OPAKOVÁNÍ 3. ROČNÍKU

Opakování nejdůležitějších témat 3. ročníku, předmětu WBA a dalších témat.

ÚVOD

Protože opakování látky z 3. ročníku je spousta (jen otázek existuje více jak 600), není v mých silách dopředu vytvořit podrobné opakování všechny látky, proto jsem se rozhodl tuto lekci věnovat spíše rozsáhlému širokému záběru, kde vysvětluji jednotlivé témata v jedné větě či souvětí. V hodinách, bude-li to potřeba, si povíme na Vaší žádost jakékoliv další detaily, které jsou nutné. Pokud to považujete za nutné, můžete si zde rovnou níže tvořit poznámky k daným tématům a více si je rozvést pro budoucí opakování.

Pro opakování si doporučuji otevírat prezentace z minulého roku (naleznete v dalších zdrojích) a i lekce, které tam naleznete. Spousta témat byla již opakována a sepsána jako maturitní otázky - ty víte, kde najít.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

WBA

HTML je značkovací jazyk, který se používá pro definování obsahu stránky. Používá tagy, které mohou být párové a nepárové. Všechny tagy mají svůj název a účel, pro který se používají.

CSS je stylovací jazyk, který pomocí pravidel definuje styly stránek. Pravidla se skládají ze selektorů, který vybírá určité prvky na stránce, a vlastností, kde každá určuje co se mění, a hodnot, které určují přesné parametry.

HTML5 je nejnovější verze HTML, která se zaměřuje na audio-vizuální podobu webu. Definovala taktéž nové tagy určené pro lepší rozdělení stránky (`header, footer, main, article, ...`).

Základní struktura HTML se skládá z `DOCTYPE`, `HTML`, `HEAD`, `TITLE`, `META` a `BODY`.

Index soubor je speciální soubor, který se zobrazí při načtení složky - při přístupu do složky se najde soubor index a ten se zobrazí.

HTML obsahuje například tagy:

- `p` - paragraf
- `b` - tlusté písmo
- `i` - kurzíva
- `u` - podtržení
- `sup` - horní index
- `sub` - dolní index
- `h1 - h6` - nadpisy
- `img` - obrázek
- `ul` - neseřazený list
- `ol` - seřazený list
- `li` - odrážka listu
- `table` - tabulka
- `tr` - řádek tabulky
- `th` - buňka nadpisu tabulky
- `td` - buňka tabulky
- `form` - formulář
- `input` - položka formuláře
- `textarea` - dlouhý text
- `label` - nadpis položky

Kaskáda je pojem, který popisuje, jak se chovají pravidla a vlastnosti CSS. Pravidla se prioritizují podle toho, jak je jejich selektor přesný vůči danému elementu. Druhá vlastnost je ta, že vlastnosti přechází z rodiče na potomka.

Pravidla se v CSS skládají ze selektoru, vlastností a jejich hodnot. Pravidla představují jeden balíček, který dle selektoru ovlivňuje různé prvky. Pravidla mohou být obalena v takzvaných at-pravidlech, které se aplikují pouze někdy, jako jsou např. responzivita či animace.

Selektory určují, jaké prvky na stránce se vyberou pro aplikaci vlastností v pravidlu. Jaké známe selektory můžeme uvést selektor dle tříd, dle identifikátoru, přímý dědic a jakýkoliv dědic.

Typy zobrazování jsou typy, jak se jednotlivé prvky zobrazují na stránce. Určujeme ji vlastnosti display. Mezi základní patří napr. řádkové - inline, blokové - block a například tam patří rozložení flex a grid.

CSS obsahuje například následující vlastnosti:

- `width` - šířka
- `height` - výška
- `display` - typ zobrazení
- `padding` - vnitřní odsazení
- `margin` - vnější odsazení
- `border` - ohraničení
- `outline` - obtažení
- `font` - font (velikost, typ, tloušťka)
- `color` - barva textu
- `background-color` - barvu pozadí
- `text-transform` - změna kapitálek a obráceně
- `transform` - změna prvku (natažení, zvětšení, ...)
- `transition` - určení přechodových animací
- `filter` - filtr (změna barev, sytost, ...)

Pseudotřídy jsou speciálním typem selektoru, který vybírá prvky na stránce, které jsou v nějakém stavu. Jako příklad můžeme uvést hover, který zaktivuje pravidlo právě tehdy, když je uživatel na daném prvku najetý myší.

Pseudoelementy jsou speciálním typem selektoru, který vytváří prvky na stránce (resp. je vybírá, i když v HTML nejsou definované) a tím můžeme na stránce vytvářet pro jeden prvek více vnitřních částí. Jako příklad můžeme uvést :after, který vytvoří prvek uvnitř vybraného prvku po všech vnitřních elementech uvnitř.

Flexbox je moderní způsob rozložení na stránce. Obecně se používá na jednodimenzionální rozložení, tedy prvky vedle sebe / pod sebou. Používá různé vlastnosti, jako je například flex-direction, flex-wrap, flex-shrink, gap a další.

Grid je moderní způsob rozložení na stránce. Používá se na dvourozměrné rozložení, je v podstatě náhradou tabulek. Používá vlastnosti jako grid-rows-template, grid-columns-template, grid-column, grid-row, gap a další.

Responzivita je způsob, jak upravujeme stránku tak, aby byla zobrazitelná na většině zařízení. Obyčejně tvoříme například stránku tak, že vytvoříme zobrazení pro desktop a poté vytvoříme tzv. breakpointy na menší zařízení.

Animace se na webových stránkách tvoří třemi způsoby, a to jako přechodové animace (v CSS pomocí transition a pseudotříd), jako pravé (v CSS pomocí tzv. keys), a komplexní v JS.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

UI A UX

Na **User Interface** (UI) a **User Experience** (UX) se zejména zaměřujeme na frontendu, což jsou vlastně webové stránky a aplikace, které vidí uživatel. Webový frontend často kombinuje se webovým backendem, tedy serverem, který např. poskytuje data a zpracovává akce. Tyto dvě části často potřebují dva rozdílné týmy, které se o části starají. Často však najdete full-stack developera, který se zaobírá celým cyklem stránek.

UI je pojem, který se zaobírá tím, jak stránka fakticky vypadá, tedy jedná se o její vzhled, volbu barev a fontů a další věci. Tuto část často navrhuje návrhář (grafik). Často je důležité dodržovat např. firemní identitu a další pravidla.

UX je pojem, který se zaobírá tím, jak je samotná stránka včetně UI vůči uživateli funkční, reaguje a odpovídá. Zjednodušeně popisuje uživatelův zážitek při používání aplikace. Často se řeší i to, co uživatel předpokládá, že dané prvky na stránce dělají a obdobně.

Pro UI a UX máme spoustu zásad, které nám napovídají k tomu, abychom měli co nejlepší stránku. Mezi základní zásady například patří konzistence, která nám říká, že by UI a UX mělo být konzistentní napříč stránkami. Tedy tlačítka se stejným názvem (ikonami) budou dělat podobné akce, stránky budou vypadat stejně a podobně. Jako další můžeme ukázat přívětivost, stručnost, zpětná vazba, standard, příjemnost a další.

Princip minimálního překvapení nám říká, že by stránka neměla být nepředvídatelná, třeba že tlačítko smaže veškeré věci bez toho, aby se to uživatele zeptalo a obdobně.

Pro obecné sledování aplikace, zejména toho, kam uživatele chodí či jak s aplikací reagují je nejlepší sledovat např. HTTP requesty a nebo pozorně uživatele pomocí komplexnějších sledovačů (např. Google Analytics, heatmapy, ...).

Pro UI je důležité vybírat správné barvy, obrázky a fonty, protože všechny tyto části ovlivňují uživatele. Důležité je taky práce s prázdným místem, aby stránky nebyla prázdná či naopak informace na ni nebyly naplácané u sebe.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

SASS

Sass je speciální preprocesor (meta jazyk), který zpracovává dva formáty SASS a SCSS, ze kterých generuje funkční CSS. SASS tedy přináší pouze nástavbu před generováním CSS, které slouží především pro zrychlení vývoje stránek. Mezi hlavní výhody patří např. funkce, mapy, cykly a další programové generování.

Pro použití SASS je nutné nejdřív pomocí kompilátoru (node-sass, dart-sass) zkompilovat daný soubor. Vygenerovaný soubor se poté linkuje normálním zápisem tagu link.

```
$primary: #fa9a9d;

body {
    background-color: $primary;
}
```

Do proměnných můžeme ukládat jakékoliv SASS hodnoty.

```
body {
    > section.about {
        a {
            &:hover {
                color: black;
            }
        }
    }
}
```

Vnořováním se automaticky tvoří složený selektor. Pomocí & označujeme aktuální selektor, ke kterému něco přidáváme. Výše uvedený kód vygeneruje:

```
body > section.about a:hover {
  color: black;
}
```

Jako důležité části považujme:

- moduly (@use k importování jiného Sass souboru či CSS)
- podmínky (a obecně řídící konstrukce)
- mixins a funkce
- předdefinované knihovny pro hodnoty (např. barvy, matematické)

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

JAVASCRIPT

JavaScript (dále často jako JS) je původně čistě skriptovací jazyk, který sloužil k vytvoření interakce uživatele se stránkou, například přidáním rozbalovacích menu či cokoliv jiného. Skriptovací v této podobě znamená, že komunikuje s DOM, což je virtuální stránka a ji upravuje.

V aktuální podobě se mu již může říkat programovací jazyk, který je schopen velmi mnoho věcí. Velmi velkým plusem je projekt Node.js, který dovolí spouštět JS v příkazové řádce, jako program, tedy zcela bez stránek. Npm je projekt, který pro Node.js poskytuje službu pro instalaci balíčků a správu projektu.

JavaScript nemá vnější datové typy (tj. není typovaný) a my je neurčujeme - proměnné existují bez jasného typu a uvnitř jich může být cokoliv. Interně však datové typy má a používá je, patří mezi ně například:

- číslo (`Number`) - desetinné i celé (i nekonečna)
- textový řetězec (`String`) - text
- objekt (`Object`)
- pole (`Array`) - které je ve skutečnosti list
- `undefined` - definuje, že je proměnná nedefinovaná
- `null` - definuje, že proměnná existuje, ale nemá hodnotu
- a další

JS používá tři typy proměnných:

- `var` - proměnná, která je při deklaraci již k dispozici na začátku bloku (nevyužívá se)
- `let` - proměnná, které lze po deklaraci přepisovat
- `const` - proměnná, která po prvotním přiřazení již nelze upravovat

V JS existují dva typy porovnání a to je striktní a nestriktní. Nestriktní porovnání (==) neporovnává datový typ, ale převede dané hodnoty na číslo, které poté porovná. Striktní zkontroluje datový typ a zároveň přesné hodnoty.

Pro sjednocování booleanových hodnot se používá `&&` jako AND, `||` jako OR a `!` jako negace. Podmínky se tvoří pomocí zápisu:

```
if(expr) {
    // do smth
} else if(expr) {
    // do smth
} else {
    // do smth
}
```

V JS existuje několik cyklů, základní je while, který běží do té doby, než je jedna podmínka pravdivá. Další typ je for, který je komplexnější:

```
while(expr) {
    // do smth
    break; // ukončí cyklus
```

```
    continue; // ukončí aktuální průchod a cyklus pokračuje
}

for(let i = 0; i < 10; i++) {
    console.log(i); // vypíše čísla 0 do 9.
}

for(let i in array) {
    console.log(i); // vypíše indexy daného pole
}

for(let i of array) {
    console.log(i); // vypíše hodnoty daného pole
}
```

Jak jsem již zmínil, pole je v JS vlastně list, tedy pole, které se automaticky zvětšuje na potřebnou velikost. V poli můžou být různé datové typy.

```
const array = [1, 3, 2, 1, 3, 312.2];

array.push(3);
array.pop(); // vrátí a smaže poslední prvek
array.shift(); // vrátí a smaže první prvek
array.splice(3, 2); // od indexu 3 smaže 2 prvky
// a další!

array.map((a) => {
    return a * 2;
}); // přemapuje a vrátí (původní zůstane) celé pole dle lambdy, výsledek tohoto je [2, 6, 4, 2

array.filter((a) => {
    return a < 20;
}); // vrátí prvky z pole, které splňují danou funkci, výsledek toho je [1, 3, 2, 1, 3, 3]

// a další metody, jako .reduce, .find, .some, ...
```

Document Object Model (DOM) je vlastně kopie HTML (i CSS) stránky, kterou můžeme pomocí JS upravovat.

```
const elements = document.querySelectorAll("a.link > span");

for(let element of elements) {
   element.addEventListener("click", (event) => {
       element.style.backgroundColor = "green";
   })
}
```

Sliby jsou v JS způsob asynchronního programování, tedy provádění více akcí na jednou. JS není čistě jazyk, který by dovoloval souběžné spouštění kódu, ale dovoluje prioritizaci (chvíli běží tohle, a až se stane tohle, poběží tohle, ...). Promises, neboli sliby, dovolují počkat na nějakou akci, až se dokončí, a potom se na ní zachytit pomocí metody .then. .catch se vyvolá v případě chyby. Pro promises existují předpřipravené metody .all, .race a další podobné.

```
new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("smth");
    }, 1500);
}).then((msg) => {
    console.log(msg);
}).catch((err) => {
    console.error(err);
};
```

Pro zjednodušení práci se sliby se používají klíčové slova `await` v `async` metodách, které čekají na splnění daného slibu a poté pokračuji v metodě.

Výjimky jsou označením chyb v kódu, které se stali při spouštění kódu. Výjimka se vyvolává pomocí slova `throw` a zachytit ji můžeme pomocí bloku `try-catch`.

O OOP již existuje samostatná lekce, kterou si můžete zobrazit ZDE, a proto se jim již zde zaobírat nebudeme.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

VUE

Vue je webový framework, který slouží pro tvoření interaktivních webových aplikací. Alternativou může být např. velmi oblíbený React, Angular, Svelte a stovky dalších. Tyto frameworky se zejména používají pro velké webové aplikace, které potřebují rozdělení do celků, a nelze s nimi pracovat s jednoduchými JS skripty.

Webové aplikace jsou pojem, který na rozdíl od webových stránek, značí, že stránka je interaktivní, používá se jako aplikace a má mnohem větší účel, než jen informační.

Vue se skládá z **komponent**, který reprezentují jeden celek stránky, který je znovupoužitelný. Komponenty se skládají ze šablony, skriptu a stylu, kde jedinou povinnou části je šablona a základní script.

Šablonovací systém Vue je velmi podobný všem jiným frameworkům a hodně si propůjčil od Angularu. Uvnitř šablon můžeme používat výpis z dat šablony pomocí `{{ expr }}` a pomocí direktiv. Direktivy se píší jako atributy a vždy začínají `v-` a nebo `:`. Každá direktiva dělá různé věci, tyto považujme za nejdůležitější:

- `v-bind:attr` / `:attr` - nastaví daný atribut na JS výraz
- `v-for` - cyklus
- `v-if` / `v-else` / `v-else-if` - podmínka
- `v-html` - vykreslí z JS výrazu HTML uvnitř
- `v-on:event` / `@event` - zachycení eventu a zavolání JS výrazu
- `v-model` - připojení inputu (resp. select a nebo textarea) do proměnné
- a další

Vue lze využívat zcela bez jakýchkoliv balíčků (tj. Node.js/npm), stačí nám pouze importovat JS script. Toto je však velmi omezené prostředí a není moc produkčně přívětivé (trvá to, nemáme více souborů, ...) a proto je lepší používat projekty pomocí např. `@vue/cli` či `vite`, které nám dokáže Vue projekt vygenerovat. Projekty poté kompilujeme pomocí příkazu `npm run build` a pracujeme na nich pomocí `npm run serve`.

Komponentám můžeme definovat spoustu vlastností a metod, jedná se např. o:

- `data` - proměnné, které můžeme upravovat
- `props` - neměnitelné proměnné, které získáváme z rodiče
- `components` - seznam komponent použitelných v této komponentě
- `computed` - vypočítané proměnné, které se přepočítají při změně reference
- `methods` - seznam metod, které můžeme volat
- `watch` - funkce pro sledování proměnných, které se zavolají po změně vázané hodnoty
- a další

Data se Vue mezi komponenty předávají pomocí mnoha způsobů. Nejzákladnější jsou props, kterými předáváme data směrem z rodiče do potomka, a události, které zachytíme z potomka v rodiči. Tyto data jsou decentralizovaná a zbytek aplikace k nim nemusí mít přístup. Centralizovaný přístup využívá toho, že data celé aplikace jsou uložené na jednom místě, v případě Pinia, v tzv. store (obchodu) a jakákoliv část aplikace je může číst či měnit.

Kompozice je alternativa k tvoření komponent pomocí Options API, které jsme používali doteď. Jedná se o vlastnosti - věci jako `data`, `computed`, `methods`, `props` a další.

Hlavní nevýhodou Options API je to, že každá komponenta má většinou na starost několik věcí. Další problém Options API je to, že je velmi těžké různé funkcionality kódu a komponent zdědit a upravovat.

Nová náhrada se jmenuje **Composition API**, neboli kompozice, která nám výše uvedené problémy (a důvody) vyřeší. Hlavní rozdíl je ten, že místo definování vlastností se definuje jedna funkce (`setup`), která zpracuje celý nutný kód pro komponentu. Hlavní velká výhody je to, že definice jednotlivých funkcionalit může být u sebe - nemusíme dávat proměnné k sobě, sledovače k sobě a podobně, můžeme si jednotlivé funkcionality k sobě nabalit. Tyto jednotlivé části kódu lze však použít znovu - kód stačí obalit např. v funkci v jiném souboru a z ničeho nic můžeme jednoduchým zavoláním funkce (resp. jejím importování a zavoláním) rozšířit funkcionalitu komponenty. Kompozice dovoluje veškerou funkcionalitu, kterou má Options API.

Options API je dokonce na kompozici postaveno, jen to uživatel nevidí. Vše, co lze pomocí Options API lze tedy i v Composition API.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

HTTP

Hypertext transport protocol (HTTP) je počítačový protokol, který se používá ke komunikaci na internetu. Dříve byl určen pro přenášení hypertextových dokumentů (tj. HTML), nyní se však používá na různé věci s různým obsahem.

HTTP se dělí na požadavky a odpovědi. V klasickém používání požadavky dělá webový prohlížeč (resp. stránka) a odpovídá na ně server. V požadavku definujeme:

- metodu, která určuje jaký typ požadavku děláme
  - GET, POST PUT, PATCH, DELETE, OPTIONS, ...
- cíl požadavku, tedy kam přesně požadavek cílíme
- verzi HTTP protokolu
- hlavičky, které určují metadata
- tělo

V odpovědi dostáváme:

- verze HTTP protokolu
- stavové číslo
  - 100 - 199: informační
  - 200 - 299: úspěch
  - 300 - 399: přesměrování
  - 400 - 499: chyba uživatele
  - 500 - 599: chyba serveru
- stavový text, které dle stavové čísla více specifikují stav
- hlavičky
- tělo

Application Programming Interface (API) je rozhraní, pomocí kterého nějak ovládáme jinou aplikaci, nebo z ní získáváme data. Tyto API mohou být např. i na úrovni HW, vykreslování a podobně, proto mluvíme o Web API, které se zaměřuje na komunikaci aplikací s API, které je vytvořeno pomocí HTTP.

Representational state transfer (REST) je zásada, která nám určuje, jak by se mělo Web API tvořit, definuje například:

- bezstavovost, tedy, že požadavek by měl vždy obsahovat vše, co je potřeba, a že server si nic důležitého mezi požadavky neukládá
- jednotné rozhraní, tedy, že máme endpointy, které jsou konzistentní, vážou se na sebe, ...
- a další

Web API, které dodržuje tyto zásady, je **RESTful**. RESTful API by taktéž mělo být verzované, to znamená, že při jakékoliv změně API se zvýší verze, a původní zůstane nějakou dobu aktivní, aby byly uživatelé schopní přejít na novou verzi.

Endpoint je název pro jednu URL adresu s konkrétní metodou, kterou využívá.

Create Read Update Delete (CRUD) je název pro základní operace, které můžeme dělat s jednotlivými daty.

Data můžeme pomocí HTTP předávat různými formáty, mezi nejznámější patří:

- HTML - předávání webových stránek
- XML - jedná se o "rodiče" HTML ale nemá definované tagy a atributy, data můžeme předávat ve vlastním rozpoložení
- JSON - jedná se o formátovaný JS objekt, který je velmi čitelný a přehledný

XML a HTML mají nevýhodu v tom, že jsou "velmi ukecané" k přenášeným datům.

Cross-origin resource sharing (CORS) je mechanismus, kterým se definuje to, jaké stránky mají k jakým datům na internetu přístup. Nastavení se kontroluje v prohlížeči.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

GIT

Git je jeden z existujících verzovacích nástrojů. Je postaven pro týmovou práci a verzování projektů. Verzování slouží k ukládání jednotlivých verzí, ve kterých se lze vracet, př. hledat věci.

Git má několik důležitých pojmů:

commit - jednotlivá verze, obsahuje mnoho souborů a má jméno. Commit je vždy vázaný alespoň na jeden jiný commit, mimo první commit ve větvi.

větev - obsahuje commity a má název.

Větvě lze slučovat proto, abychom sjednotili funkcionality. Při slučování větví může nastat problém spojení - merge conflict. Ten nastane přesně tehdy, když se v každé ze sloučených větví upraví stejný soubor na stejném místě - git neví, co z nich vybrat. Merge conflict se řeší tím, že se ručně musí upravit soubory tak, aby dávali smysl.

Gitová služba je místo, která nabízí hostování gitu a poskytuje další týmové nástroje - jako issues, projects, actions a další.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

AUTENTIZACE A AUTORIZACE

Autentizace je proces, kdy je uživatel (či jiná entita) přihlášena pomocí nějakého způsobu. Jako ukázku si můžeme představit např. přihlašování jménem a heslem, pomocí třetí služby (např. Googlu), či přihlášení pomocí NFC karty.

Autorizace je proces, kdy je nějaká akce přihlášeného uživatele schválena, např. určení práv, k jakým částem webové aplikace má uživatel přístup.

U autentizace se často potkáme s pojmy šifrování a hashování. **Šifrování** je proces, kdy pomocí nějakého algoritmu převedeme data na nečitelný text, který jde však pomocí stejného či jiného procesu znovu rozeznat (např. klíči). **Hashování** je proces převedení dat do nečitelné podoby, která již zpátky nelze získat (můžeme ale porovnávat, známe-li vstup).

JSON Web Token (JWT) je způsob jak podepsat určitá data tak, aby šlo později zkontrolovat, zda je vydaná entita podepsala. Rozděluje se na tři části:

- hlavička - obsahuje informace o tomto tokenu, např. verze a typ šifrování
- tělo - obsahuje jakákoliv data
- podpis - podpis hlavičky a těla

Jednotlivé části jsou rozdělené `.` a uvnitř mají formát JSON. Navenek jako token jsou "překonvertované" pomocí BASE64, což je zápis v 64 soustavě. V základu dovoluje JWT expiraci tokenu, která se musí však kontrolovat. Expiraci lze vidět v tělu a to v klíči `exp`.

Výhody jsou takové, že je tento způsob velmi jednoduchý a rychlý. Obsah však lze číst a v případě prolomení hesla může kdokoliv podepsat jakýkoliv token. Problémy jsou i s určením toho, zda je token již neplatný.

OAuth2 je způsob přihlašování. Oproti ostatním je velmi standardizovaný a používá se zejména pro velké firmy a aplikace, které dovolují, aby ostatní třetí strany mohli používat účty (a data) na jejich platformě - tj. uživatele přihlašovat. V OA2 je spousta tokenů, které určují jak klient a server komunikují. Základní poznatek je to, že klienti musí být zaregistrovaní (aby jim mohl autorizační server věřit). Uživatel při přihlášení získá povolenku, která se dostane třetí aplikaci, která si požádá o přístupový token. Přístupový token potom používáme k interakci s třetí aplikaci. Dalším tokenem je obnovovací token, který přístupový token umí obnovit.

Hlavní výhodou OAuth2 je právě jeho možnost přihlašování na různých aplikacích (a jejich částech) s jedním systémem. Nevýhodou je to, že je systém velice komplikovaný a těžší na implementaci.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.