**Js – podstawy**

**Typy prymitywne:**

**number** – do wszystkich możliwych liczb (3.8;50)

**string** – **true** || **false**

boolean

**null** – istnieje, ale celowo dostaje wartość ***pustą***

**undefined** – kiedy coś nie istnieje

***komentarze****,* ***operacje******arytmetyczne*** *tak samo jak w każdym innym języku*

**//** - komentarz  
**\*\*** - potęga. **(2 \*\* 4 == 2^4)**

**NaN** – reprezentuje elementy nieobliczalne, ale będące teoretycznie liczbą

**Zmienne:**

**let** {name} = {value}; - typowe zmienne

**const** {name} = {value}; - zmienne stałe

**var** {name} = {value}; - stary sposób zapisywania ***let***

**Ważne:** Można zmieniać typ zmiennej

**String:**

**let temp = „{}” LUB ‘{}’**

**„temp”[index from 0 to length-1]** ­-> “temp”[2] = “m”. Readonly

**{string}.length** – zwraca ilość znaków

**Funkcje:   
{string}.toUpperCase()** – Podniesienie liter do CAPS   
**{string}.toLowerCase()** – Obniżenie wszystkich liter

**{string}.trim()** – przycięcie pustych znaków na początku i końcu wyrazu

**{string}.indexOf({element})** – podanie pozycji elementu. Jeśli nie ma to ***-1***

**{string}.slice( {begin\_index} )** – tworzy substring od tego elementu

**{string}.slice( {begin\_index}, {end\_index})** – tworzy substring od początkowego elementu do ***ostatniego - 1***

**{string}.replace( {word1}, {word2} )** – zamienia **{word1}** na **{word2},**

**Ważne:** Można zbierać w zespoły różne funkcje ***{string}.trim().toUpperCase();***

**`tekst ${*zmienne/obliczenia*} tekst`** - przy użyciu ` można dodać ${} w którym można zapisywać elementy kodu, które zostaną wliczone w string’a

**Math:**

**Math.{**funkcje**}** – używane do dostawania się do funkcji. Przykładowe funkcje:   
**.PI** – liczba PI   
**.floor( {nr} )** – ucięcie przecinka z liczby

**.ceil( {nr})** – zaokrąglenie w górę

**.random()** – daje losową liczbę między 0, a 1

**Decision making**

**Operatory porównania:**

Większość standardowa. Oprócz:   
**===** - sprawdza typ. Działa podobnie jak ==

**!==** - sprawdza typ. Działa podobnie jak !=

**Uruchamianie skryptów:**

**<script src=”{ścieżka}”></>** - jak załadować skrypt na dole **body**

**Przydatne komendy:  
console.log( {argument1},{argument2} )** – wypisuje argumenty **console.error**

**console.warn  
alert( {argument} )** – tworzy okienko dla urzytkownika z tekstem

**prompt( {argument} )** – tworzy okienko z możliwością wpisania tekstu przez użytkownika

**let input = prompt(„temp”)**

**parseInt( {intiger} )** – zamienia ***string*** na ***int***

**Contidionals:**

**if( {warunek} )** – standardowy if

**else if( {warunek} )** – standardowy else if

**else** – standardowy else

**&& || !**

**Switch:**

Tak samo jak normalnie

**Tablice**

**let arr = [ {argument1}, {argument2} ];** - typowy sposób zapisu. Może być puste. Mogą być różne typy

**arr.length** – ilość elementów

**arr[ {index number} ]** – wyciąganie elementu. Można dodawać elementy do indexów niestworzonych. Pomiędzy powstają wtedy ***empty x {nr}***

**Metody tablic:   
arr.push( {value} )** – używane do dowania wartości do tablicy

**arr.pop()** – używane do wyrzucania ostatniego elementu

**arr.shift()** – używane do usuwania elementu z początku tablicy

**arr.unshift( {value} )** – używane do wrzucania pierwszego elementu i przesunięcia pozostałych

**let temp = arr1.concat(arr2)** – łączenie tablic ze sobą

**arr1.include( {value} )** – sprawdza czy element istnieje w tablicy   
**arr.indefOf( {value} )** - podanie pozycji elementu. Jeśli nie ma to ***-1***

**arr.reverse()** – odwraca tablicę

**arr.slice( {start}, {end} )** – zwraca tablicę od ***start*** do ***end***. **end** opcjonalne   
**arr.splice( {start}, {how many delete}, {what to add} )** – zamienia oryginalną tablicę. **{what to add}** może nie zostać podane. Jeśli **{how many delete}** 0 to wtedy nic się nie usunie

**arr.sort()** – sortuje bazowo konwertując wszystko na ASCI i tą wartość od najmniejszej

**Some&Every:**

**{arr}.every( func )** – zwraca ***true*** jeśli każdy element spełnia funkcje

**{arr}.some()** – zwraca ***true*** jeśli przynajmniej jeden element spełnia funkcje

**Reduce:   
{arr}.reduce( ( {storage}, {current} ) => { //what to do })**

**arr.reduce( (stor, curr) => {return stor + curr;} )**

**Additional:   
let arr1 = arr2;** - nie kopiuje zawartości tablicy. ***arr1*** staje się adresem do ***arr2***.

**const arr = [ {values} ]** – można dodawać wartości do tablicy, ale nie można na niej tworzyć nowych elementów

**Literały obiektowe**

**Jak stworzyć?**

**const {name} = {**

**{prop name1} : {value1},**

**{prop name2} : {value2}**

**};**

**Jak dostać się do elementów?**

**{name}[„property”] = { value }** – jak do tablicy. Można wyszukiwać poprzez zawartość zmiennej jako ***„property”***.  
**{name}.property = { value }** – normalnie

**const arr = [ {object1}, {object2} ]** – w tablicach możemy dawać obiekty

**Object.keys( {nameOfObject} )** – zwraca tablicę z nazwami properties

**Object.values( {nameOfObject} )** – zwraca tablicę z wartościami

**Object.entries( {nameOfObject} )** – zwraca tablicę dwuwymiarową z kluczami i wartościami

**Pętle**

**for(let i = 0; i < 10; i++)** – typowy for  
**while( {bool expresion} )** – typowy while

**for(let var of {arr})** – typowy foreach

**for(let var in {arr})** – foreach dla obiektów. Zwraca nazwę property (**nie wartość**)

**Funkcje – podstawy**

**Deklaracja funkcji:**

**function {name} ( {args} )**

**{...}**

\*\*\*\*

function name (arg1,arg2){

//

}

**Funkcje – rozszerzone**

**Deklaracja funkcji:**

**let {name} = function ( {args} ) {...}**

**Funkcje wyższego rządu:**Funkcje przyjmujące funkcje jako argumenty, lub też je zwracają (tj. funkcje)

**function {name} ( func )**

**{ func(); }**

**Tworzenie metod:**const name = {

func : function( {args} ) {

return ...;

}

}

**Lub**

cont name = {

name( {args} ) {

return ...;

}

}

**Argumenty mogą mieć wartość na wypadek nie podania takowej:**function name(a, b=3) { return a \* b } – dla name(2) zwróci 6. Istotna kolejność argumentów, wartościowane zmienne na końcu

**Rest:**function name(...arr) { //do something } – używane do zbierania nieskonczenie wielu argumentów do podanej tablicy

**This:**Używane do dostawania się do elementów w obiekcie poza funkcją.

**Try&Catch:**try{

// Test doing it

}

catch (e) {

// What to do if error

}

**Callbacks & array methods   
{arr}.forEach( {function name} )** – taki śmieszny w sumie forEach

**{arr}.map( {function name} )** – działa tak samo jak forEach, ale zwraca tablicę z nowymi wartościami

**Arrow functions:**Krótszy sposób zapisywania funkcji:  
const name = ( {args} ) => {

// function indsides

}

Można tworzyć krótko zapisane funkcje kiedy w jednej lini można zapisać operacje. Wtedy nie trzeba definiować return’a  
const name = ( x ) => (

x\*x;

)

**setTimeout:**

**setTimeout( {co ma zostać wykonane}, {jak długo ma czekać} )** – używane do odczekania przed wykonaniem działania. Działa asynchronicznie   
**setInterval( {co}, {ile czekać} )** – wykonuje działanie w interwałach czasowych

**Filter:**

**{arr}.filter( function )** – zwraca tablicę z elementami które spełniają funkcję   
**temp.filter( x => { return x === 4 } );**

**Spread:   
...arr** – rozbija tablice na pojedyncze elementy

**...object** – rozbija obiekt na pojedyncze elementy do kopiowania obiektów