06 KNIHOVNY, docs, MDN, ChatGPT, arrow fns

v prvom rade zo #soubory si stiahni `02-funkcie-html-elementy.zip` to je kód, ktorý som písal na hodine. ak chceš, stiahni si `02-[BONUS]-funkcie-chatgpt-if-undefined.zip` to je bonus. kopa ďalšieho kódu. rozbaľ to, otvor vo vscode a preskúmaj `main.js` súbor. je **extrémne okomentovaný.** každý riadok je vysvetlený.

na hodine sme začali **vyrábať vlastné funkcie**. to je vysvetlené v tomto texte https://discord.com/channels/1147117106450681876/1367509004271747183 preštuduj hlavne sekciu o slovíčku `return` a potom sa pusti na domáce úlohy https://discord.com/channels/1147117106450681876/1367484006366580867 funkcie zvyknú byť zvláštny koncept, keď sa učíš programovať. pretože spúšťanie kódu akoby skáče hore dole v súbore medzi riadkami. vysvetlím to, ale ty si skúšaj písať a spúšťať funkcie 😌

UKÁZAL SOM ZOPÁR KNIŽNÍC

ty si môžeš písať vlastné funkcie. ale kopa funkcií už bola napísaných. keď majú spoločnú tému, často sú tie funkcie zabalené do balíčka, ktorému hovoríme knižnica. alebo knihovna. knihovna je kolekcia kódu, často **tematicky zameraná**.

napríklad:

- https://animate.style a https://gsap.com sú **animačné knižnice**
- https://d3js.org a https://www.chartjs.org sú knižnice pre tvorbu **tabuliek a vizualizáciu dát**
- https://github.com/validatorjs/validator.js alebo https://just-validate.dev pre **validáciu údajov***
- https://threejs.org pre prácu s **3D** grafikou
- https://date-fns.org alebo https://day.js.org pre formátovanie **času a dátumu**
- https://sortablejs.github.io/Sortable/ pre **drag-n-drop**
- ...

👉 pričom najdôležitejší odkaz na každom z tých weboch je "Documentation" alebo "Docs"

> (**validácia je overovanie správneho formátu údajov**: napríklad či používateľ skutočne zadal email, či heslo je dostatočnej dĺžky... keď dáš používateľovi možnosť niečo napísať, *musíš automaticky predpokladať*, že to napíše najdementnejším možným spôsobom. doslova všetko spraví nesprávne. validačné knižnice ti pomôžu jednoznačne vymenovať pravidlá pre napríklad každé políčko formulára a potom, ak niečo vyplní nesprávne, dostane chybovú hlášku.)

toto neni objektívne správny oficiálne podpísaný zoznam správnych knižníc 🙂 **toto je len zopár, ktoré mi napadlo ukázať.** existujú toho milióny a klasický javascriptový vtip je, že kým dopíšem túto vetu, vznikne 5 ďalších. pekná a hnusná vec na web svete je, že sa neustále vyvíja. a tým pádom sa mení. popularita knižníc rastie, klesá... tým pádom to nefunguje tak, že sa naučíš a tá ti zostane naveky.

funguje to tak, že ty sa naučíš pár základov a potom **v dokumentácii vždy nájdeš**, čo presne máš písať. programátor viac číta ako píše. tieto konkrétne knižnice sú vedľajšie. dôležitá je dokumentácia.

MDN dokumentácia pre HTML, CSS, JAVASCRIPT

nikto nečaká, že budeš vedieť kód písať z hlavy. programovanie neni memorizácia príkazov. ty sa nemusíš učiť naspamäť názvy funkcii a aké hodnoty očakávajú. jednak VSCode ti dopĺňa kód, dáva ti rady. ale hlavne **všetko** si vieš dohľadať.

👉 **ak hľadáš informácie o jazykoch HTML CSS a JS, tak chceš MDN: https://developer.mozilla.org/**

ak ja hľadám, ako sa v javascripte používa `toFixed`, do google napíšem "mdn toFixed". prípadne "mdn javascript toFixed". ale proste každé vyhľadávanie začínam slovom "mdn". rovnako `repeat` a všetko ostatné. proste ak vieš, že nejaká funkcia existuje a chceš o nej vedieť viac, chceš MDN. často aj keď si vyhľadávaš, že "ako v javascripte spravím XY", často sa dostaneš na MDN.

pričom najdôležitejšia sekcia sú podľa mňa "Examples" https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/repeat#examples

nepotrebuješ ani angličtinu. čisto podľa príkladov (examples) vidíš, čo ten `repeat` robí a ako. v dokumentáciách občas nájdeš aj "Try it" sekciu alebo niečo podobné. kde si môžeš rovno skúšať písať kód a sledovať výsledky. MDN ju má napríklad tu https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed#try_it ukazoval som aj dokumentáciu jazyka Ruby https://try.ruby-lang.org/

pointa je, že dokumentácia je tvoj nový domov 🙂 programátor nemá všetky tie divné zápisy v hlave. programátor vie, že ich vie nájsť.

ChatGPT, Copilot, Claude... AI srandy

ChatGPT (a pod.) môžeš používať ako lepší google. ja som ti ukázal, ako som v MDN z `toFixed` dostanal na `toLocaleString` kde potom by som sa nájsť podstránku pre zoznam lokalít... často proste potrebuješ dokopy informácie z povedzme 5 MDN podstránok.

namiesto manuálneho vyhľadávania, môžeš skúsiť pokecať s ChatGPT. ak mu dáš otázku napr: *"ako v javascripte spravím funkciu, ktoré číslo zmení na české koruny? v minulosti som videl toLocaleString, dá sa to použiť?"*

a možno pridáš zopár doplňujúcich otázok, eventuelne sa spoločne dopracujete k funkcii, ako je táto:

```
function financial(number) {
    number = number.toLocaleString('cs-CZ', {
        style: 'currency',
        currency: 'CZK'
    });

    return number;
}
```

KAŽDÝ PROGRAMÁTOR ROBÍ CHYBY

nonstop. aj preto potrebuješ mať v prehliadači otvorenú konzolu. v konzole budeš vidieť chybové hlášky. vtedy to nefunguje tak, že ty iba rozhodíš rukami a povieš "nefunguje mi to". ty vtedy začneš vyhľadávať, že čo tá konkrétna chybová hláška znamená. a ako by si dala opraviť.

a máš v rukách silu celého internetu. plus chybú sú ďalšia vec, ktorú vieš rozdiskutovať s ChatGPT. alebo Copilot, Grok, whatever. ChatGPT budem používať ako taký všeobecný príklad 🙂

dôležité je chápať, že keď ti kód nefunguje, to je bežná súčasť práce. robota programátora neni sadnúť si a napísať správny kód, ktorý ti svieti v hlave. robota programátora je zistiť *prečo* ten kód furt nefunguje a pokúšať sa ho opraviť.

rovnako, keď budeme inštalovať alebo používať rôzne programy, ak ti robia problémy, rozdebatuj to s AI. na to sú tie srandy veľmi užitočné.

CONST vs LET

premennú v javascript môžeš okrem `let` vytvoriť cez `const`. vtedy je to **konštanta**. konštanta sa nemení. ak robíš hru, `score` je nadizajnové k tomu, že sa bude meniť. to je celá jeho pointa. ale keď robíš web pre firmu, tak názov Disney sa asi nezmení na iný, tak dáš:

```
const companyName = 'Disney';
```

písanie kódu je často o komunikácií. buď s budúcou verziou samého seba alebo s kolegami. tvoj kód bude fungovať prakticky rovnako, či už použiješ `let` alebo `const`. ale ak použiješ `const` je to signál pre kolegov, že hodnota tejto premennej sa nebude meniť.

často sa to používa napríklad pri práci s HTML elementami. keď si do premennej uložíš odkaz na nejaký HTML element, nechceš do tej istej premennej neskôr uložiť číslo 5. chceš tam fixne mať odkaz na ten konkrétny HTML element. tak často v príkladoch na internete zbadáš, že sa používa `const`.

ARROW FUNCTIONS =>

toto je relevantné možno pre pokročilých, ale 2 spôsoby ako napísať tú istú funkciu:

```
// klasika
function pozdrav(meno) {
  alert('Ahoj ' + meno);
}

// arrow function
const pozdrav = (meno) => {
```

```
  alert('Ahoj ' + meno);
}
```

skrátený spôsob, ako v javascripte vytvoriť funkciu. tzv **arrow function**. častejšie sa ale používa v momentoch, kedy funkciu napríklad posielaš do inej funkcie. MDN má tu (v dobe písania textu) príklad s `.map()`: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#try_it

map vie vyrobiť z poľa iné pole podľa funkcie, čo tam pošleš:

```
const materials = ["Hydrogen", "Helium", "Lithium", "Beryllium"];

// arrow function sposob
console.log(
  materials.map((material) => material.length)
);

// "klasicka" funkcia sposob
console.log(
  materials.map(function(material) {
    return material.length;
  })
);
```

vidíš, že arrow function ti dovolí vynechať slová `function` a `return` a zápis je teda kratší, ale výsledok je rovnaký.

znova: ak začínaš, netráp sa tým 😄 v programovaní všetko vieš spraviť viac spôsobmi. nájdi si jeden, ktorý ti je sympatický a používaj ten.