W JS aby wyświetlić wartość variable używa się nastepujacej funkcji:

* Console.log()

Aby stworzyć variable stosuje się:

* Let variable = ...

Możemy stworzyć również variable, którego wartość nie będzie mogła ulec zmianie, stosując Const.

* Const PI = 3,14

Jesli chcemy zrobić komentarz na wiele lini to stosuje się:

/\* tutaj wpisz tekst \*/

Aby sprawdzić typ variable można to zrobić przy pomocy typeof:

Console.log(typeof variable)

Istotna strona gdzie możemy sprawdzić kolejność egzekowwania operatorów:

Mdn operator precedense >

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_precedence>

Aby dodać do anszego string dynamicznie wartości wykorzystujemy do tego ``:

Let test = 8

Console.log(`This is ${test}`)

Dodatkowo tworzenie stringów przy pomocy `` pozwala na łatw tworzenie nowych linijek tekstu.

`test

test`

Warunek if else, stynatx: if (warunek typu coś > od czegoś) { baza, która zadziała jesli warunek zostął spełniony} else { }

Istnieje również else if block aka elif z pythona:

If (michal) {

} else if (ada) {

} else {}

Jeśli warunek if ma tylko jedną część to nie musimy tworzyć {} np.:

If (age === 18) console.log(`This is true`)

Jeśli chcemy manualnie zmienić typ wartości aka type conversion:

Aby uzyskać string: String(x)

Aby uzyskać liczbę: Number(x)

Javascript automatycznie zamienia wartości aka type cohersion np:

`23` - `10` - 3 da nam liczbę 10, ale jesli zrobimy: `23` + `10` + 3 to dostaniemy string 2313

Truthy and Falsy values

Istnieje 5 nieprawdziwych wartości: 0, ‘’, undefined, null, NaN (not a number).

W przypadku gdybyśmy chcieli sprawdzić przy pomocy warunku if czy variable jest True, a posiadałby jedną z powyższych wartości to dsotalibyśmy False.

**Equality Operators == i ===.**

Potrójny === nazywa się strict equality operator i spełniony ejst tylko jeśli obywdwie przyrównywane wartości sa dokładnie takie same.

Podwójny == nazywa się lose equality operator i np gdy przyrwónujemy „18” == 18 dostaniemy True. Zasadniczo staramy się unikać tego.

Uzyskanie wartości od użytkownika aka input, w tym celu używa się funkcji **prompt**:

const user\_value = prompt(„what’s your favorite color”)

Boolean operators = and, or, not(!) < tak jak w Python.

And operator jest zapisywany jako &&,

Or operator jest zapisywany jako ||,

Not operator jest zapisany jako !n,

**Switch statments:**

Działają na zasadnie bloków if ale z inną kosntrukcją np.:

Switch(day) {

case ‘monday’:

consolo.log(„…”)

consolo.log(„…”)

**break**

case ‘tuseday’:

consol.log(„…”)

**break**

case ‘wednesday:

case ‘thursday: (zadziałą dla obydwu rzeczy)

consol.log(„…”)

**break (jest potrzebny bo inaczej kod przejdzie do dalszej częsci)**

default:

consol.log(“Not a valid day”)

}

**Statements and Expressions**

**Expressions to np. 3 +4, 1991, true && false && !false**

**Statements to większa część kodu np. if else statements.**

**Conditional (Ternary) Operator**

Poniżej jest budowa conditional operatora, gdzie tworzymy warunek, następnie znakiem ? tworzymy blok spełnionego założenia i po : blok negatywnego spełnienia. Pozwala to zaoszczędzić dużo czasu przy prostych sprawdzeniach warunku if else.

Const age = 24

age >= 18 ? console.log(“I like alcohol”) : console.log(„I like wataa”)

Z przypisaniem do variable:

Const drink = age >= 18 ? “wine”: “water”

**Strict Mode**

Celem strict mode jest umożliwienie włączenie widzenia błędów oraz uniemożliwia wykonanie pewnych funkcji.

Aby aktywować strict mode należy dodać na początku skryptów **„use strict”**

**Funkcje**

Funkcje tak jak w python, to blok kodu, który można w łatwy sposób ponownie wykorzystać. Konstrukcja:

Function Blob(parameter1, parameter2) {

Console.log(parameter1, parameter2)

}

**Functions Declarations vs Expressions**

Obydwie formy pisania funkcji są właściwe, osobiście preferuje f. declaration. Różnica między nimi polega, że declaration f. może być wezwana przed jej blokiem.

Function declaraction:

function calcAge1 (birthYear) {

return 2033 - birthYear

}

const age1 = calcAge1(1991)

console.log(age1)

Function expression:

const calcAge2 = **function** (birthYear) {

return 2037 - birthYear

}

const age2 = calcAge2(1991)

console.log(age1)

**Arrow Function**

Jest to forma funkcji o bardzo prostej budowie, nadająca się do prostych funkcji, która automatycznie zwraca wartość swojego ciała.

Const calcAge3 = birthYear => 2033 – birthyear

Console.log(calcAge3(1990))

A co w przypadku bardziej rozbudowanej funkcji:

Const yearsUntilRetirement = birthYear => {

const age = 2037 – birthYear

const retirement = 65 - age

return retirement

}

W przypadku wielu parametrów:

Const yearsUntilRetirement = (birthyear, firstName) => {

const age = 2037 – birthYear

const retirement = 65 - age

return `${firstName} retirement in ${ retirement }`

}

**Functions Calling Other Functions**

Dla mnie logiczne.

Const cutFrutPieces = function (fruit) {

return fruit \* 4

}

Function fruitProcessor(apples, oranges) {

Const applePieces = cutFrutPieces(apples)

Const orangePieces = cutFrutPieces(oranges)

const juice = `Juice with ${applePieces} apples and ${orangePieces} oranges`

return juice

}

**Arrays**

Tak jak w Python syntax: const lista = [x, y, z]

Jest też drugi sposób: const lista = new Array(x, y, z)

Aby uzyskać długość listy stosuje się jej paramter: **lista.length**

Const Listy moga być zmieniane bo nie należa do primitive values, jak variable. Jednak nie mogą być zmienione w pełni. Np poniższa zamiana zwróci nam error:

Const lista = [x, y, z]

Lista = [b, c]

Arrays Methods:

PUSH – funkcja, która dodaje(mutuje orginalną listę) na koniec listy element:

lista.push(a)

ciekawostka, push zwraca nową długość zmutowanej listy:

const newlength = lista.push(a) -> zwróci 4.

UNSHIFT – funkcja, która dodaje element na poczatek listy:

lista.unshift(a)

POP – funkcja, która zabiera ostatni element z listy:

Lista.pop()

Tak jak w Python, pop zwraca element zabrany z listy:

Const element = Lista.pop() - > zwróci nam „z”

SHIFT – funkcja, która zabiera z początku listy element:

Lista.shift()

Tak jak w Python, pop zwraca element zabrany z listy:

Const element = Lista.shift() - > zwróci nam „x”

INDEXOF – funkcja, która zwraca nam index elementu w danej liście:

Lista.indexOf(y) -> zwróci nam 1.

Jeśli nie ma danego elemetu w liście zostanie nam zwrócone -1.

INCLUDES – funkcja, która zwraca nam true or false jeśli element znajduje się w danej liście.

Lista.includes(abc) -> zwróci nam False

**Introduction to objects**

**OBJECTS –** Obiekty sa podobne w konstrukcji do struktóry dictionary w Python. W JS mówi się, że obiket michal ma property name o wartości „Michał”.

Const michal = {

Name: „Michał”,

Lastname: „Krepiniewicz”,

Etc..

}

Dot vs. Bracket Notation

Tak jak w python, można uzyskac property poprzez dostanie się do obiektu na dwa sposoby:

Obiekt.name lub obiekt[„name”]

Zasadniczo używa się dot notation aby uzyskać wartość z obiektu ale tylko w przypadku kiedy znamy property name. Gdybyśmy chcili skorzystać z prmp, gdzie zadajemy pytanie i opcje do wyboru jak: age, name etc. To nalezy skorzystać z brackets:

Const data = prompt(„what do you ..., chose from age, firstname etc. ”)

naszObiekt[data]

gdybyśmy zrobili to z wykorzystaniem . naszObiekt.data dostalibyśmy undeifned bo nie istenieje taka właściwość obiektu.

Aby dodac elementy do obiektu możemy to zrobić na 2 sposoby:

naszObiekt[„location”] = „Warsaw”

albo

naszObiekt.facebook = „Fuck You”.

Ciekawa opcja do wyświetlenia danych obiektu w formie tabeli jest skorzystanie z **console.table(obiekt);**

**Objects Methods**

Obiekty moga posiadać swoje własne funkcje, np.

michal = {

birthYear: 1991,

calcAge: function (birthYear) { return 2037 – birthYear }

}

michal.function(1991) lub michal[“calcAge”](1991)

Aby dostac się property obiektu stosuje „**this**” variable/object, który równa się naszemu obiektowi ala self z Pythona.

michal = {

birthYear: 1991,

calcAge: function () { return 2037 – this. birthYear }

}

michal.function() lub michal[“calcAge”]()

Możemy również stworzyć nowe property przy pomocy wbudowanej funkcji:

michal = {

birthYear: 1991,

calcAge: function () {

this.age = 2037 – this. birthyear

}

}

michal.function() lub michal[“calcAge”]()

**For Loop**

Zasada działania jest jak w pytonie, jednak syntax jest inny:

Wzywamy for, gdzie w body mamy 3 wyrażenia: 1) które tworzy zmienną, 2) warunek

sprawdzający, 3) wpłynięcie na stworzoną zmienną na koniec loopa.

for( 1. let rep = 1; 2. rep&lt;11; 2. rep++) {

    console.log(rep)

}

const types = [];

for(i=0; i&lt;=mikeArray.length-1; i++) {

    // One of the ways of adding things into arrays

    types.push(typeof(mikeArray[i]))

    // Second way of adding things into arrays

    types[i] = typeof mikeArray[i]

    // Reading of the array

    console.log(mikeArray[i], typeof(mikeArray[i]))

}

console.log(types)

Mamy również continue and break statement. Continue sprawia, że loop przeksakuje do

kolejnej pętli. Break sprawia, że cały loop ulega przerwaniu.

Loop backwards

// Loop backwards

const mikeArray = [

    &quot;Mike&quot;, &quot;Krepiniewicz&quot;, 2037-1990, &quot;programmer&quot;, [1, 2 ,3]

]

for(let i=mikeArray.length-1; i&gt;=0; i--){

    console.log(mikeArray[i])

}

**While Loop**

Konstrukcja while loop jest dość ciekawa, tworzymy tylko warunek:

While Loop

let rep = 0

while(rep &lt;=10){

console.log(`${rep}`)

rep++

}

Drugi przykład z losowniem losowej liczby aż nie osiągniemy docelowej:

let score;

while(score !== 6){

    score = Math.trunc(Math.random()\*6)+1

    console.log(score)

}

Math.random zwraca nam liczbę pomiędzy 0 a 1, np. 0.2345, 0.4592461 etc.

Math.trunc pozbywa się wartości po przecinku.

Ciekawa rzeczy warunek gdzie: score === 6, nie działa.

**Setting up Prettier in Vs Code**

A screenshot of a computer

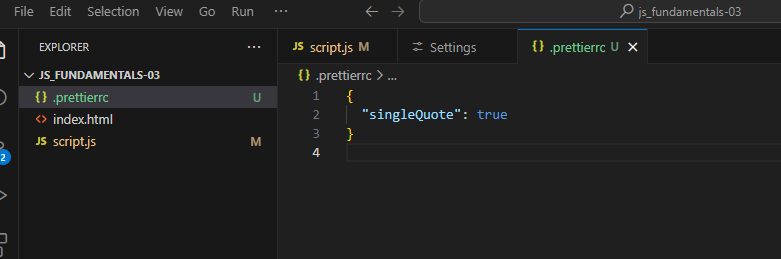
Description automatically generated

Następnie po ściagnięciu I zainstalowaniu należy zrobić „Enable” i wskazać go jako default format w settings:

A screenshot of a computer

Description automatically generated

Jesli jakaś bazowa funkcja nie pasuje w Prittier możemy wpłynać na bazowe sutawienia poprzez stworzenie pliku .priettierrc



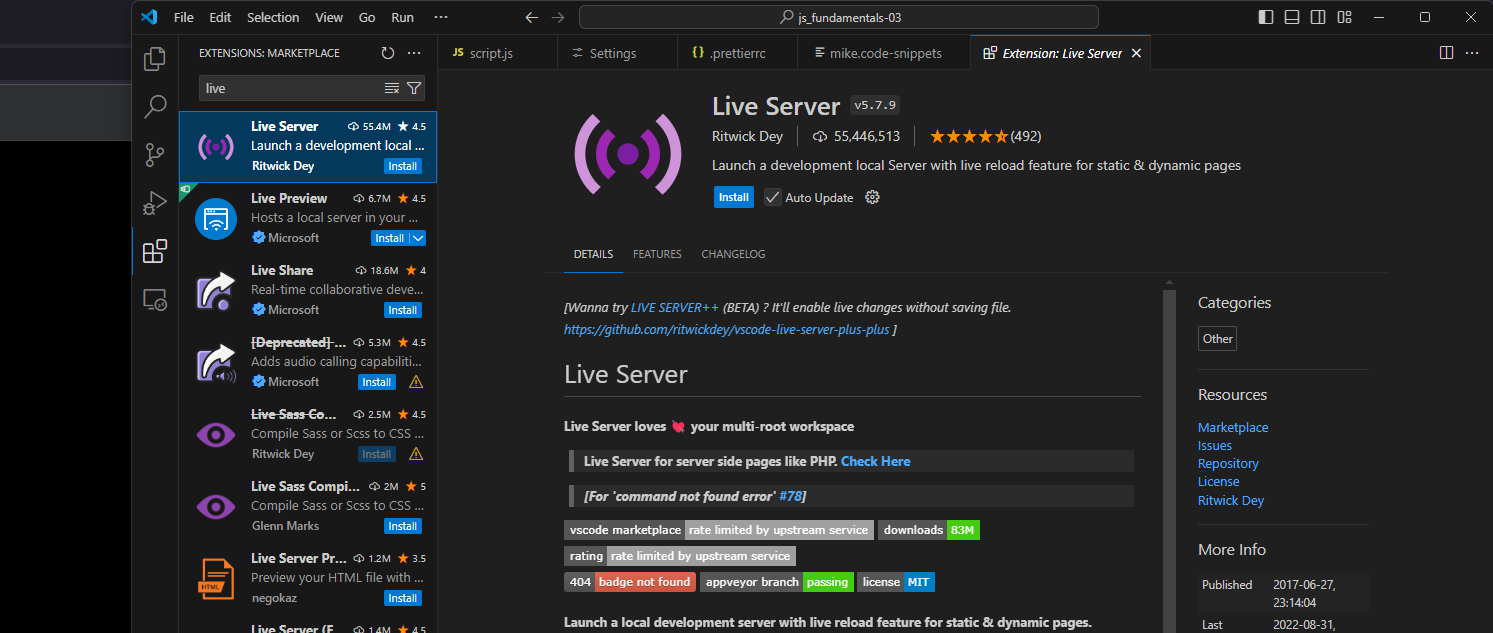
Snippets

Dzięki snippets możemy skonfigurować takie rzeczy jak uproszczenie pewnych funkcji jak: console.log(). W settings > snippets > new global snipet.

A screenshot of a computer program

Description automatically generated

VS Code extension do Live Server



Nastepnie instalujemy globalnie (-g) live-server:

A screen shot of a computer code

Description automatically generated

Jeśli dostaniemy bład przy odpaleniu live-server należy zmienić politkę uruchomiania skryptów w powershell jako admin:

A screen shot of a computer program

Description automatically generated

Aby skorzystać z live-server wystarczy, że w terminalu wpiszemy: **live-server**

A computer screen with text

Description automatically generated

Debugger przeglądarkowy

Można sprawdzać w przeglądarce w zakładce debugger ewentulane buggi zamiast w środowisku developera:

A computer screen shot of a program

Description automatically generated

Możemy dodać manualnie break point w postaci debuggera w wanyszm kodzie:

A screen shot of a computer program

Description automatically generated