05 NAŠE FUNKCIE, argumenty, return

ak toto čítaš cca vtedy, kedy som to napísal, tak vieš, že hodina 03 ešte nebola 🙂 **čiže toto sú infomácie z budúcnosti!** to si nečakal, že sa tu naučíš cestovať v čase, čo? na tretej hodine začneme s funkciami. ja som popridával kopu úloh do https://discord.com/channels/1147117106450681876/1348937779983421503 a prakticky všetky z nich vyžadujú, aby si vytvárala nové funkcie.

takže môžeš počkať na hodinu, alebo si ich môžeš naštudovať už teraz 👇 ak máš čas a chuť, študuj!

FUNKCIE

pracujeme s dátami. konkrétne hodnoty často dopredu nepoznáme, preto vytvárame **premenné**. hodnoty do nich prídu z databázy, od používateľa, ... často ale v "nesprávnom" tvare. do správneho tvaru ich postupne upravujeme pomocou **funkcií**. programovacie jazyky majú množstvo vstavaných funkcií, ktoré môžeš veselo používať.

videli sme funkciu `alert()`: vyskočí ti okno. videli sme funkciu `.repeat()`, ktorú ak zavoláme na string, zopakuje ho x-krát. **funkcie sú schopnosti, ktoré javascript má**. javascript vie veľa, ale rozhodne nie všetko, čo my budeme potrebovať.

to je v pohode. pretože my javascriptu vieme doprogramovať nové schopnosti! naučiť ho nové veci!

`alert` a `repeat` v ňom už existujú. ale neexistuje tam `pozdrav()`.

NAŠE VLASTNÉ FUNKCIE

kľúčovým slovíčkom `function` vytvoríš do javascriptu novú funkciu:

```
function pozdrav() {
    alert('Ahoj Vašo');
}
```

a teraz už v javascripte `pozdrav()` existuje. keď napíšeš tieto riadky, naučil/a si javascript sa komplet novú vec! zatiaľ sa ale nič nedeje. nová funkcia pozdrav síce *existuje*, ale ešte si jej nepovedala, aby sa *spustila*. rovnako ako ten `.repeat()`. on v javascripte je, ale nič nerobí, dokým si to ako programátor nevyžiadaš.

tvoj nový `pozdrav` teraz môžeš brať (a používať) tak isto, ako keby to bol napríklad `alert`, čiže:

```
pozdrav();
```

a bum, zrazu na teba vyskočí okno s textom "Ahoj Vašo"!

keď si zo #soubory stiahneš `js-academy-template.zip` tak v `main.js` nájdeš takýto komentár:

```
// ------------------------
// ---- MAIN PROGRAM
```

nad tento komentár si budeš pridávať nové funkcie. tento komentár rozdeľuje súbor na hornú a dolnú časť. v hornej časti pridávaš funkcie, v dolnej časti ich používaš v tvojom programe.

ARGUMENTY

`"vašo".toUpperCase()` zmení text "vašo" na "VAŠO". `"vašo".repeat()` nespraví nič.

`repeat` má za úlohu text zopakovať x-krát. ale aby to fungovalo, ty potrebuješ dodať to x. koľkokrát sa má "vašo" zopakovať? `"vašo".repeat(23)` ahaaaa!! 23x! to číslo 23 je tzv. **argument**.

`.toFixed()` vie zoseknúť číslo na počet desatinných miest. ale ten počet tam musíš poslať! čiže niektoré funkcie fungujú iba vtedy, ak do nich pošleš hodnotu.

```
function pozdravMeno(meno) {
    console.log('Ahoj ' + meno);
}

pozdravMeno('St. Vincent');  // do konzole vypise Ahoj St. Vincent
pozdravMeno('David Byrne');  // do konzole vypise Ahoj David Byrne
```

táto funkcia tzv. **prijíma argument**. konkrétne čaká, že do nej pošleš string. ideálne niekoho meno. vo vnútri funkcie vznikne premenná s názvom `meno`, s ktorou vieš vo funkcii pracovať. napríklad, v tomto prípade, vypísať `meno` do konzole.

podobne, ako v kóde môžeš napísať `console.log('Ahoj');` teraz môžeš napísať aj `pozdravMeno('David Byrne');`. v tom momento program "skočí do tela" funkcie pozdravMeno a spustia sa riadky medzi `{ }` takže v tomto prípade sa spustí riadok `console.log('Ahoj ' + meno);` kde v premennej `meno` bude hodnota "David Byrne".

FUNKCIA JE SAMOSTATNÁ JEDNOTKA, KAŽDÁ JE VLASTNÝ VESMÍR

v zmysle, že ak vo vnútri funkcie vytvoríš premennú, tá premenná existuje iba tam. iba vo funkcii. ak v kóde zavolám `pozdravMeno('David Byrne')`, tak vo vnútri funkcie `pozdravMeno` automaticky vznikne nová premenná s názvom `meno` a jej hodnota sa nastaví na 'David Byrne'. podobne, ako keby som tam manuálne napísal `let meno = 'David Byrne';`

táto premenná `meno` existuje iba vo vnútri funkcie `pozdravMeno`. iba medzi tými `{ }` zátvorkami, ktoré označujú začiatok a koniec funkcie. všetky riadky kódu medzi tými `{ }` zátvorkami patria do funkcie. ak

zavoláš funkciu, spustia sa jeden po druhom všetky riadky medzi `{` a `}`.

pridajme ďalšiu:

```
function financial(number) {
    number = number.toFixed(2);
    number = number.replace('.', ',');
    number = number + ' Kč';

    console.log('Čakám, že mi dáš presne ' + number);
}
```

teraz som javascript naučil novú vec. okrem `alert`, `repeat` a `pozdrav` v javascript odteraz existuje aj `financial`. ak v kóde zavolám `financial(44.453262);` spustia sa jeden po druhom všetky riadky kódu medzi `{` a `}`.

vo funkcii automaticky vznikne nová premenná `number` a uloží sa do nej hodnota `44.453262`.

potom prvý riadok funkcie (čiže `number = number.toFixed(2);`) odsekne nadbytočné desatinné miesta. druhý riadok zmení všetky bodky na čiarky. tretí riadok pridá text " Kč". potom je tam prázdny riadok. ten nespraví nič. ten je tam len pre moje oči. a pre pokoj mojej duše. posledný riadok vypíše do konzole text "Čakám, že mi dáš presne 44,45 Kč".

takže my sme dostali chladné, sterilné, počítačovo exaktné číslo `44.453262`. a naša úloha bola zmeniť ho na peknú, ľudskú vetu. postupne sme to číslo upravovali, kúsok po kúsku, jak sochár keď robí s kameňom, až sme z neho vytvorili to, čo potrebujeme.

SPOLOČNÁ MISIA

každá funkcia je **skupina riadkov, ktoré majú spoločnú misiu.** pozri na funkciu `financial`. odmysli si ten prázdny riadok. tá funkcia má 4 riadky, ktoré spoločne majú jasnú úlohu: zmeniť číslo na vetu o tom, koľko českých korún mi máš dať.

programovanie je často organický proces. kód vznikná postupne. ty pod seba píšeš riadky, ktoré upravia dáta do správej podoby. povedzme, že na to potrebuješ 6 riadkov kódu. neskôr si všimneš, že znova potrebuješ tých istých 6 riadkov použiť na iné dáta. **to je indíca, že je správny čas vytvoriť funkciu!** ak vo svojom kóde spozoruješ x riadkov, ktoré majú spoločnú úlohu, vyrob z nich funkciu. funkcia ti uľahčí život, pretože zrazu v javascripte pridubol nový nástroj, ktorý môžeš používať. a to ti pomôže udržať kód čistý, pretože namiesto 6 riadkov ti teraz stačí napísať jeden.

RETURN

pozri sa na našu funkciu `financial()` tam vyššie. ona spracuje číslo a vo výsledku vždy do konzole vypíše vetu: *"'Čakám, že mi dáš presne XY"*. jej úloha má byť naformátovať obyčajné číslo na finančný obnos. lenže keby robím eshop, tak ten finačný obnos chcem zobraziť na mnohých miestach. v košíke, v objednávke, na

faktúre... a možno ho chcem zobraziť v rôznych jazykoch. naša funkcia vždy vyprodukuje vetu v slovenčine. ale bola by užitočnejšia, kedy dokázala vyrobiť len to naformátované číslo! ten obnos samotný! pretože *to* by som potom mohol použiť kdekoľvek a ako len chcem. s tým nám pomôže `return`.

absolútna väčšina funkcií (ktoré budeš používať alebo sama vytvoríš), bude končiť príkazom `return`. napíšeš slovo `return` a potom pridáš hodnotu, ktorá sa z funkcie takzvane "vráti".

predstavme si funkciu `plus()`, do ktorej vieš poslať 2 čísla a jej úloha je zistiť ich súčet. proste vyrábame veľmi zlé plus na kalkulačke:

```
function plus(a, b) {
    let sucet = a + b;
    return sucet; // <---
}
```

`return` znamená *"vráť hodnotu do programu na miesto, kde si ju zavolal."* čiže ak v mojom kóde mám zápis:

```
plus(10, 5); // <--- ako keby som sem napisal 15
```

takže aby som výsledok aj videl:

```
console.log( plus(10, 5) );
```

kdekoľvek v kóde mám zápis `sucet(10, 5)` je to ako keby som tam napísal číslo 15.

```
// mozem ho poslat do alertu
alert( plus(10, 5) );

// mozem ho pouzit vo vete
console.log( "sucet krasnych cisel je: " + plus(10, 5) );

// mozem ho vlozit do HTML kodu stranky
document.querySelector('body').innerHTML = plus(10, 5);

// kdekolvek napisem sucet(10, 5), tam sa ocitne vysledok funkcie, cize 15
```

S RETURN JE FUNKCIA VŠEOBECNEJŠIA

ešte raz: **ak by sme return nepoužili** a funkcia by končila console.logom:

```
function plus(a, b) {
    let sucet = a + b;
    console.log('súčet je ' + sucet); // <---
}
```

tak **vždy**, keď ju zavolám, dostanem výpis "súčet je [číslo]".

čo ak potrebujem na rôznych miestach iný výpis? čo ak potrebujem výpis v rôznych jazykoch?

ak funkcia cez `return sucet;` vráti číslo:

```
function plus(a, b) {
    let sucet = a + b;
    return sucet; // <---
}

console.log( 'súčet je ' + plus(10, 5) );
console.log( 'die wunderschöne Zahl ist: ' + plus(10, 5) );
```

viem to číslo použiť **jak len potrebujem!**

takže javascript vie veľa vecí, ale nevie všetko. javascript má v sebe kopu funkcií, ktoré riešia všeobecné problémy. ale tvoje úlohy nebudú všeobecné. tvoj úlohy budú konkrétne. a ty si budeš vytvárať vlastné funkcie, ktoré ti pomôžu tvoje konkrétne úlohy riešiť. a absolútna väčšina funkcií, ktoré budeš vytvárať, bude končiť príkazom `return`.

funkcia je mašina, do ktorej nahádžeš údaje, ona ich spracuje a vráti ti výsledok. s výsledkom potom ďalej v kóde pracuješ. funkcie sú možno najdôležitejší koncept v programovaní.

ROZDIEL MEDZI FUNKCIAMI S A BEZ RETURN

skús si spustiť takýto kód:

```
function plus(a, b) {
    let sucet = a + b;
}

console.log( plus( 10, 5 ) );
```

v tomto prípade tam chýba ten return. ak spustíš tento kód, do konzole sa nevypíše nič.

ale keby spustíš takúto funkciu:

```
function plus(a, b) {
    let sucet = a + b;
    return sucet;
}

console.log( plus( 10, 5 ) );
```

táto v sebe má return. teraz v konzole uvidíš číslo 15;

ak vo funkcii je return, tak tam, kde ty v kóde napíšeš `plus( 10, 5 )`, sa ocitne výsledok funkcie. v tomto prípade číslo 15. je to jak keby si napísal `console.log( 15 );` bez toho slovíčka return sa funkcia síce spustí, ale žiaden výsledok sa takzvane nevráti do toho console.logu.

existujú funkcie, ktorých úloha je iba niečo spraviť a tam to celé skončí. napríklad naša funkcia `createNewCard`, ona vytvorí nový element a tam to celé skončí. **takáto funkcia nepotrebuje slovíčko return.**

ale existuje druhý typ funkcie: taká, ktorá vyrobí nejaký výsledok, s ktorým ty potom môžeš ďalej v kóde pracovať. taká funkcia, do ktorej nahádžeš dáta a ona ti ich vráti upravené. napríklad naša funkcia `financial()`. do nej hodíš číslo a ona ti vráti text čo vyzerá ako české koruny. **takáto funkcia potrebuje slovíčko return.**

čiže ak chceš funkciu, ktorá len spraví svoju robotu, ale ty už nepotrebuješ ďalej pracovať s jej výsledkom, nemusí tam byť return. ak ale chceš funkciu, ktorá vyrobí výsledok, s ktorým by si rád ďalej pracoval, tá potrebuje slovíčko return.

❤️