers) {
  console.log(numbers[index]);
}
for (let value of numbers) {
  console.log(value);
}
```

[58] Na obyčejném objektu `for..of` nefunguje, protože není iterovatelný.

> 🤝 **Použití .forEeach()**
>
> Studenti často k procházení polí používají metodu `.forEach()`, která přijímá funkci jako argument a volá ji pro každý prvek pole.
>
> Tato funkce je užitečná, přesto ji nedoporučuji, protože:
> - Může být méně efektivní než klasické smyčky,
> - Může být méně čitelná pro začátečníky,
> - Neumožňuje použití `break` nebo `continue` pro řízení toku smyčky.

6.2.2.8 Práce s typy a standardní knihovnou

V rámci JavaScriptu máme k dispozici několik vestavěných objektů a funkcí, které nám umožňují pracovat s různými datovými typy a provádět běžné operace. Nyní si představíme ty nejdůležitější z nich.

Konzole:

V globálním prostoru máme k dispozici objekt `console`, který nám umožňuje vypisovat informace do konzole prohlížeče.

Mezi nejčastěji používané metody patří:
- `console.log()`: Vypíše zprávu do konzole,
- `console.group()`: Začne seskupení zpráv v konzoli,
- `console.groupEnd()`: Ukončí seskupení zpráv v konzoli,
- `console.time()`: Spustí časovač s daným názvem,
- `console.timeEnd()`: Zastaví časovač a vypíše uplynulý čas do konzole,
- A další.

Matematické operace:

Pro práci s čísly máme k dispozici objekt `Math`, který poskytuje různé matematické funkce a konstanty.

Mezí nejčastěji používané metody a vlastnosti patří:
- `Math.PI`: Konstanta π (pí),
- `Math.sqrt()`: Vypočítá druhou odmocninu čísla,
- `Math.pow()`: Vypočítá mocninu čísla,
- `Math.random()`: Vrátí náhodné číslo mezi 0 (včetně) a 1 (vyloučeno),
- `Math.floor()`: Zaokrouhlí číslo dolů na nejbližší celé číslo,
- `Math.ceil()`: Zaokrouhlí číslo nahoru na nejbližší celé číslo,
- `Math.round()`: Zaokrouhlí číslo na nejbližší celé číslo,
- A další.

Pro čísla máme k dispozici ještě globální funkce pro konverzi:
- `Number()`: Převádí hodnotu na číslo nehledě na její původní typ, neobsahuje žádné speciální chování,
- `parseInt()`: Převádí řetězec na celé číslo, umožňuje specifikovat základ číselné soustavy,
- `parseFloat()`: Převádí řetězec na desetinné číslo.

Pro nejlepší praxi doporučuji používat `Number()`, protože je nejpřehlednější a nejméně náchylná k chybám.

Pro validaci čísel máme k dispozici globální funkce:

- `Number.isNaN()`: Zjistí, zda je hodnota `NaN` (Not-a-Number),
- `Number.isFinite()`: Zjistí, zda je hodnota konečné číslo.

Na konkrétním čísle poté můžeme zavolat různé metody, například:
- `toFixed()`: Vrátí řetězec reprezentující číslo zaokrouhlené na zadaný počet desetinných míst,
- `toString()`: Vrátí řetězec reprezentující číslo v zadané číselné soustavě (základ).

Řetězce:

Na konkrétních řetězcích máme k dispozici různé metody pro práci s textem.

Mezi nejčastěji používané metody patří:
- `length`: Vlastnost, která vrací délku řetězce,
- `toUpperCase()`: Vrátí řetězec převedený na velká písmena,
- `toLowerCase()`: Vrátí řetězec převedený na malá písmena,
- `indexOf()`: Vrátí index prvního výskytu podřetězce v řetězci,
- `substring()`: Vrátí podřetězec mezi dvěma indexy,
- `replace()`: Nahradí část řetězce jiným řetězcem,
- `replaceAll()`: Nahradí všechny výskyty části řetězce jiným řetězcem,
- `split()`: Rozdělí řetězec na pole podřetězců podle zadaného oddělovače,
- `trim()`: Odstraní bílé[59] znaky z začátku a konce řetězce,
- `includes()`: Zjistí, zda řetězec obsahuje zadaný podřetězec,
- `startsWith()`: Zjistí, zda řetězec začíná zadaným podřetězcem,
- `endsWith()`: Zjistí, zda řetězec končí zadaným podřetězcem,
- A další.

Objekty:

Pro práci s objekty máme k dispozici několik užitečných metod na objektu `Object`.

Mezi nejčastěji používané metody patří:
- `Object.keys()`: Vrátí pole obsahující všechny klíče vlastností objektu,
- `Object.values()`: Vrátí pole obsahující všechny hodnoty vlastností objektu,
- `Object.entries()`: Vrátí pole obsahující všechny dvojice klíč-hodnota vlastností objektu jako podpole,
- A další.

6.3 Document Object Model

JavaScriptu již nyní tedy trochu rozumíme, abychom ho mohli používat pro to, k čemu byl navržen – tedy pro webové stránky, musíme si představit, jakým způsobem JavaScript komunikuje s webovou stránkou.

První věc, kterou si musíme osvětlit, je to, že název předmětu Webové aplikace je trochu zavádějící. Do teď (do tohoto momentu!) jsme se zabývali pouze statickými webovými **stránkami**, které jsou tvořeny HTML a CSS. Webové aplikace jsou ale něco jiného – jedná se o něco komplexnějšího (až aplikace), které dovolují interakci s uživatelem, dynamický obsah a další funkce.

K tomu všemu nám slouží JavaScript a právě Document Object Model (DOM).

[59] Mezi bílé znaky patří mezery, tabulátory, nové řádky a další neviditelné znaky na začátku a konci řetězce.

DOM je programové rozhraní (API), které skriptu dává k dispozici prohlížeč pro manipulaci s obsahem, strukturou a stylem webové stránky. Každý prvek, který je vytvořen na stránce (definován v HTML) je reprezentován jako objekt v DOM stromu. Každý prvek může mít své děti (podřízené prvky), atributy a obsah.

6.3.1 Výběry prvků

Nejdůležitější je, jakým způsobem vůbec ve stromě prvků najdeme ty, se kterými chceme pracovat.

V první řadě máme od prohlížeče v JS globálně definované objekty `document` a `window`. Objekt `window` reprezentuje samotné okno prohlížeče a poskytuje přístup k různým funkcím a vlastnostem okna – například změnit historii prohlížení, přesměrovat na jinou stránku, pracovat s místním úložištěm (local storage) a další. Objekt `document` reprezentuje samotný HTML dokument načtený v okně prohlížeče a poskytuje přístup k jeho obsahu a struktuře.

Tedy už uvnitř objektu `document` můžeme začít vybírat jednotlivé prvky na stránce. Uvnitř něho najdeme prvek „těla“ – `document.body`, který reprezentuje obsah stránky.

Uvnitř něho bychom mohli začít hledat (např. rekurzí) jednotlivé prvky, ale to by bylo velmi nepraktické. Proto máme k dispozici několik metod pro výběr prvků podle různých kritérií:

- `getElementById(id)`: Vrátí prvek s daným ID[60],
- `getElementsByClassName(className)`: Vrátí kolekci prvků, které mají tuto třídu (a mohou mít i jiné třídy),
- `getElementsByTagName(tagName)`: Vrátí kolekci prvků s daným názvem značky (tagu).

Tyto uvedené metody jsou ale velmi pracné, protože často potřebujeme například vybrat všechny nadpisy ze všech příspěvků na stránce, což bychom museli dělat například následujícím způsobem.

Výběr nadpisů z příspěvků pomocí getElementsByTagName v JavaScriptu

```js
const headings = [];
const articles = document.getElementsByTagName("article");

for (let i = 0; i < articles.length; i++) {
  if (!articles[i].classList.contains("post")) {
    continue;
  }

  const headings = articles[i].getElementsByTagName("h2");

  if (headings.length > 1) {
    console.error("Více než jeden nadpis v příspěvku?!");
    continue;
  }

  const heading = headings[0];
  headings.push(heading);
}
```

[60] Jedno ID na stránce = vrátí jeden prvek!

To nechceme dělat ručně, a to tohle byl ještě relativně jednoduchý příklad.

V ukázce si taktéž všimneme, že pro průchod vrácené kolekce musíme použít klasickou smyčku `for`, protože se nejedná o pole, ale o tzv. `HTMLCollection`, která nemá všechny metody pole. V případě, že bychom chtěli používat metody pro pole, nebo např. použít cyklus `for..of`, museli bychom kolekci převést na pole pomocí `Array.from()` nebo rozšiřujícího operátoru (`...`). Například:

Převod HTMLCollection na pole pomocí rozšiřujícího operátoru v JavaScriptu

```js
for (let article of [...articles]) {}
```

Proto máme k dispozici modernější a flexibilnější metody, které využívají CSS selektory pro výběr prvků, které již skvěle umíme z předchozích kapitol. Tyto metody jsou:

- `querySelector(selector)`: Vrátí první prvek, který odpovídá zadanému CSS selektoru,
- `querySelectorAll(selector)`: Vrátí všechny prvky, které odpovídají zadanému CSS selektoru jako statickou NodeList kolekci.

Například pro stejný případ výběru nadpisů z příspěvků bychom mohli použít následující kód:

Výběr nadpisů z příspěvků pomocí querySelectorAll v JavaScriptu

```js
const headings = [...document.querySelectorAll("article.post h2")];
```

6.3.2 Práce s prvky

Na daném prvku poté máme nespočet vlastností a metod, které nám dovolují s ním pracovat. Mezi nejdůležitější vlastnosti a metody patří:

- `innerHTML`: Vlastnost, která obsahuje HTML kód uvnitř prvku, nastavením přepíšeme celý obsah[61],
- `textContent`: Vlastnost, která obsahuje čistý text uvnitř prvku,
- `classList`: Vlastnost, která poskytuje přístup k seznamu tříd prvku a umožňuje přidávat, odebírat nebo přepínat třídy,
  - `add()`: Přidá třídu k prvku,
  - `remove()`: Odebere třídu z prvku,
  - `toggle()`: Přepne přítomnost třídy na prvku,
  - `contains()`: Zjistí, zda prvek obsahuje danou třídu,
- `innerText`: Vlastnost, která obsahuje viditelný text uvnitř prvku,
- `getAttribute(name)`: Metoda pro získání hodnoty atributu prvku,
- `setAttribute(name, value)`: Metoda pro nastavení hodnoty atributu prvku,
- `appendChild(child)`: Metoda pro přidání podřízeného prvku na konec seznamu dětí prvku,
- `removeChild(child)`: Metoda pro odstranění podřízeného prvku z prvku,
- `style`: Vlastnost, která poskytuje přístup k inline stylům prvku[62],
  - `[property]`: Nastaví nebo získá hodnotu konkrétní CSS vlastnosti (např. `element.style.color = "red";`).

Dle typu prvku máme k dispozici i další specifické vlastnosti a metody. Například pro formulářové prvky (input, textarea, select) máme vlastnost `value`, která obsahuje aktuální hodnotu

[61] Pozor na bezpečnostní rizika spojená s XSS útoky při používání `innerHTML` s neověřeným vstupem!

[62] A tady se přesně inline styly hodí, protože my je měníme v závislosti na kontextu akce v JS, a tedy je přednostně nemůžeme definovat v CSS.

prvku. Pro formulář jako takový máme funkce `submit()` a `reset()`, které odesílají nebo resetují formulář.

Pro vytvoření nového prvku máme k dispozici metodu `createElement(tagName)` na objektu `document`, která vytvoří nový prvek s daným názvem značky (tagu).

Vytvoření nového prvku v JavaScriptu

```js
const newDiv = document.createElement("div");
```

Daný prvek poté můžeme upravovat pomocí věcí zmíněných výše, ale dokud ho nepřidáme do nějakého prvku, který již je součástí DOM stromu, nebude viditelný na stránce. Přidání provedeme pomocí metody `appendChild()` nebo `insertBefore()` na rodičovském prvku.

Přidání nového prvku do DOM stromu v JavaScriptu

```js
const parentElement = document.querySelector("#parent");
parentElement.appendChild(newDiv);
```

6.3.3 Události

Události jsou akce na webové stránce (např. kliknutí myši, stisk klávesy, načtení stránky), na které můžeme reagovat pomocí JavaScriptu. V rámci každého prvku na webové stránce se můžeme zaregistrovat na určité události pomocí metody `addEventListener(eventType, callback)`. Kde `eventType` je řetězec reprezentující typ události (např. `"click"`, `"keydown"`, `"load"`), a `callback` je funkce, která se vykoná, když dojde k dané události. Na nějakých konkrétních prvcích existují speciální události jen pro ně, například pro formulář jsou to události jako `"submit"`, která se spustí před odesláním formuláře.

Jako první parametr funkce nám přijde objekt události, který obsahuje informace o události, jako je typ události, cílový prvek, čas události a další. Na tomto objektu můžeme zavolat taktéž dvě různé metody:

- `preventDefault()`: Zabrání výchozímu chování události (např. zabrání odeslání formuláře),
- `stopPropagation()`: Zastaví šíření události na rodičovské prvky.

Propagace znamená, že když klikneme na tlačítko, tak došlo sice kliknutí na tlačítku, a to se to prvně dozví, ale následně se tato událost „šíří“ na rodičovské prvky, které mohou mít taktéž registrované události pro kliknutí. Poté můžeme zaregistrovat třeba kliknutí na celé obrazovce a to ikdyž si tuto událost zapracovali i jiné (konkrétnější) prvky.

Příklad registrace události kliknutí v JavaScriptu

```js
const button = document.querySelector("button");
button.addEventListener("click", function(event) {
  console.log("Tlačítko bylo kliknuto!");
  event.preventDefault(); // Zabrání výchozímu chování (např. odeslání formuláře)
});
```

V ukázce výše si všimněte, že jsme si definovali anonymní funkci. Je to podobné těm, které jsme si ukazovali, jen, že tady nemá funkce jméno a je předávána jako „hodnota“ do jiné funkce.

Mezi nejznámější události patří:

- `click`: Spustí se při kliknutí na prvek,
- `mouseover`: Spustí se, když myš přejede nad prvek,
- `mouseout`: Spustí se, když myš opustí prvek,
- `keydown`: Spustí se při stisknutí klávesy na klávesnici,
- `keyup`: Spustí se při uvolnění klávesy na klávesnici,
- `load`: Spustí se, když je stránka nebo obrázek plně načten (pouze na `window` nebo `img` prvcích)[63],
- `submit`: Spustí se při odeslání formuláře,
- A mnoho dalších.

> 😔 **Nešvar**
>
> Studenti mají často problém s tim, že se události nastavuji uvnitř skriptu a ne uvnitř HTML. Největší problém je to, že HTML definuje strukturu stránky, zatímco JavaScript by měl být použit pro logiku a interakci.
>
> Studenti si často najdou, že mohou na konkrétní prvek nastavit atribut `onclick`, který přijímá JavaScriptový kód, který se vykoná při kliknutí na prvek. Často se tam dává funkce, takže např. `<button onclick="myFunction()">Klikni mě</button>`.
>
> Otázka je, kde je tahle funkce definována? Nevíme. Pravděpodobně někde v `<script>` bloku, ale kde přesně? Tím pádem ale musí být k dispozici v globálním kontextu, což je automaticky špatně, protože poté funkce stejného jména může kolidovat s jinou funkcí.
>
> Nepoužívejte tento přístup!
>
> Podobně studenti používají místo metody `addEventListener()` přímo přiřazení k vlastnosti události, například `button.onclick = myFunction;`. Tento přístup je sice lepší než použití atributu v HTML, ale sémanticky je to stále nepřehledné - přiřazujeme jednu funkci, i kdybych jich tam mohlo být více.

6.3.4 Zpoždění a časovače

V JavaScriptu máme k dispozici dvě hlavní funkce pro zpoždění vykonání kódu nebo jeho opakování v čase:

- `setTimeout(callback, delay)`: Vykoná funkci `callback` jednou po uplynutí zadaného zpoždění v milisekundách,
- `setInterval(callback, interval)`: Vykoná funkci `callback` opakovaně každých zadaných milisekund.

Příklad použití setTimeout a setInterval v JavaScriptu

```js
setTimeout(function() {
  console.log("Toto se vykoná po 2 sekundách");
}, 2000);
```

[63] Vzpomeňte si na začátek kapitoly o JS, kde jsme si říkali, že JS někdy potřebují čekat, než je stránka plně načtena, aby mohl pracovat s prvky na stránce. Existuje ještě událost `DOMContentLoaded`, která se spustí, když je celý HTML dokument plně načten a zpracován, ale před načtením stylů, obrázků a dalších zdrojů. Tato událost je často používána pro inicializaci skriptů, protože zaručuje, že všechny prvky v DOM jsou dostupné pro manipulaci, ale ne nutně že všechny externí zdroje jsou načteny.

```
4 setInterval(function() {
5   console.log("Toto se vykoná každou sekundu");
6 }, 1000);
```

Je důležité těmto funkcím nevěřit. Z velmi dobrých důvodů není garantované, že se kód vykoná přesně po uplynutí zadaného času. Jediná garance je, že se kód vykoná **nejdříve** po uplynutí zadaného času.

Interval i timeout vrací jedinečný identifikátor, který můžeme použít k zastavení časovače pomocí funkcí:

- `clearTimeout(timeoutId)`: Zastaví timeout s daným identifikátorem,
- `clearInterval(intervalId)`: Zastaví interval s daným identifikátorem.

6.3.5 Animace

Jak jsme se dozvěděli v animacích, viz Kapitola 5.4, animace na webových stránkách se často provádí pomocí CSS přechodů a klíčových snímků. Pokud potřebujeme něco komplexnějšího, můžeme použít JavaScript pro manipulaci s vlastnostmi prvků v čase.

Máme dvě možnosti:

- Ručně měnit vlastnosti prvků – pomocí vlastnosti `style`,
- Použít připravenou metodu `animate()` na prvku, která nám zanimuje konkrétní vlastnosti.

Oboje metody mají své výhody a nevýhody. Ruční změna vlastností pomocí `style` je flexibilní a umožňuje nám plnou kontrolu nad animací, ale vyžaduje více kódu a může být méně efektivní. Metoda `animate()` je jednodušší na použití a optimalizovaná pro výkon, ale může být méně flexibilní pro složité animace.

Začneme s příkladem ruční změny vlastností pomocí `style` a `setInterval()`.

js

Příklad animace pomocí style a setInterval v JavaScriptu

```
1 const box = document.querySelector(".box");
2 let position = 0;
3 const intervalId = setInterval(function() {
4   position += 5;
5   box.style.transform = "translateX(" + position + "px)";
6   if (position >= 300) {
7     clearInterval(intervalId);
8   }
9 }, 100);
```

Všimneme si, že v ukázce používáme interpolaci a to konrétně lineární změnu pozice. Mohli bychom použít i jiné časovací funkce:

js

Příklad časovací funkce easeInOut v JavaScriptu

```
1 function easeInOut(t) {
2   return t < 0.5 ? 2 * t * t : -1 + (4 - 2 * t) * t;
3 }
4
5 const box = document.querySelector(".box");
```

```
let start = 0;
setInterval(function() {
  start += 16; // Přibližně 60 FPS
  let t = Math.min(start / 2000, 1); // Normalizovaný čas od 0 do 1
  let easedT = easeInOut(t);
  box.style.transform = "translateX(" + (easedT * 300) + "px)";
  if (t === 1) {
    clearInterval(this);
  }
}, 16);
```

Všimneme si, že pokud bude z nějakého důvodu zpožděno volání funkce v intervalu (např. kvůli vytížení hlavního vlákna), animace poběží pomaleji, protože používáme pevný časový krok. To lze samozřejmě zlepšit, ale to je nad rámec této kapitoly.

Nyní si ukážeme použití metody `animate()`, která nám dovoluje definovat animaci pomocí klíčových snímků a možností.

Příklad animace pomocí metody animate v JavaScriptu

```js
const box = document.querySelector(".box");
box.animate([
  { transform: 'translateX(0px)' },
  { transform: 'translateX(300px)' }
], {
  duration: 2000,
  iterations: 1,
  easing: 'ease-in-out'
});
```

Jako první parametr přijímá metoda `animate()` pole klíčových snímků, kde každý klíčový snímek je objekt definující vlastnosti a jejich hodnoty v daném bodě animace. Je to podobné tomu, co jsme si ukazovali v CSS klíčových snímcích. Tady ale můžeme v závislosti na potřebách animace přidat jednotlivé snímky, vlastnosti a další věci dynamicky za běhu. Oproti CSS animacím nám to tedy dává větší flexibilitu ale nemáme třeba možnost určit kdy daný snímek „začne“ – všechny jsou rovnoměrně rozloženy v čase.

Jako druhý parametr přijímá metoda `animate()` objekt možností, kde můžeme určit různé vlastnosti animace:

- `duration`: Doba trvání animace v milisekundách,
- `iterations`: Počet opakování animace (může být i `Infinity` pro nekonečné opakování),
- `easing`: Časovací funkce pro animaci (např. `'linear'`, `'ease-in-out'`), podobně jako v CSS,
- A další.

Animace vytvořená pomocí metody `animate()` vrací objekt `Animation`, který nám umožňuje kontrolovat animaci (např. pozastavit, obnovit, zastavit). Pro animaci se můžeme zachytit na událost `finish`, která se spustí, když animace skončí.

Bonusové materiály

7.1 Úvod

Následující kapitoly obsahují různé bonusové materiály, které doplňují hlavní obsah skript. Tato látka se neprobírá na žádných hodinách Webových aplikací a slouží pouze jako doplněk pro zájemce o hlubší pochopení některých témat, resp. dalších informací.

Tato látka často není obsažena ani v následujících předmětech, které se zaobírají webovými aplikacemi – tedy MVOP Webového inženýrství a podobných kurzech.

Některé části zde chybí a to proto, že jsou obsaženy právě v následujících předmětech. Proto se doporučuji se podívat i do materiálů těchto předmětů.

Všechny zmíněné věci nepokrývají (ani s následujícími předměty) kompletně všechny možnosti, které webové technologie nabízejí. Webové aplikace se pořád vyvíjejí.

7.2 Další HTML tagy

<figure>: Obsah s popiskem

Slouží k vložení obsahu na webovou stránku spolu s popiskem. Tento obsah má vztah k obsahu stránky – je součástí významu stránky. Používá se zejména tehdy, když chceme k obsahu přidat popisek, který vysvětluje jeho význam.

Obsahem může být obrázek, ilustrace, diagram, kódový výpis nebo jiný obsah.

Je to párový tag, který obsahuje další tagy, nejčastěji obsahový tag a `<figcaption>`.

html
```
1 <figure>
2   <img src="obrazek.jpg" alt="Popis obrázku">
3 </figure>
```

<figcaption>: Popisek obsahu

Slouží k vložení popisku k obsahu, který je obsažen v tagu `<figure>`. Popisek vysvětluje význam obsahu a jeho vztah k ostatnímu obsahu na stránce.

Je to párový tag, který musí být obsažen uvnitř tagu `<figure>`. Může být umístěn buď před nebo za obsahový tag uvnitř `<figure>`.

html
```
1 <figure>
2   <img src="cat.jpg" alt="Popis obrázku">
3   <figcaption>Kočka sedící na okně.</figcaption>
4 </figure>
```

`<iframe>`: Vložení obsahu z jiné stránky

Slouží k vložení obsahu z jiné webové stránky do aktuální stránky pomocí rámce (frame). Tento tag vytváří samostatný kontext pro obsah, který je načten z jiné URL adresy.

Používá se zejména tehdy, když chceme zobrazit obsah z jiné domény nebo když chceme izolovat určitý obsah od zbytku stránky.

Mezi běžné použití patří vkládání videí, map, formulářů nebo jiných interaktivních prvků z externích zdrojů. Například vložení YouTube videa na stránku.

Je velmi zabězpečený a samotné stránky mohou dovolit vkládání zakázat[64]. Stránky mezi sebou nemohou komunikovat nějak obyčejně, je nutné aby se rodičovská stránka a vložená stránka nějak domluvily pomocí `postMessage` API.

Atributy:

src Určuje URL adresu stránky, která se má načíst do rámce. Může být absolutní nebo relativní URL.

```html
<iframe src="https://www.example.com" width="600" height="400"
  title="Popis obsahu iframe"></iframe>
```

`<details>`: Rozbalitelný obsah

Slouží k vytvoření rozbalitelného a sbalitelného obsahu na webové stránce. Tento tag umožňuje uživatelům zobrazit nebo skrýt dodatečný obsah podle potřeby.

Obsah uvnitř tagu `<details>` je ve výchozím stavu skrytý a zobrazí se až po kliknutí na nadpis, který je definován pomocí tagu `<summary>`.

Používá se zejména tehdy, když chceme poskytnout uživatelům možnost zobrazit více informací bez nutnosti načítat novou stránku nebo přetěžovat stránku přílišným množstvím obsahu. Sémanticky se taktéž jedná o vhodný způsob oproti např. skrývání obsahu pomocí JavaScriptu - po načtení stránky totiž nejspíše nebude přítomen přímo v HTML (DOMu).

```html
<details>
  <summary>Klikněte zde pro více informací</summary>
  <p>Toto je dodatečný obsah, který se zobrazí po rozbalení.</p>
</details>
```

`<summary>`: Sumarizace rozbalitelného obsahu

Slouží k definování nadpisu nebo shrnutí pro obsah uvnitř tagu `<details>`. Tento tag je interaktivní a umožňuje uživatelům kliknout na něj, aby zobrazili nebo skryli dodatečný obsah.

[64] Pro pokročilé, podívejte se na atributy `sandbox` a `CSP` (Content Security Policy).

Je to povinný tag uvnitř `<details>`, protože bez něj by uživatelé neměli žádný způsob, jak rozbalit nebo sbalit obsah.

Používá se zejména tehdy, když chceme poskytnout uživatelům jasný a srozumitelný způsob, jak interagovat s rozbalitelným obsahem na stránce.

`<summary>` je poměrně nový tag a většina prohlížečů ho interně bere jako tlačítko. Uvnitř tlačítka se ale nemůže nacházet jiné prvky, než text nebo inline prvky. Proto je vhodné se zdržet použití jiných tagů uvnitř tohoto tagu.

```html
<details>
  <summary>Klikněte zde pro více informací</summary>
  <p>Toto je dodatečný obsah, který se zobrazí po rozbalení.</p>
</details>
```

`<dialog>`: Dialog

Slouží k vytvoření modálního dialogového okna na webové stránce. Tento tag umožňuje zobrazit obsah, který vyžaduje uživatelskou interakci, například potvrzení akce nebo zadání informací.

Obsah uvnitř tagu `<dialog>` je ve výchozím stavu skrytý a zobrazí se až po jeho otevření pomocí JavaScriptu nebo formuláře.

Element taktéž používá pro stylování speciální pseudo-element `::backdrop`, který umožňuje vytvořit efekt pozadí za dialogem, například ztmavení zbytku stránky.

Používá se zejména tehdy, když chceme poskytnout uživatelům možnost interakce s určitým obsahem bez nutnosti načítat novou stránku nebo přerušovat jejich aktuální činnost na stránce.

Atributy:

open Určuje, zda je dialogové okno otevřené nebo zavřené. Pokud je atribut přítomen, dialog je otevřený; pokud chybí, dialog je zavřený.

```html
<dialog>
  <p>Toto je dialogové okno.</p>

  <form method="dialog">
    <button value="close">Zavřít</button>
  </form>
</dialog>
```

`<em>`: Zdůrazněný text

Slouží k označení textu, který má být zdůrazněn nebo kladen na něj zvláštní důraz. Tento tag obvykle mění vzhled textu, například kurzívou, aby bylo jasné, že se jedná o důležitý nebo zvýrazněný obsah.

Používá se zejména tehdy, když chceme upozornit čtenáře na určitá slova nebo fráze v textu, které mají speciální význam nebo důležitost v kontextu věty nebo odstavce.

Sémanticky to znamená, že čtenář by měl věnovat tomuto textu větší pozornost. Oproti například tagu `<i>`, sloužím `<em>` k vyjádření nalehavosti či důrazu, oproti `<i>`, který slouží k výjimce z normálního textu (např. cizí slova, technické termíny apod.).

html

```
1 <p><em>Teď</em> je ten správný čas na akci.</p>
```

<data>: Strojově čitelná hodnota

Slouží k propojení obsahu s jeho strojově čitelnou reprezentací. Zatímco uživatel vidí formátovaný text, skripty a roboti mohou snadno přečíst hodnotu v atributu `value`.

Používá se například pro ID produktů, překlady měrných jednotek nebo jiné údaje, které mají specifickou hodnotu pro zpracování, ale pro uživatele se zobrazují jinak.

Atributy:

value Určuje strojově čitelnou hodnotu spojenou s obsahem tagu `<data>`.

html

```
1 <ul>
2   <li><data value="21053">Rajčata</data></li>
3   <li><data value="21054">Okurky</data></li>
4   <li><data value="21055">Papriky</data></li>
5 </ul>
```

<time>: Datum a čas

Slouží k označení konkrétního času nebo data. Umožňuje prohlížečům a vyhledávačům lépe porozumět časovým údajům na stránce (např. pro přidání události do kalendáře).

Atributy:

datetime Určuje strojově čitelný formát data a času podle standardu ISO 8601.

html

```
1 <p>Koncert začíná <time datetime="2023-11-18T20:00">18. listopadu v
  osm večer</time>.</p>
```

<cite>: Citace díla

Slouží k označení názvu tvůrčího díla (např. knihy, filmu, písně, obrazu, divadelní hry). Tento tag by měl obsahovat název díla, nikoliv jméno autora.

Je možné vložený obsah obalit například do odkazu na dané dílo.

html

```
1 <p>Mým oblíbeným románem je <cite>Hobit</cite> od J. R. R.
  Tolkiena.</p>
```

<blockquote>: Bloková citace

Slouží k označení sekce, která je citována z jiného zdroje. Zdroj je možné uvést pomocí atributu `cite`.

Název autora, př. díla (v tagu `<cite>`) nebo odkaz se uvádí mimo tag `<blockquote>`, například v odstavci pod citací.

Atributy:

cite Určuje zdroj citace, obvykle URL adresu dokumentu nebo stránky, odkud byla citace převzata.

html

```
1 <blockquote cite="https://matej.cajthaml.eu/citaty">
2   <p>Porovnávat se s h***** není moudré.
3 </blockquote>
4 <p>—Matěj Cajthaml, <cite>Záznam z hodiny Webových aplikací</cite></
  p>
```

<q>: Vložená citace

Slouží k označení krátké citace uvnitř odstavce nebo věty. Na rozdíl od `<blockquote>`, který je pro bloky textu, je `<q>` určen pro inline (řádkové) citace.

html

```
1 <p>Jak řekl klasik: <q>Vím, že nic nevím.</q></p>
```

<address>: Kontaktní informace

Slouží k uvedení kontaktních informací autora článku nebo vlastníka dokumentu. Často se umisťuje do zápatí (`<footer>`) článku nebo stránky.

Obsahem může být email, fyzická adresa, telefonní číslo nebo odkaz na profil na sociálních sítích.

html

```
1 <address>
2   <p>Napsal <a href="mailto:jan@novak.cz">Jan Novák</a>.</p>
3   <p>V případě dotazů nás navštivte na:</p>
4   <p>Příkladová 123, Praha, Česká republika</p>
5 </address>
```

<abbr>: Zkratka

Slouží k označení zkratky nebo akronymu. Pomocí atributu `title` lze poskytnout plné znění zkratky, které se zobrazí po najetí myší (tooltip).

Určujeme zejména pro asistivní technologie, aby uživatelé pochopili význam zkratky.

Atributy:

title Poskytuje plné znění zkratky nebo akronymu. Tento text se zobrazí jako tooltip při najetí myší na zkratku.

html
```
1 <p>Studuji na <abbr title="České vysoké učení technické">ČVUT</abbr>
  v Praze.</p>
```

<mark>: Zvýraznění textu

Slouží k vizuálnímu zvýraznění části textu pro referenční účely, kvůli její relevanci v jiném kontextu. Často se používá například pro zvýraznění nalezených výrazů ve výsledcích vyhledávání.

Ve výchozím nastavení má text žluté pozadí.

html
```
1 <p>Výsledky hledání pro slovo "pes":</p>
2 <p>Můj <mark>pes</mark> je velmi hodný.</p>
```

<menu>: Seznam příkazů

Sémantická alternativa k tagu <ul>, která je určena pro seznam interaktivních příkazů (tlačítek), nikoliv pro seznam položek obsahu. Dříve byl tento tag zavržen, ale v HTML5 se vrátil jako kontejner pro panel nástrojů (toolbar).

html
```
1 <menu>
2   <li><button>Kopírovat</button></li>
3   <li><button>Vyjmout</button></li>
4   <li><button>Vložit</button></li>
5 </menu>
```

<meter>: Skalární hodnota

Slouží k zobrazení hodnoty v rámci známého rozsahu (např. využití disku, teplota). Nejedná se o ukazatel průběhu (k tomu slouží <progress>), ale o statickou hodnotu „měřáku“.

Barva identifikátoru se počítá automaticky na základě hodnot a nastavených atributů `low`, `high` a `optimum`.

Atributy:

min Určuje minimální hodnotu měřáku. Výchozí je `0`.

max Určuje maximální hodnotu měřáku. Výchozí je `1`.

low Určuje spodní hranici „nízké“ hodnoty. Hodnoty pod touto hranicí jsou považovány za nízké.

high Určuje horní hranici „vysoké“ hodnoty. Hodnoty nad touto hranicí jsou považovány za vysoké.

optimum Určuje optimální hodnotu měřáku. Hodnoty blíže této hodnotě jsou považovány za lepší.

html

```html
<label>Využití disku C:</label>
<meter min="0" max="100" low="20" high="80" optimum="10"
value="60">60%</meter>
```

<progress>: Ukazatel průběhu

Slouží k zobrazení průběhu dokončení úkolu (progress bar). Používá se pro stahování souborů, instalace nebo vícekrokové formuláře.

Pokud není nastaven atribut `value`, indikuje probíhající činnost bez známého konce (neurčitý stav - „loading“).

Dovnitř tagu můžeme vložit textový popisek, který se zobrazí jako fallback pro prohlížeče, které nepodporují tento tag.

Atributy:

value Určuje aktuální hodnotu průběhu. Musí být menší nebo rovna hodnotě `max`.

max Určuje maximální hodnotu průběhu. Výchozí je `1`.

html

```html
<label>Stahování souboru:
<progress value="32" max="100">32 %</progress></label>
```

<optgroup>: Skupina možností

Slouží k seskupení souvisejících možností (`<option>`) uvnitř výběrového seznamu (`<select>`). Uživatel nemůže vybrat samotnou skupinu, pouze položky v ní.

Vytváří přehlednější strukturu v dlouhých seznamech.

html

```html
<select name="jidlo">
  <optgroup label="Ovoce">
    <option value="jablko">Jablko</option>
```

```
    <option value="banan">Banán</option>
  </optgroup>
  <optgroup label="Zelenina">
    <option value="mrkev">Mrkev</option>
    <option value="brambor">Brambor</option>
  </optgroup>
</select>
```

<picture>: Vícenásobné zdroje obrázku

Umožňuje definovat více zdrojů (<source>) pro jeden obrázek (<img>). Prohlížeč vybere nejvhodnější zdroj podle podmínek (šířka displeje, formát obrázku).

Používá se pro „Art Direction“ (např. oříznutý obrázek pro mobily a široký pro desktop) nebo pro moderní formáty (AVIF/WebP) s fallbackem na JPG/PNG. Vždy musí jako poslední prvek obsahovat tag <img>.

html

```
<picture>
  <source media="(min-width: 650px)" srcset="img_large.jpg">
  <source media="(min-width: 465px)" srcset="img_small.jpg">
  <img src="img_default.jpg" alt="Květina">
</picture>
```

<source>: Zdroj média

Definuje mediální zdroje pro tagy <picture>, <audio> nebo <video>. Není to samostatný prvek, používá se vždy uvnitř těchto rodičovských tagů.

Umožňuje prohlížeči vybrat formát, který podporuje (např. OGG vs MP3), nebo rozlišení vhodné pro dané zařízení.

Atributy:

src Určuje cestu k mediálnímu souboru (obrázku, zvuku nebo videu).

type Určuje MIME typ mediálního souboru (např. `image/webp`, `video/mp4`, `audio/ogg`).

media Používá se pouze uvnitř tagu <picture>. Určuje podmínky (media query), za kterých má být tento zdroj použit (např. šířka obrazovky).

html

```
<video controls>
  <source src="film.mp4" type="video/mp4">
  <source src="film.ogg" type="video/ogg">
  Váš prohlížeč nepodporuje video tag.
</video>
```

Tag <audio> a <video> jsou popsány v skriptech MVOP Webového inženýrství pro 3. ročník.

<small>: Drobný tisk

Původně sloužil jen pro zmenšení textu, ale v HTML5 má sémantický význam „drobného tisku“ (fine print). Používá se pro právní doložky, autorská práva, licenční podmínky nebo poznámky pod čarou.

html
```
<p>Cena produktu je 500 Kč.</p>
<small>Cena je uvedena bez DPH.</small>
```

<template>: Šablona obsahu

Slouží k uložení HTML kódu, který se po načtení stránky nemá zobrazit, ale může být později použit pomocí JavaScriptu. Obsah šablony je inertní – skripty se nespouští, obrázky se nenačítají, dokud není šablona naklonována a vložena do DOMu.

Ideální pro opakující se prvky generované dynamicky.

html
```
<template id="my-template">
  <div class="card">
    <h2>Název</h2>
    <p>Popis...</p>
  </div>
</template>
```

<hgroup>: Skupina nadpisů

Slouží k seskupení nadpisu (<h1>–<h6>) s podnadpisem (obvykle <p>), který nadpis rozvíjí. Umožňuje sémanticky svázat nadpis a jeho dovětek, aniž by dovětek vytvářel novou sekci v osnově dokumentu.

html
```
<hgroup>
  <h1>Pán Prstenů</h1>
  <p>Společenstvo Prstenu</p>
</hgroup>
```

<fieldset>: Skupina formulářových prvků

Slouží k logickému seskupení souvisejících prvků ve formuláři. Prohlížeče obvykle vykreslí rámeček kolem obsahu fieldsetu.

Často se používá spolu s tagem <legend>, který definuje popisek (nadpis) této skupiny a vkládá se jako první potomek. Užitečné pro rozdělení dlouhých formulářů na sekce (např. Osobní údaje, Dodací adresa).

html

```html
<form>
  <fieldset>
    <legend>Osobní údaje</legend>
    <label>Jméno: <input type="text"></label><br>
    <label>Email: <input type="email"></label>
  </fieldset>
</form>
```

<legend>: Popisek skupiny

Slouží k definování nadpisu pro element `<fieldset>`. Musí být uveden jako úplně první potomek uvnitř `<fieldset>`.

Je klíčový pro přístupnost formulářů. Čtečky obrazovky tento popisek oznámí uživateli při vstupu do skupiny polí, což pomáhá pochopit kontext (např. zda se zadává „Fakturační adresa“ nebo „Dodací adresa“). Vizuálně se text legendy často zobrazuje jako přerušení horní linie rámečku fieldsetu.

html

```html
<fieldset>
  <legend>Vyberte způsob platby</legend>
  <label><input type="radio" name="platba" value="karta"> Kartou</label>
  <label><input type="radio" name="platba" value="prevod"> Převodem</label>
</fieldset>
```

7.3 Další selektory v CSS

Selektor: Jedinečný potomek

Syntax: `E:only-child`,

Vybere všechny elementy `E`, které jsou v rámci jeho rodiče jediné. To znamená, že rodič má pouze jeden elementový potomek a tím je právě tento element `E`.

`E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

css

```css
button:only-child {
  border: 2px solid red;
}
```

Selektor: Jedinečný typ potomka

Syntax: `E:only-of-type`,

Vybere všechny elementy `E`, které jsou v rámci jeho rodiče jediné svého typu. To znamená, že rodič má pouze jeden elementový potomek typu `E` a tím je právě tento element `E`.

`E` je zde zástupný znak pro jakýkoliv jiný selektor. `E` nemusíme uvádět, potom by to znamenalo „všechny elementy“ – selektor všeho.

```css
button:only-of-type {
  border-top-left-radius: 1em;
}
```

Tyto dva následující pseudo-elementy se v rámci MVOP Webového inženýrství probírají, a je možné, že budou probrány bonusově i v tonto předmětu. Přestože jsou na toto téma lepší materiály v pokročilém předmětu, uvádím je zde pro úplnost.

Selektor: Obsah před obsahem

Syntax: `E::before`

Vybere obsah uvnitř elementu `E`, ale před jeho skutečným obsahem. Tedy vloží něco před to, co je v HTML. Tento pseudoelement se často používá k přidání dekorativního obsahu, jako jsou ikony nebo speciální znaky, bez nutnosti měnit HTML.

Pokud chceme přidat jakýkoliv obsah, je nutné použít vlastnost `content`.

Podobně funguje `E::after` pro výběr obsahu za obsahem elementu.

```css
div::before {
  content: ">  ";
  color: red;
}
```

7.4 Další vlastnosti v CSS

line-height: Výška řádku

Určuje, jak vysoký bude řádek textu.

Používáme zejména tehdy, kdy chceme mít větší či menší rozestupy mezi řádky textu, než je výchozí hodnota. Často z důvodu, že text s větší výškou řádku se lépe čte.

Definice hodnoty:

Možnosti zápisu:

- **Bez jednotky** – číslo, které se násobí s výškou písma (font-size). Například `1.5` jedna a půl násobek výšky písma,
- **S jednotkou** – například `20px`, což znamená pevnou výšku řádku dvacet pixelů, v případě použití jednotky **em** můžou vzniknout nepředvídatelné výsledky,
- **S použitím procent** – například `150%`, což znamená jedna a půl násobek výšky písma, taktéž ale může způsobit nepředvídatelné výsledky.

Preferuje se použití čísla bez jednotky.

```css
p {
  line-height: 1.5;
}
```

content: Obsah

Určuje obsah, který se má zobrazit v daném tagu.

Definice hodnoty:

Můžeme vkládat:

- Text, například `'Hello'`, ve kterém můžeme používat i speciální znaky jako `\A` pro nový řádek.
- URL obrázku, například `url(image.png)`,
- Speciální znaky, například `attr(data-info)` pro vložení hodnoty
- Počítadla, například `counter(section)` pro číslování sekcí,
- Prázdný obsah pomocí `""` (dvojité uvozovky),
- A další.

Vložený obsah v prohlížeči není interaktivní a nelze ho například zkopírovat.

```css
div::before {
  content: "Hello \A World";
  white-space: pre; /* Aby se respektovaly nové řádky */
}
```

pointer-events: Události myši

Určuje, jakým způsobem bude element reagovat na události myši, jako jsou kliknutí nebo pohyb myši.

Definice hodnoty:

- `auto` – Výchozí chování, element reaguje na události myši podle svého typu,
- `none` – Element ignoruje všechny události myši, což znamená, že kliknutí a pohyb myši budou ignorovány.

Pro SVG elementy existují i tyto hodnoty:

- `visiblePainted` – Element reaguje na události myši pouze tehdy, když je viditelný a má vyplněnou barvu,

- `visibleFill` – Element reaguje na události myši pouze tehdy, když je viditelný a má vyplněnou oblast,
- `visibleStroke` – Element reaguje na události myši pouze tehdy, když je viditelný a má ohraničení,
- `visible` – Element reaguje na události myši pouze tehdy, když je viditelný,
- `painted` – Element reaguje na události myši pouze tehdy, když má vyplněnou barvu,
- `fill` – Element reaguje na události myši pouze tehdy, když má vyplněnou oblast,
- `stroke` – Element reaguje na události myši pouze tehdy, když má ohraničení.

```css
.watermark {
  pointer-events: none;
}
```

user-select: Vybrání obsahu

Určuje, zda může uživatel vybrat text nebo jiný obsah na webové stránce.

Definice hodnoty:

- `auto` – Výchozí chování, prohlížeč rozhodne, zda je obsah vybratelný,
- `none` – Uživatel nemůže vybrat obsah,
- `text` – Uživatel může vybrat textový obsah,
- `all` – Uživatel může vybrat celý obsah najednou.

V rámci různých prohlížečů je nutné u této vlastnosti používat i prefixy pro zajištění kompatibility, například `-webkit-user-select` pro WebKit prohlížeče.

```css
.pagination .page-number {
  user-select: none;
}
```

aspect-ratio: Poměr velikosti

Určuje poměr mezi šířkou a výškou elementu.

Definice hodnoty:

- Poměr může být zadán jako dvě čísla oddělená lomítkem, například `16/9` pro širokoúhlý formát,
- Nebo jako jedno číslo, například `1.5`, což znamená, že šířka je 1.5krát větší než výška.

```
.profile .avatar {
  aspect-ratio: 1 / 1;
  border-radius: 50%;
}
```

rotate: Otočení prvku

Určuje úhel otočení elementu kolem jeho středu. Velmi podobné jako funkce `rotate()` ve vlastnosti `transform`.

Oproti vlastnosti `transform` je jednoduší z důvodu pořadí aplikování transformací.

Definice hodnoty:

Úhel může být zadán v různých jednotkách:
- `deg` (stupně),
- `rad` (radiány),
- `turn` (otáčky),
- `grad` (grady).

Stejně jako vlastnost `transform` nemodifikuje skutečnou pozici elementu v dokumentu, ale pouze jeho vizuální zobrazení.

```
1 .play-button {
2   rotate: 45deg;
3 }
```

scale: Velikost prvku

Určuje měřítko (zvětšení nebo zmenšení) elementu. Velmi podobné jako funkce `scale()` ve vlastnosti `transform`.

Oproti vlastnosti `transform` je jednoduší z důvodu pořadí aplikování transformací.

Definice hodnoty:

Jde zadat až tři hodnoty, kdy postupně reprezentují měřítko na ose X, Y a Z. Pokud je zadána pouze jedna hodnota, použije se pro všechny osy stejná hodnota.

Každá z hodnot může být:
- Číslo bez jednotky (například `1.5` pro zvětšení o 50%),
- Procenta (například `150%` pro zvětšení o 50%).

Je možné zadat i negativní hodnoty pro zrcadlení elementu.

Stejně jako vlastnost `transform` nemodifikuje skutečnou pozici elementu v dokumentu, ale pouze jeho vizuální zobrazení.

```
1 .icon {
2   scale: 1.2;
3 }
```

translate: Posun prvku

Určuje posun elementu v prostoru. Velmi podobné jako funkce `translate()` ve vlastnosti `transform`.

Oproti vlastnosti `transform` je jednoduší z důvodu pořadí aplikování transformací.

Definice hodnoty:

Jde zadat až tři hodnoty, kdy postupně reprezentují posun na ose X, Y a Z. Pokud je zadána pouze jedna hodnota, použije se pro osu X a osa Y bude mít hodnotu `0`.

Každá z hodnot může být:
- Hodnota s jednotkou (například `10px`, `2em`, `5%`),
- Hodnota `0` pro žádný posun.

Stejně jako vlastnost `transform` nemodifikuje skutečnou pozici elementu v dokumentu, ale pouze jeho vizuální zobrazení.

```
1 .tooltip {
2   translate: 0 10px;
3 }
```

zoom: Přiblížení prvku

Určuje úroveň přiblížení (zoomu) elementu. Na rozdíl od vlastnosti `transform: scale()`, která mění pouze vizuální zobrazení, vlastnost `zoom` mění i skutečnou velikost elementu v dokumentu.

Definice hodnoty:

Možné hodnoty jsou:
- Číslo bez jednotky (například `1.5` pro zvětšení o 50%),
- Procenta (například `150%` pro zvětšení o 50%),
- Klíčové slovo `normal`, které znamená výchozí velikost (100%).

```
1 .diagram {
2   zoom: 120%;
3 }
```

text-overflow: Přetečení textu

Určuje, jakým způsobem se má zobrazit text, který se nevejde do svého kontejneru a přetéká. Obvykle se používá v kombinaci s `white-space: nowrap` a `overflow: hidden`.

Definice hodnoty:

- `clip` – Text je jednoduše uříznut na okraji kontejneru,
- `ellipsis` – Text je zakončen třemi tečkami (...), což naznačuje, že pokračuje dál.

U hodnoty `ellipsis` je důležité, aby byl kontejner s pevnou šířkou a aby bylo nastaveno `white-space: nowrap` a `overflow: hidden`, jinak se efekt neprojeví.

css

```
1 p.truncate {
```

```
2   white-space: nowrap;
3   overflow: hidden;
4   text-overflow: ellipsis;
5 }
```

white-space: Práce s bílými znaky

Určuje, jak prohlížeč zachází s mezerami, tabulátory a zalomením řádků uvnitř elementu.

Definice hodnoty:

- `normal` – Výchozí chování, více mezer se sloučí, text se zalamuje,
- `nowrap` – Text se nikdy nezalomí na nový řádek, pokračuje dál i mimo kontejner,
- `pre` – Zachová všechny mezery a konce řádků (to v základu dělá tag `<pre>`),
- `pre-wrap` – Zachová mezery, ale zalamuje text, pokud je řádek příliš dlouhý,
- `pre-line` – Sloučí mezery, ale zachová konce řádků.

css

```
1 div {
2   white-space: pre-wrap;
3 }
```

word-break: Zalamování slov

Určuje pravidla pro zalamování slov na konci řádku.

Definice hodnoty:

- `normal` – Použije výchozí pravidla pro zalamování,
- `break-all` – Umožní zalomit slovo na jakémkoliv místě znaku, aby se vešlo na řádek (vhodné pro velmi dlouhá slova bez mezer),
- `keep-all` – Zakáže zalamování v čínštině, japonštině a korejštině (CJK), pro ostatní jazyky se chová jako `normal`.

Novinkou je hodnota `auto-phrase`, která umožňuje zalamování na základě frází v konkrétních jazycích (zatím není široce podporována).

css

```
1 p.long-urls {
2   word-break: break-all;
3 }
```

text-wrap: Typografie zalamování

Řídí způsob, jakým je text zalamován do odstavců, s cílem zlepšit typografickou kvalitu a čitelnost.

Definice hodnoty:

- `wrap` – Klasické zalamování,

- `nowrap` – Text se nezalamuje,
- `balance` – Vyváží délku řádků tak, aby byly vizuálně podobné (vhodné pro nadpisy),
- `pretty` – Optimalizuje zalamování, aby se předešlo osamoceným slovům na konci odstavce (sirotkům).

```css
h1 {
  text-wrap: balance;
}
```

hyphens: Dělení slov

Určuje, jak se mají slova dělit na konci řádku pomocí spojovníku (pomlčky). Pro správnou funkci musí být nastaven jazyk dokumentu (`lang` atribut v HTML).

Definice hodnoty:

- `none` – Slova se nedělí,
- `manual` – Slova se dělí pouze tam, kde je ručně vložen znak `&shy;`[65],
- `auto` – Prohlížeč automaticky dělí slova podle slovníku daného jazyka.

```css
p {
  hyphens: auto;
}
```

break-after: Zalomit po

Určuje, jak se má lámat stránka, sloupec nebo oblast (region) po vyrenderovaném elementu. Je to moderní náhrada za starší vlastnost `page-break-after`, která funguje univerzálněji i pro sloupce a další kontexty.

Vlastnosti `break-before` a `break-inside` fungují analogicky pro zlomy před elementem a uvnitř elementu.

Definice hodnoty:

- `auto` – Výchozí chování, prohlížeč rozhodne o zlomu.
- `avoid` – Zabraňuje zlomu (stránky či sloupce) bezprostředně po elementu.
- `always` – Vynutí zlom po elementu (chová se jako `page` v tiskovém kontextu).
- `all` – Vynutí zlom stránky i sloupce (pokud je v kontextu sloupců).
- `page` – Vynutí zlom stránky.
- `column` – Vynutí zlom sloupce.
- `left` / `right` – Vynutí zlom stránky tak, aby následující stránka byla levá nebo pravá.

```css
h2 {
```

[65] **S*oft** Hy*phenation v HTML způsobí, že pomlčka není vidět, ale prohlížeč může na tomto místě slovo rozdělit.

```
  break-before: page;
}
table {
  break-inside: avoid;
}
```

columns: Vícesloupcový layout

Složená vlastnost pro `column-width` a `column-count`.

Umožňuje rozdělit text do více sloupců, podobně jako v novinách.

Definice hodnoty:

- Lze zadat šířku sloupce (číslo s jednotkou, např. `200px`) nebo počet sloupců (číslo, např. `3`), případně obojí.

Jako další vlastnosti pro úpravu sloupců jsou dostupné:

- `column-gap` – Určuje vzdálenost mezi sloupci,
- `column-rule` – Definuje čáru mezi sloupci (barva, styl, šířka), znovu složená vlastnost,
- `column-span` – Umožňuje elementu přesáhnout více sloupců (např. nadpis přes všechny sloupce),
- `column-fill` – Řídí, jak se obsah rozděluje mezi sloupce (např. vyváženě nebo postupně).

css

```
article {
  columns: 300px 3;
  column-gap: 2rem;
}
```

float: Plovoucí elementy

Určuje, že element má být vyjmut z normálního toku dokumentu a „plavat“ na levé nebo pravé straně kontejneru. Ostatní text ho bude obtékat.

Definice hodnoty:

- `left` – Element plave vlevo,
- `right` – Element plave vpravo,
- `none` – Element neplave (výchozí).

css

```
img {
  float: right;
  margin-left: 10px;
}
```

clear: Ukončení obtékání

Určuje, zda element může přiléhat k plovoucím (`float`) elementům, nebo zda se musí posunout pod ně.

Definice hodnoty:

- `left` – Posune se pod plovoucí elementy vlevo,
- `right` – Posune se pod plovoucí elementy vpravo,
- `both` – Posune se pod všechny plovoucí elementy (často používané pro patičku).

```css
.footer {
  clear: both;
}
```

mask: Maskování

Umožňuje skrýt část elementu pomocí obrázku masky (alfa kanálu). Funguje podobně jako `background`, ale ovlivňuje viditelnost.

Definice hodnoty:

- Odkaz na obrázek (URL), režim masky, pozice atd.,
- Často vyžaduje prefix `-webkit-mask` pro konkrétní prohlížeče.

```css
.avatar {
  mask: url(circle-mask.png) no-repeat center / contain;
}
```

place-content: Zarovnání obsahu

Zkratka pro vlastnosti `align-content` a `justify-content` v Grid nebo Flexbox layoutu. Zarovnává celý obsah kontejneru najednou.

Definice hodnoty:

- První hodnota je pro blokovou osu (align), druhá pro řádkovou (justify),
- `center`, `start`, `end`, `space-between`, `space-around`, `stretch`.

```css
.grid-container {
  display: grid;
  place-content: center;
}
```

place-items: Zarovnání položek

Zkratka pro vlastnosti `align-items` a `justify-items` v Grid nebo Flexbox layoutu. Nastavuje výchozí zarovnání pro všechny položky v mřížce.

Definice hodnoty:

- `center` – Vycentruje položky v jejich buňkách vodorovně i svisle,
- `stretch`, `start`, `end`.

```css
.grid-container {
  display: grid;
  place-items: center;
}
```

place-self: Zarovnání sebe sama

Zkratka pro vlastnosti `align-self` a `justify-self`. Umožňuje konkrétní položce v Gridu přepsat globální zarovnání nastavené na kontejneru.

Definice hodnoty:

- `center` – Vycentruje tuto konkrétní položku,
- `stretch`, `start`, `end`, `auto`.

```css
.item-special {
  place-self: end center;
}
```

7.5 Průchod pomocí klávesnice

V rámci přístupnosti webu je důležité umožnit uživatelům navigaci pomocí klávesnice. Pro průchdo můžeme používat pouze zmáčknutí klávesy Tab (a Shift+Tab pro zpětný směr). Případně jiné klávesy, které jsme si v prohlížeči nastavili.

Průchod pomocí klávesnice by měl dovolit jednoduše přepínat mezi těmi nejdůležitějšími interaktivními prvky na stránce, jako jsou odkazy, tlačítka, formulářové prvky atd. Uživatel s nimi pak může interagovat bez nutnosti používat myš. Může tedy začít psát, zmáčknout Enter pro odeslání formuláře, mezerník pro stisknutí tlačítka atd.

Mimo hendikepované uživatele je to také užitečné pro rychlou navigaci na klávesnici bez nutnosti sundávat ruce z klávesnice (užitečné např. pro vývojáře).

Každý prvek na webové stránce má tzv. **focus order** (pořadí zaostření), což je pořadí, ve kterém se na něj dostaneme pomocí klávesy Tab. Ze základu mají všechny prvky stejnou „prioritu“ a pořadí je určeno jejich umístěním v HTML dokumentu (odshora dolů, zleva doprava). To tedy znamená, že vnořenější prvky se dostanou na řadu až po prvcích vnějších.

Někdy ale chceme toto pořadí upravit, například pokud máme navigační menu na boku stránky, ale v HTML je umístěno až na konci dokumentu. K tomu slouží atribut `tabindex`, který můžeme přidat na jakýkoliv HTML element. Hodnoty atributu `tabindex`:

- `tabindex="0"` – Prvek je zařazen do přirozeného pořadí podle jeho umístění v dokumentu a budou „vybrány“ až po všech prvcích s kladným tabindexem,
- `tabindex="-1"` – Prvek není zařazen do pořadí průchodu pomocí klávesnice, ale může být zaostřen programově (například pomocí JavaScriptu),
- `tabindex="kladné číslo"` – Prvek je zařazen do pořadí průchodu podle hodnoty tohoto čísla. Prvky s nižším číslem budou „vybrány“ dříve než prvky s vyšším číslem. Použití kladných čísel se ale nedoporučuje, protože může vést k nepředvídatelnému pořadí a zhoršit přístupnost.

Maximální hodnota `tabindex` je stanovena na 32767[66].

Většinou se ale použití `tabindex` nedoporučuje, protože jeho použití většinou značí nelogickou strukturu HTML dokumentu. Měli bychom se snažit strukturovat HTML tak, aby přirozené pořadí odpovídalo logickému toku stránky.

7.6 Směr textu

V rámci mezinárodních webů je důležité správně nastavit směr textu podle jazyka. Pro jazyky jako je angličtina, čeština nebo němčina je směr textu zleva doprava (LTR - Left to Right). Naopak pro jazyky jako je arabština, hebrejština nebo persština je směr textu zprava doleva (RTL - Right to Left). Nesprávné nastavení směru textu může vést k nečitelnému textu a špatnému uživatelskému zážitku.

Směr textu se nastavuje pomocí atributu `dir` na HTML elementu nebo na konkrétním bloku textu. Hodnoty atributu `dir`:

- `dir="ltr"` – Text je zleva doprava (výchozí pro většinu jazyků),
- `dir="rtl"` – Text je zprava doleva.

Samotné CSS je připravené na to, že v rámci jiných směrů textu přestane dávat smysl „left“ a „right“. Proto nějaké vlastnosti misto nich dovoluji používat jiné hodnoty. Například u flexboxu můžeme použít `flex-start` a `flex-end`, které se přizpůsobí směru textu. Nebo u zarovnání textu můžeme použít `start` a `end` místo `left` a `right`.

7.7 Další at-pravidla

Nyní si ukážeme několik dalších at-pravidel v CSS, které nejsou tak běžné, ale mohou být užitečné v určitých situacích.

At-pravidlo `@supports` umožňuje aplikovat CSS styly pouze tehdy, pokud prohlížeč podporuje určitou vlastnost nebo hodnotu. To je užitečné pro zajištění kompatibility s různými prohlížeči a pro použití moderních vlastností pouze tam, kde jsou dostupné. Jako parametry přijímá podmínky ve formátu `property: value`. Uvnitř definované pravidla se aplikují pouze tehdy, pokud lze daná vlastnost použít s danou hodnotou.

```css
Použití @supports
```

[66] Pro matematiky a programátory: je to $2^{15} - 1$

```css
@supports (display: grid) {
  .container {
    display: grid;
  }
}
```

Tímto můžeme taktéž vytvořit tzv. fallback pravidla, které se použí vždy, a když prohlížeč podporuje moderní vlastnost, použije jiné pravidlo, které bude mít větší specificitu.

At-pravidlo `@scope` umožňuje definovat lokální kontext pro CSS selektory. To znamená, že můžeme omezit platnost určitých stylů pouze na specifickou část dokumentu, aniž bychom museli opakovaně psát dlouhé selektory. Uvnitř bloku `@scope` můžeme používat relativní selektory, které se vztahují k definovanému kontextu.

Použití @scope

```css
@scope (.card) {
  .title {
    font-size: 1.5em;
    color: blue;
  }
}
```

Výše uvedené pravidlo aplikuje styl na všechny elementy s třídou `title`, které jsou potomky elementů s třídou `card`.

Závěr

8.1 Shrnutí

Máme za sebou rok plný nových pojmů, závorek a středníků. Prošli jsme cestu od prostého pochopení toho, jak vlastně funguje internet, až po vytvoření vaší vlastní, plně funkční a interaktivní webové stránky.

V těchto skriptech jsme společně probrali základní stavební kameny webu:

- **HTML a struktura**: Naučili jste se, že web není jen o tom, jak věci vypadají, ale především o tom, co znamenají. Chápete sílu sémantiky a víte, jak postavit pevnou kostru dokumentu.
- **CSS a design**: Zjistili jste, že stylování není jen o barvičkách, ale o komplexním systému pravidel. Osvojili jste si Box model, naučili jste se krotit layouty pomocí Flexboxu a Gridu a pochopili jste, jak udělat web responzivní pro mobilní zařízení.
- **Interaktivita a JavaScript**: Nakoukli jsme pod pokličku programování webu. Umíte zpracovat formuláře, reagovat na kliknutí uživatele a měnit obsah stránky za běhu pomocí DOMu.

Nebylo to vždy jednoduché. Možná jste strávili hodiny hledáním chybějícího středníku nebo bojem s prvky, které se odmítaly vycentrovat. Ale právě řešením těchto problémů jste se naučili přemýšlet jako vývojáři.

8.2 Motivace do budoucna

Tímto předmětem pro vás povinná výuka webových technologií končí. Jsem si vědom toho, že ne každý z vás se chce stát webovým vývojářem, a to je naprosto v pořádku.

Pro ty, kteří se webům dále věnovat nechtějí: Doufám, že si odnášíte alespoň pochopení toho, jak funguje svět, který vás denně obklopuje na obrazovkách vašich telefonů a počítačů. Schopnost podívat se „pod kapotu“ webové stránky, rozumět bezpečnosti na internetu a chápat strukturu informací se vám bude hodit v jakémkoliv technickém oboru, který si zvolíte.

Pro ty, které tvorba webů zaujala: Tohle byl teprve začátek. To, co jsme se naučili, je nezbytný základ, ale svět webového vývoje je mnohem hlubší. Pokud vás bavilo tvořit, experimentovat a vidět okamžité výsledky své práce, budu rád, když se potkáme v 3. a 4. ročníku na volitelném předmětu Webové inženýrství. Tam se podíváme na pokročilé frameworky, serverovou část aplikací, databáze a moderní nástroje, které používají profesionálové. Případně v jiných předmětech. Samozřejmě můžete pokračovat i samostudiem – na konci této kapitoly najdete několik doporučených zdrojů.

Ať už se vaše cesty vydají jakýmkoliv směrem, pamatujte si, že schopnost učit se nové technologie je tou nejdůležitější dovedností, kterou jste zde mohli získat.

8.3 Další zdroje informací

Pokud chcete své znalosti dále rozvíjet nebo si jen něco zopakovat, doporučuji tyto ověřené zdroje, které vám pomohou lépe pochopit látku z tohoto ročníku:

Dokumentace a referenční příručky:

- MDN Web Docs – bible webového vývojáře. Pokud si nejste jistí, jak funguje nějaký tag nebo CSS vlastnost, hledejte to primárně zde,

- Can I Use – nástroj pro ověření, zda vámi zvolená technologie funguje ve všech prohlížečích.

Interaktivní učení hrou:
- Flexbox Froggy – skvělá hra na procvičení Flexboxu,
- Grid Garden – hra na pochopení principů CSS Gridu.

Komunity a návody:
- W3Schools – jednoduché a rychlé ukázky kódů pro začátečníky,
- Stack Overflow – největší komunita vývojářů,
- Dev.to – články a tutoriály od vývojářů,
- CSS-Tricks – pokročilé techniky a tutoriály pro CSS.

Přeji vám hodně úspěchů v dalším studiu a doufám, že vám tyto základy dobře poslouží, ať už budete programovat cokoliv.

Příloha A.

Ověření znalostí

A.1 Internet a web

- [ ] Umím vysvětlit, co je to počítačová síť a jaký je rozdíl mezi architekturou peer-to-peer a client-server.
- [ ] Dokážu popsat a rozlišit typy sítí podle velikosti (PAN, LAN, WAN).
- [ ] Umím vysvětlit rozdíl mezi veřejnou a soukromou sítí.

- [ ] Dokážu stručně popsat historii internetu a roli sítě ARPANET.
- [ ] Umím vysvětlit rozdíl mezi pojmy Internet a World Wide Web (WWW).
- [ ] Dokážu vysvětlit pojem hypertextový dokument a k čemu slouží hyperlinky.

- [ ] Umím definovat, co je to doména a proč se používá místo IP adres.
- [ ] Dokážu popsat hierarchii domén a vysvětlit pojmy TLD a doména druhého řádu.
- [ ] Vím, co je to subdoména a dokážu uvést příklady jejího využití.
- [ ] Dokážu vysvětlit pojem IDN a důvody, proč se u českých domén často nepoužívá.

- [ ] Umím rozepsat strukturu URL adresy a popsat význam jednotlivých částí (protokol, cíl, port, cesta, parametry, fragment).
- [ ] Vím, proč se v URL používá procentové kódování a jak se řeší speciální znaky.
- [ ] Dokážu vysvětlit rozdíl mezi webovým vyhledávačem a webovým katalogem.
- [ ] Umím vysvětlit rozdíl mezi pojmy Deep web a Dark web.

- [ ] Dokážu popsat funkci protokolu HTTP a vysvětlit, v čem se liší od HTTPS.
- [ ] Vím, k čemu slouží protokol IP a jaký je rozdíl mezi IPv4 a IPv6.
- [ ] Umím vysvětlit princip fungování DNS a k čemu slouží cachování záznamů.
- [ ] Dokážu popsat účel protokolu FTP a model, na kterém funguje.

- [ ] Dokážu vyjmenovat a popsat základní typy malwaru (virus, spyware, trojský kůň, červ, adware).
- [ ] Umím vysvětlit princip fungování ransomwaru a proč je nebezpečný.
- [ ] Dokážu definovat pojem phishing a popsat, jaké techniky útočníci využívají.
- [ ] Umím vysvětlit rozdíl mezi antivirem a antimalwarem.

- [ ] Dokážu vysvětlit pojem digitální stopa a rozlišit vědomou a nevědomou stopu.
- [ ] Vím, co jsou metadata a jaké informace mohou obsahovat.
- [ ] Umím vysvětlit funkci webového prohlížeče a co zajišťuje vykreslovací jádro.
- [ ] Dokážu popsat projekt Chromium a jeho vztah k ostatním prohlížečům.
- [ ] Dokážu vysvětlit nevýhody dominance jádra Chromium na trhu prohlížečů.

- [ ] Umím vysvětlit princip digitálních certifikátů a asymetrického šifrování (veřejný a soukromý klíč).
- [ ] Umím vysvětlit, co je to self-signed certifikát a proč ho prohlížeče označují jako nedůvěryhodný.
- [ ] Vím, co je to certifikační autorita a jaká je její role při ověřování pravosti stránek.

- [ ] Dokážu vysvětlit rozdíly mezi serverhostingem, webhostingem a freehostingem.
- [ ] Umím vysvětlit základní právní aspekty webu (autorský zákon, Creative Commons, GDPR, cookies).
- [ ] Dokážu definovat, co je to CMS a k čemu slouží.

A.1.1 Praktická část

- [ ] Umím v prohlížeči zjistit, do kdy je platný certifikát navštívené webové stránky a kdo jej vydal.
- [ ] Dokážu v cizí URL adrese bezpečně identifikovat doménu, na kterou odkaz skutečně vede.
- [ ] Umím v textu URL adresy rozpoznat parametry a jejich hodnoty.

- [ ] Umím si v nastavení prohlížeče zkontrolovat a případně upravit úroveň ochrany soukromí.
- [ ] Umím rozlišit bezpečné připojení (HTTPS) od nebezpečného (HTTP) v adresním řádku.
- [ ] Dokážu najít a identifikovat verzi prohlížeče, který právě používám.

A.1.2 Bonusové části

- [ ] Vím, kdy a jaká první instituce v České republice byla připojena k internetu.
- [ ] Dokážu popsat, jak funguje překlad znaků v IDN na ASCII (např. punycode).
- [ ] Vím, co jsou root DNS servery a kolik jich přibližně na světě je.
- [ ] Dokážu vysvětlit pojem hash souboru a jak může pomoci při ověřování nezávadnosti stahovaného softwaru.

- [ ] Umím vysvětlit, proč se používá soubor robots.txt a co v něm lze nastavit.
- [ ] Vím, jaké porty standardně využívají protokoly HTTP a HTTPS.

A.2 Webové stránky

- [ ] Umím vysvětlit, co je to HTML a jaký je jeho rozdíl oproti programovacím jazykům.
- [ ] Dokážu vyjmenovat a popsat základní strukturu HTML dokumentu.
- [ ] Vím, co je to sémantika a proč je důležitá pro vyhledávače a přístupnost.
- [ ] Umím vysvětlit rozdíl mezi párovým a nepárovým tagem.

- [ ] Dokážu vysvětlit rozdíl mezi absolutní a relativní cestou k souboru.
- [ ] Vím, jaký je rozdíl mezi identifikátorem (`id`) a třídou (`class`) a kdy které použít.
- [ ] Umím vysvětlit pojem Box model a vyjmenovat jeho čtyři části.
- [ ] Dokážu vysvětlit rozdíl mezi `box-sizing: content-box` a `border-box`.

- [ ] Umím popsat rozdíl mezi blokovými (`block`) a řádkovými (`inline`) elementy.
- [ ] Dokážu vysvětlit rozdíl mezi jednotkami `px`, `em`, `rem` a `%`.
- [ ] Vím, jak funguje kaskáda a specificita v CSS (které pravidlo vyhraje).

- [ ] Umím vysvětlit rozdíl mezi prvky `div` a `span` a kdy je (ne)používat.

- [ ] Dokážu popsat rozdíly mezi rastrovou a vektorovou grafikou a uvést příklady formátů (JPG, PNG, SVG).
- [ ] Vím, k čemu slouží atribut `alt` u obrázků a proč je povinný.
- [ ] Umím vysvětlit rozdíl mezi obrázkem vloženým pomocí `<img>` a přes CSS `background-image`.
- [ ] Dokážu vyjmenovat tři typy seznamů v HTML (`ul`, `ol`, `dl`) a popsat jejich použití.

- [ ] Vím, proč se tabulky nemají používat pro rozložení (layout) stránky.
- [ ] Umím popsat základní strukturu tabulky (`table`, `tr`, `td`, `th`).
- [ ] Dokážu vysvětlit funkci atributů `colspan` a `rowspan`.
- [ ] Umím vysvětlit, co je to resetování stylů (CSS reset) a proč se používá.

A.2.1 Praktická část

- [ ] Umím napsat základní kostru HTML5 dokumentu bez nápovědy.
- [ ] Dokážu propojit HTML soubor s externím CSS souborem pomocí tagu `link`.
- [ ] Umím vytvořit funkční hypertextový odkaz na jinou webovou stránku i na sekci v rámci jedné stránky (kotva).
- [ ] Dokážu v CSS vybrat prvek podle jeho tagu, třídy, ID a vnoření.

- [ ] Umím nastavit barvu textu, velikost písma a font pomocí CSS.
- [ ] Dokážu vycentrovat blokový prvek vodorovně.
- [ ] Umím v CSS nastavit prvku vnější (margin) a vnitřní (padding) odsazení.
- [ ] Dokážu pomocí CSS změnit vzhled odkazu při najetí myši.

- [ ] Umím vložit do stránky obrázek se správnými atributy.
- [ ] Dokážu nastavit obrázek jako pozadí elementu a upravit jeho velikost.
- [ ] Umím vytvořit vnořený seznam (např. odrážkový seznam uvnitř číslovaného).
- [ ] Dokážu vytvořit tabulku s hlavičkou, tělem a patičkou.
- [ ] Umím používat pokročilé CSS selektory jako `:nth-child`, `:first-of-type` nebo atributové selektory.
- [ ] Umím používat vývojářské nástroje v prohlížeči (DevTools) pro ladění HTML a CSS.

A.2.2 Bonusové části

- [ ] Umím vysvětlit a použít moderní jednotky pro viewport (`dvh`, `lvh`, `svh`) a v čem se liší od klasického `vh`.
- [ ] Dokážu vysvětlit funkci vlastnosti `object-fit` pro obrázky.
- [ ] Vím, jak funguje líné načítání obrázků (`loading="lazy"`) a k čemu slouží atributy `srcset` a `sizes`.

- [ ] Dokážu vysvětlit rozdíl mezi `display: none`, `visibility: hidden` a `opacity: 0`.
- [ ] Vím, jaké jsou konvence pro pojmenovávání tříd a ID (např. kebab-case, BEM).

- [ ] Dokážu vysvětlit, jak funguje vlastnost `scroll-behavior: smooth`.

A.3 Rozložení stránek

- [ ] Umím vysvětlit význam sémantických tagů oproti použití elementu `div`.
- [ ] Umím popsat účel a vhodné použití tagů `header`, `nav` a dalších.
- [ ] Vím, kolikrát by se měl na stránce vyskytovat element `main` a co obsahuje.
- [ ] Umím vysvětlit rozdíl mezi elementy `article` a `section`.
- [ ] Chápu význam elementu `aside` a vím, jaký obsah do něj patří.

- [ ] Umím vysvětlit princip Flexboxu a rozdíl mezi hlavní a vedlejší osou.
- [ ] Vím, jak udělat z běžného elementu flex kontejner.
- [ ] Umím vyjmenovat a popsat hodnoty pro `justify-content` a `align-items`.
- [ ] Chápu, jak funguje vlastnost `flex-direction` a k čemu slouží `row-reverse` či `column`.
- [ ] Rozumím vlastnosti `flex-wrap` a vím, co se stane s položkami při nedostatku místa.
- [ ] Umím vysvětlit funkci vlastností `flex-grow`, `flex-shrink` a `flex-basis`.

- [ ] Umím vysvětlit rozdíl mezi Flexboxem (1D) a Gridem (2D).
- [ ] Chápu význam jednotky `fr` v rámci Grid layoutu.
- [ ] Rozumím funkci `repeat()` a `minmax()` při definici mřížky.
- [ ] Vím, jak fungují vlastnosti `justify-items` a `align-items` v kontextu Gridu.
- [ ] Umím vysvětlit rozdíl mezi `justify-content` a `align-content` u Gridu.

- [ ] Umím definovat, co je to viewport a jaký má vztah k responzivitě.
- [ ] Vím, proč je důležité nastavit `<meta>` tag s hodnotou viewport.
- [ ] Rozumím pojmu „bod zlomu“ (breakpoint) a k čemu slouží.
- [ ] Umím vysvětlit syntaxi a logiku `@media` pravidel (media queries).
- [ ] Chápu rozdíl mezi responzivitou pomocí `@media` a `@container`.

A.3.1 Praktická část

- [ ] Umím v kódu správně strukturovat stránku pomocí sémantických tagů.
- [ ] Umím vycentrovat prvek vodorovně i svisle pomocí Flexboxu.
- [ ] Umím změnit směr řazení prvků ve Flexboxu z řádku na sloupec.
- [ ] Umím nastavit mezery mezi položkami pomocí vlastnosti `gap`.
- [ ] Umím změnit pořadí položek ve Flexboxu pomocí vlastnosti `order` bez zásahu do HTML.
- [ ] Umím nastavit položce, aby se neroztahovala ani nesmršťovala (`flex-grow`, `flex-shrink`).

- [ ] Umím vytvořit mřížku se třemi sloupci, kde jeden zabírá polovinu místa a zbylé dva čtvrtinu.
- [ ] Umím umístit položku v Gridu tak, aby zabírala dva sloupce a dva řádky.
- [ ] Umím použít `grid-auto-rows` pro nastavení výšky automaticky přidaných řádků.
- [ ] Umím zapsat zkrácenou syntaxi pro umístění v mřížce pomocí lomítka (např. `1 / 3`).

- [ ] Umím napsat media query, která se aplikuje pouze na zařízení s maximální šířkou 700px.
- [ ] Umím využít logické operátory `and` nebo `or` v rámci media query.
- [ ] Umím používat relativní jednotky (`%`, `rem`, `vw`, `vh`) namísto pevných pixelů.
- [ ] Umím definovat kontejner pro container queries pomocí `container-name` a `container-type`.
- [ ] Umím napsat pravidlo `@container`, které změní styl prvku na základě šířky jeho rodiče.

A.3.2 Bonusové části

- [ ] Vím, co je `grid-template-areas` a jak pomocí něj pojmenovat oblasti v mřížce.
- [ ] Tuším, k čemu slouží vlastnost `grid-auto-flow` a jak ovlivňuje vkládání prvků.
- [ ] Znám funkci `style()` v rámci container queries pro dotazování na hodnoty CSS vlastností.
- [ ] Mám povědomí o funkci `scroll-state()` a jejím využití pro detekci „přilepení“ (sticky) prvku.
- [ ] Umím používat speciální jednotky pro container queries (`cqw`, `cqh`, `cqi` atd.).
- [ ] Umím využít media feature `prefers-color-scheme` pro implementaci tmavého režimu.
- [ ] Vím, jak zjistit, zda uživatel preferuje omezení animací pomocí `prefers-reduced-motion`.
- [ ] Rozumím principu `min-content` a `max-content` v CSS Gridu.

A.4 Design webových stránek

- [ ] Umím vysvětlit, jak importovat vlastní písmo pomocí pravidla `@font-face`.
- [ ] Znám běžné formáty souborů pro webová písma.
- [ ] Vím, jaké je doporučené maximální množství různých písem a jejich variant na jedné stránce.
- [ ] Rozumím rozdílu mezi použitím ikon jako fontů a jako SVG obrázků.
- [ ] Vím, jak zajistit přístupnost ikon pro čtečky obrazovky.

- [ ] Umím popsat, jak funguje vlastnost `box-shadow` a `text-shadow` a z jakých hodnot se skládá.
- [ ] Dokážu vysvětlit rozdíl mezi vlastností `filter` a `backdrop-filter`.
- [ ] Rozumím tomu, jak vlastnost `transform` ovlivňuje (nebo neovlivňuje) okolní prvky a tok dokumentu.
- [ ] Znám jednotky používané pro rotaci prvků.
- [ ] Vím, co určuje vlastnost `transform-origin`.

- [ ] Chápu princip interpolace v kontextu webových animací.
- [ ] Umím vysvětlit rozdíl mezi diskrétními a interpolovatelnými hodnotami.
- [ ] Dokážu popsat rozdíl mezi přechody (transitions) a animacemi (animations).
- [ ] Znám základní vlastnosti pro definici přechodu.
- [ ] Rozumím, k čemu slouží funkce časování (easing functions) jako `linear` nebo `ease-in-out`.
- [ ] Vím, jaké časové jednotky se v CSS používají.

- [ ] Rozumím struktuře pravidla `@keyframes` a jak se definují jednotlivé kroky animace.
- [ ] Umím vyjmenovat vlastnosti určující chování animace (trvání, opakování, směr, zachování stavu).
- [ ] Chápu, co dělá vlastnost `animation-fill-mode`.

- [ ] Dokážu vysvětlit rozdíl mezi pozicováním `static`, `relative`, `absolute`, `fixed` a `sticky`.
- [ ] Vím, k jakému prvku se vztahuje absolutně pozicovaný prvek.
- [ ] Rozumím tomu, co je to viewport a jak souvisí s fixním pozicováním.
- [ ] Chápu funkci vlastnosti `z-index` a kdy ji lze použít.
- [ ] Vím, proč se nedoporučuje používat pozicování pro tvorbu základního layoutu stránky.

- [ ] Umím popsat rozdíl mezi lineárním, radiálním a kuželovým přechodem.
- [ ] Dokážu vysvětlit, co je to softwarový framework a jak se liší od knihovny.
- [ ] Znám výhody a nevýhody používání CSS frameworků.
- [ ] Rozumím rozdílu mezi utility-first přístupem (Tailwind) a komponentovým přístupem (Bootstrap).
- [ ] Vím, jaký je rozdíl mezi UI (User Interface) a UX (User Experience).
- [ ] Znám základní principy dobrého designu a uživatelského zážitku.

A.4.1 Praktická část

- [ ] Umím napsat pravidlo `@font-face` a aplikovat ho na text.
- [ ] Zvládnu vložit ikonu z knihovny jako Font Awesome.
- [ ] Dokážu prvku nastavit stín a zaoblení rohů.
- [ ] Umím aplikovat filtr pro změnu jasu nebo rozmazání obrázku.
- [ ] Zvládnu prvek otočit, zvětšit nebo posunout pomocí `transform`.

- [ ] Umím vytvořit plynulý přechod barvy pozadí při najetí myší pomocí `transition`.
- [ ] Dokážu použít zkrácený zápis vlastnosti `transition`.
- [ ] Zvládnu vytvořit jednoduchou nekonečnou animaci pomocí `@keyframes` a vlastnosti `animation`.
- [ ] Umím nastavit animaci tak, aby po skončení zůstal prvek v posledním stavu.

- [ ] Dokážu vycentrovat absolutně pozicovaný prvek přesně na střed rodiče.
- [ ] Umím vytvořit fixní navigační lištu, která zůstává nahoře při scrollování.
- [ ] Zvládnu vytvořit prvek, který se při scrollování „přilepí“ pod horní okraj okna.
- [ ] Umím vytvořit lineární gradient jako pozadí tlačítka.

A.4.2 Bonusové části

- [ ] Rozumím pokročilé vlastnosti `transition-behavior` pro animaci diskrétních vlastností.
- [ ] Vím, k čemu slouží at-pravidlo `@starting-style`.
- [ ] Umím definovat vlastní křivku časování pomocí `cubic-bezier`.
- [ ] Chápu princip CSS kotev (Anchors) a vlastností `anchor-name` a `position-anchor`.
- [ ] Zvládnu vytvořit „schodovou“ animaci pomocí funkce `steps`.

- [ ] Rozumím principu 12sloupcového gridu v Bootstrapu.
- [ ] Vím, co je to Figma a k čemu slouží v procesu vývoje webu.

A.5 Interaktivita

- [ ] Umím vysvětlit, k čemu slouží HTML formuláře a jak se definují.
- [ ] Umím popsat rozdíl mezi metodami GET a POST při odesílání dat.
- [ ] Vím, co určuje atribut action v prvku form.
- [ ] Umím vyjmenovat základní typy vstupních polí input.
- [ ] Rozumím rozdílu mezi typy checkbox a radio.
- [ ] Vím, jak zajistit, aby skupina přepínačů radio fungovala společně.
- [ ] Umím vysvětlit funkci atributů placeholder, value a name.
- [ ] Rozumím významu a použití prvku label a jeho atributu for.
- [ ] Vím, jaký je rozdíl mezi prvky select a datalist.
- [ ] Umím popsat, k čemu slouží prvek textarea a jak se liší od input type=„text“.

- [ ] Umím vysvětlit, proč nelze věřit validaci dat na straně klienta.
- [ ] Vím, jaké atributy se používají pro základní validaci formuláře v HTML.
- [ ] Umím vysvětlit, co je MIME typ multipart/form-data a kdy je nutné ho použít.
- [ ] Vím, jakým způsobem se kódují data v URL při použití metody GET.

- [ ] Umím popsat rozdíly mezi deklarací proměnných pomocí var, let a const.
- [ ] Rozumím rozdílu mezi volným (==) a striktním (===) porovnáním.
- [ ] Umím vysvětlit, co je to DOM a jaký má vztah k HTML.
- [ ] Vím, jaký je rozdíl mezi objekty window a document.
- [ ] Umím popsat rozdíl mezi metodami querySelector a getElementById.
- [ ] Vím, co dělá vlastnost innerHTML a jaká rizika přináší.
- [ ] Rozumím principu událostí a k čemu slouží metoda addEventListener.
- [ ] Umím vysvětlit rozdíl mezi setTimeout a setInterval.

A.5.1 Praktická část

- [ ] Umím vytvořit formulář, který odesílá data metodou POST na zadanou URL.
- [ ] Umím správně propojit popisek label s odpovídajícím vstupním polem.
- [ ] Umím vytvořit rozbalovací seznam pomocí značek select a option.
- [ ] Umím nastavit pole jako povinné a omezit délku vstupního textu.
- [ ] Umím vložit JavaScriptový kód do HTML stránky.

- [ ] Umím v JavaScriptu napsat podmínku if-else a cyklus for.
- [ ] Umím vytvořit funkci v JavaScriptu a zavolat ji s parametry.
- [ ] Umím vybrat konkrétní HTML prvek pomocí JavaScriptu a změnit jeho textový obsah.
- [ ] Umím pomocí JavaScriptu přidat nebo odebrat CSS třídu z elementu.
- [ ] Umím zaregistrovat událost kliknutí na tlačítko a spustit při ní funkci.
- [ ] Umím zabránit výchozímu chování formuláře při odeslání pomocí JavaScriptu.

- [ ] Umím dynamicky vytvořit nový HTML prvek a vložit ho do stránky.

A.5.2 Bonusové části

- [ ] Umím nastylovat vlastní vzhled zaškrtávacího políčka pomocí CSS a pseudo-třídy :checked.
- [ ] Rozumím použití pseudo-třídy :has() v CSS.
- [ ] Vím, jak funguje validace pomocí regulárních výrazů v atributu pattern.
- [ ] Umím vysvětlit chování operátoru modulo v JavaScriptu u záporných čísel.
- [ ] Rozumím bitovým operátorům v JavaScriptu.
- [ ] Umím použít metodu animate() pro vytvoření animace v JavaScriptu.
- [ ] Znám pokročilé metody objektu console, jako jsou group nebo time.

Příloha B.

Seznamy

B.1 Seznam obrázků

B.2 Seznam kódových bloků

B.3 Seznam HTML tagů

Bonusové části

B.4 Seznam CSS vlastností

Bonusové části

B.5 Seznam CSS selektorů

Bonusové části

B.6 Seznam CSS jednotek

Bonusové části

B.7 Seznam CSS funkcí

Bonusové části

B.8 Seznam CSS at-pravidel

Bonusové části