Úvod do JavaScriptu

JS je programovací jazyk zodpovedajúci štandardu ECMAScript (preto sa využíva aj označenie ES), ktorý možno spolu s HTML (HyperText Markup Language) a CSS (Cascading Style Sheets) zaradiť k základným webovým technológiám na strane klienta.

JS a HTML alebo ako vložiť JavaScript kód do HTML dokumentu

Na vloženie (injekciu) JS kódu do HTML dokumentu využívame tag <script> ten možno umiestniť v rámci HTML dokumentu ľubovoľný počet krát v časti <body>, <head>, prípadne v oboch. *The best practice: vložiť tag <script> tesne pred uzatváracím tagom </body>*. K samotnej injekcii pritom možno pristúpiť 2 spôsobmi:

- vložiť JS kód priamo medzi tágy <script> </script> v rámci HTML

```
<script type="text/javascript">
    //JS code is included directly in HTML

    console.log("Hello world")

</script>
```

- prostredníctvom odkazu na externý súbor, využitím src atribútu tagu <script>

```
<script type="text/javascript" src="path-to-javascript-file.js"></script>
```

  Výhody tohto prístupu:
  - separácia HTML a JS kódu
  - čitateľnejší JS kód ako aj HTML
  - externé JavaScript súbory ukladajú prehliadače do vyrovnávacej pamäte (caching) a môžu urýchliť načítanie stránky
  - možnosť využiť defer a async atribúty tágu <script>

Super článok https://levelup.gitconnected.com/all-about-script-87fea475b976

Premenné v JS

Pre deklaráciu premenných v JS možno využiť dve kľúčové slová: var a let

Var	Let
• Pred inicializáciou je typu undefined • Rozsah platnosti premennej : funkcia • Starší spôsob deklarácie premennej (nepoužívať)	• Pred inicializáciou je typu undefined • Rozsah platnosti premennej : blok • Zavedená v ES6

JS je dynamicky typovaný jazyk premenne teda môžu v rámci kódu nadobúdať v rámci rôznych častí rôzne typy.

Konštanty v JS

Deklarujú sa pomocou kľúčového slova const, rozdiel oproti premenným spočíva v tom, že musia byť inicializované hneď pri deklarácii a ich hodnota sa už následne nemení. *The best practice: všetko to, čo sa nemení, deklaruj ako konštantu, premenné používaj len ak je tam potenciál zmeny hodnoty!*

Template string alebo šablónový reťazec

Je zjednodušený spôsob zápisu viacriadkových reťazcov a kombinovania reťazcov a hodnôt výrazov bez nutnosti využívať operátor spájania reťazcov (+) či metaznaky ako napríklad: \n, \t..., ktorý bol do

JS zavedený vo verzii ES6. Celý reťazec je uzatvorený do ` ` (backquotes) pre vloženie premennej sa využíva nasledujúca syntax ${<variable_name>}

ES5

```
const myName='Frank'
const newStr= '\nName:\t'+ myName
```

ES6

```
const myName='Frank'
const newStr=`
Name:    ${myName}`
```

Cykly

3 základné:

- for – cyklus s 3 voliteľnými výrazmi
  - expression1 sa vykoná raz na začiatku ešte pred overením podmienky
  - condition definuje podmienku pre vykonanie bloku kódu
  - expression2 sa vykoná vždy po vykonaní bloku kódu

```
for (expression1;condition;expression2){
    //code block to be executed
}
```

- while – cyklus opakuje blok kódu pokiaľ je podmienka pravdivá

```
while(condition){
    //code block to be executed
}
```

- do while – je variant while cyklu, ktorý vykoná blok kódu pred overením či je podmienka pravdivá a potom opakuje tento blok kódu pokiaľ je podmienka pravdivá

```
do{
    //code block to be executed
}while(condition)
```

Podmienky

3 základné typy:

- if

```
if(condition){
    //code block to be executed if condition is true
}
```

- if else

```
if(condition){
    //code block to be executed if condition is true
}else{
    //code block to be executed if condition is false
}
```

- if else if

```
if(condition1){
    //code block to be executed if condition1 is true
}else if(condition2){
    //code block to be executed if condition1 is false and condition2 is true
}else{
    //code block to be executed if condition1 and condition2 are false
}
```

- ternárny operátor – alternatívny zápis if else pri priradeniach alebo návratových hodnotách funkcii

```
condition ? expression1 : expression2
```

if else

```
let status
if (tasks.length > 5){
    status = 'bussy'
}else{
    status = 'not bussy'
}


if (tasks.length > 5){
    return 'bussy'
}else{
    return 'not bussy'
}
```

Ternary operator

```
let status = tasks.length > 5 ? 'bussy' : 'not bussy'

return tasks.length > 5 ? 'bussy' : 'not bussy'
```

Funkcie

Funkcie sú hlavnými „stavebnými kameňmi“ programu. Umožňujú volať kód mnohokrát bez opakovania. V JS rozoznávame 2 základne vzory funkcii:

- pomenované funkcie – majú meno
- anonymné funkcie – nemajú meno väčšinou sa vytvárajú v prípade keď je treba odovzdať funkciu ako argument inej funkcii

Pri vytváraní funkcií máme hneď niekoľko spôsobov, ktoré vedú k výsledkom s rôznymi vlastnosťami:

- *funkčná deklarácia (function declaration)* – ide o vytvorenie funkcie pomocou kľúčového slova function za ktorým nasleduje názov funkcie (takto teda ide vytvoriť len pomenovanú funkciu) a v zátvorke uvedené názvy parametrov oddelené čiarkami (môže ich byť aj nula v takom prípade uvedieme len prázdne zátvorky())
  Poznámka: V jazyku C by sme to, čo tu nazývame deklaráciou nazvali definícia funkcie. No v JS je zaužívaný pojem deklarácia funkcie

```
function function_name (parameter1, parameter2, ...){
    //code block to be executed when function is called
}
```

- *funkčný výraz (function expression)* – keďže funkcie sú v taktiež objekty možno ich priradzovať do premenných a konštánt pričom na tomto princípe funguje funkčný výraz. Ide o vytvorenie funkcie využívajúce jej priradenie do premennej resp. konštanty k nej sa potom

možno správať ako k názvu funkcie. Takto možno vytvoriť ako pomenovanú funkciu tak aj anonymnú funkciu.

- Funkčný výraz s pomenovanou funkciou

```
const myFunction = function function_name (parameter1, parameter2, ... ){
    //code block to be executed when function is called
}
```

- Funkčný výraz s anonymnou funkciou

```
const myFunction = function (parameter1, parameter2,    ){
    //code block to be executed when function is called
}
```

Parametre vs. Agumenty funkcie

Parameter je premenná v rámci funkčnej definície resp. funkčného výrazu. Je to zástupný symbol, a preto nemá konkrétnu hodnotu. Argument je skutočná hodnota odovzdaná počas vyvolania funkcie preto sa môže označovať aj ako skutočný parameter.

```
function function_name(parameter1, parameter2, ...){
    //code block to be executed when function is called
}

function_name(argument1, argument2, ...)
```

Návratová hodnota funkcie alebo return

Funkcia bez explicitne uvedenej návratovej hodnoty pomocou kľúčového slova return vracia po vykonaní svojho tela defaultnú návratovú hodnotu undefined. V prípade ak použijeme v rámci tela funkcie kľúčové slovo return ukončí sa vykonávanie tela funkcie a funkcia vráti hodnotu výrazu nasledujúceho za kľúčovým slovom return.

Vyššie uvedené možno vidieť pri nasledujúcej práci s konzolou.

```
> function myFunction (){}
< undefined
> myFunction()
< undefined
> function hello (){return "Hello world"}
< undefined
> hello()
< 'Hello world'
```

Defaultné hodnoty parametrov

Defaultná hodnota parametrov sa využíva v prípade ak pri vyvolaní funkcie nie je zadaný argument respektíve ak je ako argument zadaná hodnota undefined. Pri vykonávaní tela funkcie sa potom v prípade parametra, ktorému nezodpovedá žiadny argument resp. argument undefined využíva jeho defaultná hodnota.

Parametre funkcie bez explicitne uvedenej (nami uvedenej) defaultnej hodnoty majú implicitne nastavenú defaultnú hodnotu na undefined.

Defaultnú hodnotu parametra možno nastaviť v rámci deklarácie funkcie, a to tak, že za názvom parametra uvedenie = defaultValue.

Vyššie uvedené možno vidieť pri nasledujúcej práci s konzolou.

```
> function myFunction (name='NAME', surname){
      console.log(`Name is: ${name} and surname is: ${surname}`)
  }
< undefined
> myFunction('Samo','Novotný')
  Name is: Samo and surname is: Novotný
< undefined
> myFunction('Samo')
  Name is: Samo and surname is: undefined
< undefined
> myFunction(undefined,'Novotný')
  Name is: NAME and surname is: Novotný
```

Arrow function

Ide o zjednodušenú notáciu anonymnej funkcie s nasledujúcou syntaxou:

Vynecháva sa kľúčové slovo function, parametre sú uvedené v zátvorke oddelené čiarkami, v prípade ak má funkcia len jeden parameter zátvorky možno vynechať. Telo funkcie je od parametrov oddelené => a uzavreté do {}, v prípade ak telo možno siahnuť len na príkaz return zátvorky spolu s kľúčovým slovom return možno vynechať a hneď uviesť návratovú hodnotu.

Normálna notácia

```
function (a,b){
    console.log(a,b)
    return a*b
}
```

```
function (a){
    return a*a
}
```

Arrow notation

```
(a,b) => {
    console.log(a,b)
    return a*b
}
```

```
a => a*a
```

JS Objekty

Malá odbočka k dátovým typom

Hodnota v JavaScripte je vždy určitého typu napríklad string alebo number. V JavaScripte je osem základných dátových typov.

Sedem primitívnych (pretože ich hodnoty obsahujú len jedinú vec) dátových typov:

- number – reálne čísla
- bigint - celé číslo s ľubovoľnou presnosťou
- string – znakové reťazce
- boolean – logický typ pre true a false
- null - obsahuje jediný prvok, null, s významom „žiadna hodnota“
- undefined - obsahuje jediný prvok, undefined, s významom „hodnota neinicializovanej premennej“
- symbol - od ES6, pre jedinečné nemenné údaje

Jeden neprimitívny dátový typ

- object
    - function – funkcia
    - array – pole
    - Date , Math , RegExp – príklady zabudovaných objektov
    - Number, String, Boolean – objeky zaobaľujúce primitívne typy

JS object literal

Objekty sa používajú na ukladanie kolekcií dvojíc kľúč: hodnota (key: value pairs), ktoré nazývame vlastnosti objektu (properties). Typy hodnôt vlastnosti môžu byť primitívne alebo objekty. V prípade ak ide o typ function nazývame tieto vlastnosti metódy.

Metóda je teda funkcia, ktorá je viazaná na objekt (asociovaná s objektom).

Deklarácia objektu

Deklarácia objektu ma nasledujúcu syntax:

```
let obj = {
    property1: propertyValue1,
    property2: propertyValue2,
    ...
    method1: function (){
    },
    method2: ()=>{},
    ...
}
```

Pristupovanie k vlastnostiam objektu

Pri pristupovaní k vlastnostiam (teda aj metódam) objektu máme dve možnosti, a to:

- využitím tzv. bodkovej notácie (dot notation)
  ```
  obj.property1
  obj.method1()
  ```
- využitím tzv. zátvorkovej notácie (bracket notation)
  ```
  obj['property2']
  obj['method2']()
  ```

Dekonštrukcia objektu

Dekonštrukcia objektu v podstate taktiež slúži na prístup k jeho vlastnostia (teda aj metódam) s tým rozdielom, že vytvárame nové premenné resp. konštanty, ktoré inicializujeme na hodnoty im zodpovedajúcich vlastnosti objektu podľa nasledujúcej syntaxe.

- Názov novej premennej resp. konštanty musí byť taký istý ako názov vlastnosti objektu, ktorej hodnotou ju chcem inicializovať
  ```
  const {propert1} = obj
  ```
- V prípade ak chcem aby bol názov novej premennej resp. konštanty iný ako je názov vlastnosti objektu, ktorej hodnotou ju chcem inicializovať použijem nasledujúcu notáciu
  ```
  const {propert1: new_name} = obj
  ```
- Takto možno vyberať ľubovoľné množstvo vlastností objektu
  ```
  const {propert1: new_name, propert2, ...} = obj
  ```

Polia

V JavaScripte polia nie sú primitívne, ale sú to objekty Array. Keďže ide o objekty polia majú svoje vlastnosti (teda aj metódy). Medzi základné vlastnosti poľa patrí length (dĺžka poľa) a medzi základné metódy (map, filter, sort,...).

- vytvorenie prázdneho poľa
  ```
  let arr = []
  ```
- vytvorenie poľa s počiatočnými prvkami
  ```
  let arr = [7,3,4,1,5]
  ```
- k jednotlivým prvkom poľa možno pristupovať pomocou indexu (pole je indexované od 0)
  ```
  arr[index]
  ```
- metóda map – slúži na mapovanie prvkov poľa na prvky nového poľa. Má jeden parameter a to funkciu, ktorá zabezpečuje mapovanie prvkov. Vracia teda nové pole, ktoré vytvorí volaním funkcie pre každý prvok poľa.
  ```
  arr.map(item=>item*2)
  ```
- metóda filter – slúži na vytvorenie nového poľa ktoré vzniklo filtrovaním poľa nad ktorým bola táto metóda volaná. Má jeden parameter a to funkciu (predikát), ktorá zabezpečuje filtrovanie prvkov. Vracia teda nové pole naplnené prvkami, ktoré prejdú testom poskytovaným funkciou (funkcia vráti pre daný prvok hodnotu true)
  ```
  arr.filter(item=>item>=5)
  ```
- metóda sort – slúži na vytvorenie nového poľa ktoré vzniklo zoradením prvkov poľa, nad ktorým bola táto metóda volaná. Má jeden voliteľný parameter, a to funkciu, ktorá definuje poradie zoradenia. Ak sa vynechá, prvky poľa sa skonvertujú na reťazce a potom sa zoradia podľa Unicode hodnoty každého znaku. Táto funkcia musí mať dva parametre a, b (reprezentujúce dva prvky poľa ktoré porovnávame) a návratová hodnota určuje ako tieto dva prvky usporiadať:
  - ak je == 0 ponechať poradie
  - ak je > 0 usporiadať a pre b
  - ak je < 0 usporiadať a za b

  ```
  arr.sort((a,b)=>a-b)
  ```