JavaScript

Wstęp

Istnieją dwa sposoby wstawiania skryptów na stronę - możemy je umieścić w elemencie `<head>` lub `<body>`

```
<!DOCTYPE html> <html lang="en"> <head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial- scale=1.0"
>
<meta http-equiv="X-UA-Compatible" content="ie=edge"> <title>Document<
/title>
<script>
</script>
</head>
<body>
</body>
</html>
```

Drugim sposobem jest umieszczenie linku do zewnętrznego pliku JS - podobnie jak ma się z CSS. Pamiętaj, że pliki JS mają zakończenie js (np. `index.js`).

Aby umieścić link należy w sekcji `<head>` napisać:

```
<script src="js/index.js"> </script>
```

Rezultat:

```
<!DOCTYPE html>
<html lang="en"> <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="js/index.js"> </script>
</head>
<body>
</body>
</html>
```

Przetestujmy działananie poprzez umieszczenie w zewnętrzym pliku JS alertu.

Umieszczenie pliku JS w sekcji a umieszczenie w innym miejscu. Wszystkie skrypty, które są w sekcji muszą zostać wykonane aby parser przeglądarki mógł wykonywać kod. Dobrą praktyką jest umieszczanie linkowania do kodu JS przed tagiem - dzięki temu wczyta się kod HTML i CSS, a dopiero potem kod JS.

Typy danych i operatory

Zmienne - co to jest i do czego służą?

Deklaracja zmiennej polega na określeniu jej nazwy, która będzie przechowywać jakieś dane.

Często jest ona porównywana do etykiety na segregatorach, która określa zawartość jaką w nim znajdziemy.

Możemy od razu przypisać do niej wartość za pomocą operatora przypisania (=), lub pozostawić na razie bez wartości co będzie oznaczać, że jest undefined (niezdefiniowana).

```
var zmienna01 = "przyklad";
let zmienna02 = 123;
const zmienna03;
```

Dzięki zmiennym przechowujemy różne wartości w naszym programie, aby w odpowiednim momencie móc ich użyć.

```
const firstName = 'Marcin';
const lastName = 'Dłubis';
const fieldName = 'age';
const userAge = 31;
let userPoints = 0;
```

Dzięki zmiennym przechowujemy różne wartości w naszym programie, aby w odpowiednim momencie móc ich użyć.

Dobrą praktyką jest używanie języka angielskiego dla nazw zmiennych (jak i pozostałych elementów kodu).

Jeśli nazwa zmiennej składa się z kilku wyrazów to powinniśmy trzymać się formatu camelCase.

Nazwa zmiennej powinna jednoznacznie określać co przechowuje. Niedozwolone jest używanie liczb jako pierwszy znak nazwy zmiennej. Nie możemy używać spacji, kropki przecinka oraz myślnika w nazwie. Wielkość liter ma znaczenie dlatego warto trzymać się

wspomnianej zasady camelCase.

Dobrą praktyką jest deklarowanie zmiennych na samym początku kodu (lub zakresu), aby było wiadomo jakich zmiennych będziemy używać.

Przykład użycia zmiennych:

```
const number01 = 12.2;
const number02 = 12.3, number03 = 0.334;

const sum = number01+number02+number03;
const avarage =  sum/3;

console.log(avarage)
```

Co mogą przechowywać zmienne:

- Wartości prymitywne - „przyklad”
- Referencje do obiektów - Date
- Referencje do funkcji - f(a,b)

Jakie nazwy zmiennych są dozwolone? Nazwa może zaczynać się od:

- _ podkreślinka
- $ znaku dolara
- małych i wielkich liter

Przykłady:

- $zmienna
- _zmienna
- Zmienna_1
- Zmienna2

Błędne przykłady

- 0zmienna
- zmienna#1
- -zmienna1
- .zmienna

var & let & const

Słowo kluczowe var, które pozwala nam deklarować zmienne było wykorzystywane przed pojawieniem się ES6 (standard JavaScript z 2015r.).

Obecnie bardziej prawidłowym rozwiązaniem będzie korzystanie ze słów kluczowych let oraz const.

Co nie oznacza, że użycie słowa kluczowego var będzie błędem. Możesz się z tym spotkać w starszych rozwiązaniach lub takich, które muszą wspierać starsze przeglądarki.

const

Słowo kluczowe const pozwala deklarować zmienne, które w dalszej części kodu nie mogą być nadpisywane.

```
const price = 1.5;
const bigPrice = price + 100;
bigPrice = 1.75;
```

Rezultat:

```
// Uncaught TypeError:
// Assignment to constant variable.
```

let

```
let price = 1.5;
let bigPrice = price + 100;
bigPrice = 1.75;
```

Rezultat:

```
// OK
```

Wnioski:

Bezpieczniejszym rozwiązaniem będzie deklarowanie zmiennej przy pomocy `const` ponieważ nie zmienisz przypadkowo jej wartości w trakcie pisania kodu.

Słowa kluczowego `let` używaj tylko dla zmiennych, których zmienności jesteś pewien.

Co mogą przechowywać zmienne:

- Wartości prymitywne - „przyklad”
- Referencje do obiektów - Date
- Referencje do funkcji - f(a,b)

Słowa zarezerwowane:

W języku JavaScript istnieją słowa zarezerwowane, których nie można użyć jako nazwy

zmiennych.

Takimi słowami są: abstract, break, char, debugger, else, false, true, final, float, var, with itp.

Pełna lista słów zarezerwowanych: https://www.w3schools.com/js/js_reserved.asp

Typy wartości:

Przykłady zmiennych wraz z przypisaniem:

```
const firstName = 'Marcin';
const lastName = 'Dłubis';
const fieldName = 'age';
const userAge = 31;
let userPoints;
let subjects = [];
```

Typy proste

- number - np. 100, 0.92, NaN
- string - np. ‘studia podyplomowe’, "100"
- boolean - true, false
- undefined
- null //* symbol - np. nazwa()

Typy złożone

- object - np. [], {}
- function - np. function example(){}

Rodzaj przechowywanej wartości w zmiennej możemy sprawdzić za pomocą operatora typeof.

```
var example = 100;
console.log(typeof example);
```

Operatory

Operatory to symbole, które reprezentują pewną operację.

Operatory możemy podzielić na kilka kategorii:

- arytmetyczne
- przypisania
- porównania

- logiczne

Operatory arytmetyczne

- dodawanie
- odejmowanie
- mnożenie
- dzielenie
- konkatenacja
- reszta z dzielenia
- inkrementacja
- dekrementacja

Przykłady:

```
const number1 = 1;
const number2 = 2;
```

Przykład 1:

```
const number1 = 1;
const number2 = 2;
const result = number1 + number 2;
```

Sprawdź w konsoli rezultat!

Przykład 2:

Sprawdź w konsoli rezultat następującego działania:

```
const result = 1 + "2";
```

Zauważ, że rezultatem jest `string`.

Operację taką nazywamy konkatenacją, czyli łączeniem ze sobą wyrażeń.

Przykład 3:

Sprawdź w konsoli rezultat i typ następującego działania:

```
const result01 = number1 * "2";
const result02 = number1 * "liczba";
const result03 = number1/number2;
const result04 = number1 % number2;
const result05 = 4 % 2;
const result06 = 5 % 2;
```

Kolejność działań jest identyczna jak w zasadach matematyki!

Przypisanie

Przykład

```
number1 = number1 +3;
```

Zauważ, że zmienną number1 deklarowaliśmy wyżej, więc nie ma potrzeby używać ponownie słowa `var`, `let` lub `const`.

Sprawdź w konsoli, jaka jest nowa wartość number1.

Powyższy zapis możemy skrócić do:

```
number1 += 3
```

Operatory porównania

Rodaje operatorów:

- `==` - równanie z niejawną konwersją, przykład: `x == 2`
- `!=` - różne z niejawną konwersją, przykład `x !=2`
- `===` - równa bez konwersji, przykład `x === 1` lub `x === "1"`
- `!==` - rózne bez konwerji, przykład `x !== 2` lub `x !== "2"`
- `>` - większo od, przykład: `x > 3`
- `<` - mniejsze od, przyklad: `x <3`
- `>=` - większe lub równe, przyklad: `x >=3`
- `<=` - mniejsze lub równe, przyklad `x <=3`

Spróbuj teraz w oknie przeglądarki porównać powyższe z `x = 1`

Dobra praktyka

Najlepiej używać operatorów, które nie pozwalają na konwersję tj. `===` czy `!==`.

Więcej do poczytania:https://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3

Operatory logiczne

Rodzaje operatorów logicznych:

- `&&` - AND - (koniunkcja)
- `||` - OR - (alternatywa)

Przykad koniunkcji:

```
if ((number1< 100) && (number1>=0)) {
console.log(”ok”);
}
else
{ console.log(”błąd”); }
```

Przykład alternatywy:

```
if ((number1< 2) || (number1>=0))
{
console.log(”ok”);
}
else
{ console.log(”błąd”); }
```

Inkrementacja, dekrementacja

Inkrementacja - zwiększenie

```
var number1 = 5;
number1 = number1 + 1;
number1+=1;
number++;
```

Dekrementacja - zmniejszenie

```
var number1 = 5;
number1 = number1 - 1;
number1-=1;
number--;
```

Zadanie

Sprawdź czym rożni się `number1++` od `++number1` `number1--` od `--number1`

Instrukcje warunkowe

instrukcja warunkowa `If`

Służą do podejmowania decyzji w trakcie wykonywania kodu - np. wpisując do formularza pewną wartość sprawdzamy poprawność wpisania tej wartości i w zależności od poprawności (prawda lub fałsz) wyświetlać się może komunikat o błędnych danych lub poprawny wynik.

Warunek sprawdzamy w następujący sposób (pseudokod):

```
Jeżeli (warunek) to {czynność1 } a w przeciwnym wypadku {czynność2}
```

W kodzie będzie wyglądało to następująco:

```
if (warunek)
{ czynność1}
else {czynność2}
```

Przykład

```
const number = 100;
if (number != 100)
{
console.log(”nie jest ok”); }
else
{
console.log(”jest ok”); }
```

Przykład

```
const number = 100;
if (number > 100)
{
console.log(”nie jest ok”); }
else if (number < 100)
{
console.log(”nie jest ok”); }
else {
 console.log(”jest ok”); }
```

Przykład:

Mając warunek:

```
let x=10, y=50, res=0;
if (x > y ){console.log(x)} else {console.log(y)}
```

możemy go zapisać w skróconej wersji:

```
let res = (x > y) ? console.log(x) : console.log(y);
```

Co oznacza: jeżeli x > y to spełniony zostanie x, jeżeli nie to y

instrukcja warunkowa `switch`

```
let Number = 5;

switch (Number) {
    case 5:
        console.log("Szczęścliwa liczba to 5 ");
        break;
    case 30:
        console.log("Pechowa licza");
        break;
    case 99:
        console.log("Za duzo!!");
        break;
    default:
        console.log("coś poszło nie tak"); }
```

Przykład z zakresami

```
const x = 12;
switch (true) {
    case (x < 5):
        console.log("x<5");
        break;
    case (x < 9):
        console.log("x jest pomiędzy 5 i 8");
        break;
    case (x < 12):
        console.log("x jest pomiędzy 9 i 11");
        break;
    default:
        console.log("Nieoczekiwany wyjątek");
        break;
}
```

Zadanie: Napisz program, w którym podajesz swój wiek i wyświetla się komunikat, że jeżeli

masz poniżej 18 lat to ta strona nie jest dla Ciebie.

Pętle

Pętle to struktura pozwalająca na wykonywanie czynnośc do momentu osiągnięcia warunek jej przerwania - np.

- idź tak długo, zanim nie dotrzeć do celu
- zatankuj samochód do momentu wybycia wlewaka przez dystrybutor

Przykładem może być tak długie przeszukiwanie pewnego zbioru (fachowa nazwa - iteracja), aż nie natrafimy na pożądaną wartość.

W języku JavaScript wykorzystuje się następujące rodzaje pętli:

```
* while
* do while
* for
* for in
* for of
```

Pętla `for`

Pętla for jest wykorzystywana do czynności, które mają z góry narzuconą ilość operacji.

Zawiera ona zmienną przechowującą liczby (nazywaną sterującą), która zazwyczaj nosi nazwę i lub kolejno j, k itd.

Zmienna sterująca zmienia swoją wartość (np. i++ lub i–) po każdym wykonaniu kodu zapisanego między nawiasami klamrowymi.

```
for(let i=0; i <=10 ; i++) {
    console.log(i);
}
```

Kolejny przykład:

```
for(var i = 0; i <= 20; i++) {
    if( (i % 2) === 0 ) { console.log("ciekawa liczba"); }
    console.log(i);
}
```

Pętla `while`

Działa do momentu spełnienia zadanego warunku!

```
let number = 5;
while (number < =10) {
    console.log(number);
    number++;
}
```

Sprawdź, jaki rezultat byłby bez dodania `number++` .

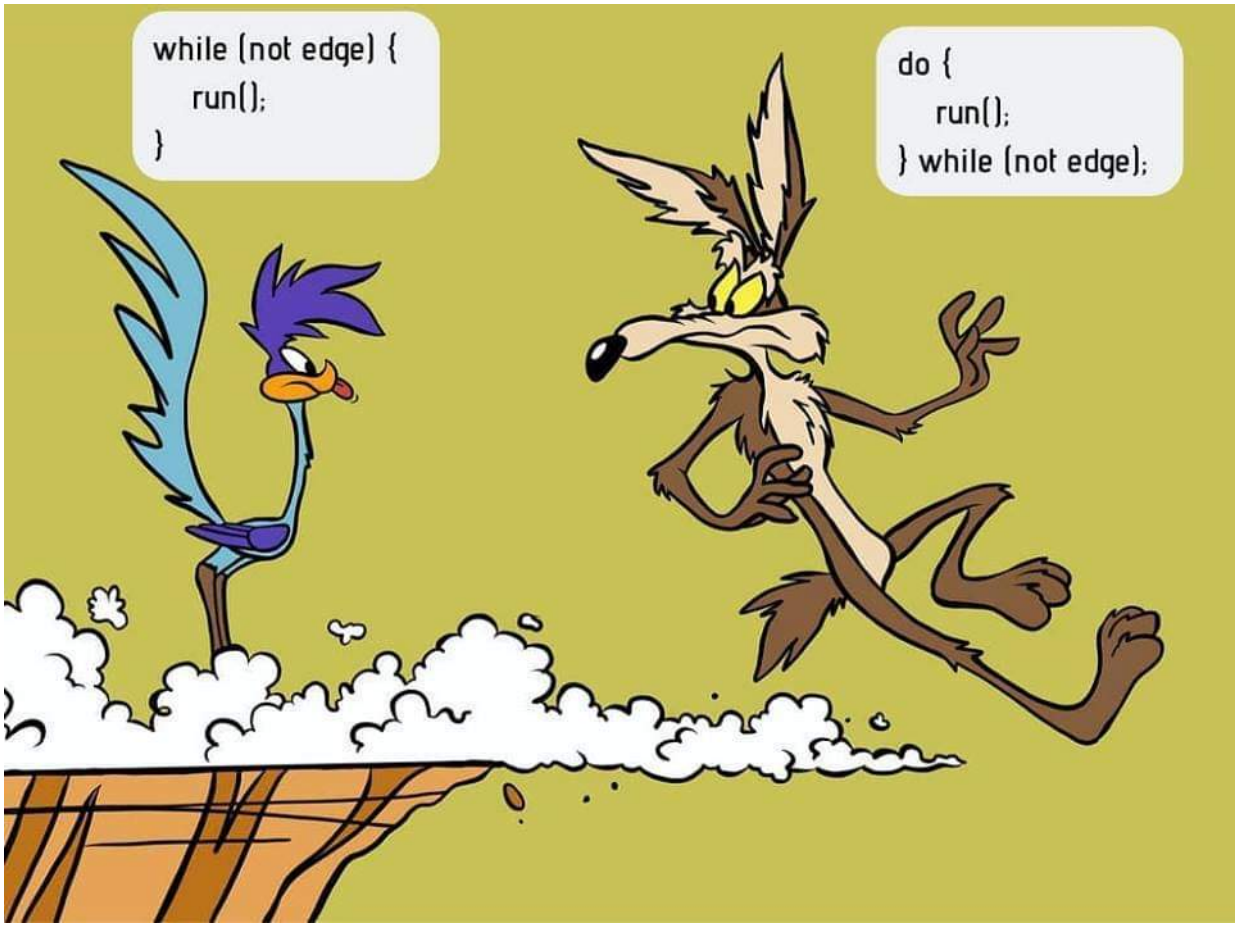
Przykład

```
while (prompt("Podaj liczbę") != 10) {
    alert("źle");
}
alert("dobrze");
```

Pętla `do while`

```
let number = 5
do
{
    console.log(number);
    number++;
}
while (number < 10)
```

Różnica pomiędzy pętlami



Zadania

- Wypisz liczby od 1 do 100
- Wypisz liczby od 5 do 50 z krokiem co 1
- Wypisz liczby podzielne przez 8 bez reszty w zakresie od 1 do 100

Funkcje

Funkcja to struktura, która pozwala grupować kod i uruchamiać go w dowolnym miejscu Twojego programu.

Podstawowa struktura funkcji zaczyna się od słowa kluczowego function, następnie wprowadzamy jej nazwę kończąc nawiasami okrągłymi, za którymi mamy już nawiasy klamrowe.

```
function saySomething(){
    console.log("Marcin");
}
```

To co znajduje się między nawiasami klamrowymi (tzw. wąsami) jest określane mianem ciała funkcji.

Wywołanie funkcji odbywa się poprzez wprowadzenie jej nazwy i dodanie do niej nawiasów okrągłych.

```
saySomething();
```

To w tym miejscu będzie uruchomiony } nasz kod znajdujący się w ciele funkcji.

Aby wykorzystać taką funkcję w kodzie możemy to zrobić zarówno przed, jak i po funkcji - w przeciwieństwie do zmiennych.

Nazewnictwo funkcji jest identyczne co dla zmiennych, tj. camelCase oraz jednoznaczne określenie co funkcja robi.

Aby wykorzystać taką funkcję w kodzie możemy to zrobić po funkcji - jak przy pisaniu zmiennych Wywołanie w konsoli: saySomething2()

Kolejny sposób tworzenia funkcji:

```
const saySomething2 = function() {
    console.log("saySomething2");
};
```

Zmienna lokalna a zmienna globalna

Przykład

```
var/const/let przykladowy_tekst="jeden";
function saySomething() {
    var przykladowy_tekst="dwa";
console.log(przykladowy_tekst); };

saySomething();
```

Zwracanie wartości

Utwórz funkcję i otwórz konsolę:

```
function saySomething() {
    console.log(”Dwa”);
};
```

Otrzymasz:

```
> saySomething()
  Dwa
< undefined
>
```

`Undefined` - to, co zwraca funkcja (bo nie skorzystaliśmy z return)

Aby funkcja coś zwracała:

```
function saySomething() { return ”Witaj”;
};
```

```
> saySomething()
< "Witaj"
>
```

Zwracanie wartości - zadanie

```
function sayYourName() {
    return "Cześć";
};

const showName = sayYourName();
console.log(showName + ", Marcin");
```

Niedziałający przykład

```
function number(){
    console.log(2)
}
const summary = number() + 3;
```

```
> function number(){
      console.log(2)
  }
  const summary = number() + 3;
  2
```

** Działający przykład**

```
function number(){
    return 2
}
const summary = number() + 3;
```

Parametry funkcji

Parametry funkcji to nazwy zmiennych pisane we wnętrzu nawiasów okrągłych, rozdzielonych przecinkiem.

```
function przykladowa_nazwa(parametr1)
    { return parametr1+parmetr1;
};
```

Aby wyświetlić:

```
console.log(przykladowa_nazwa(5));
```

lub:

```
const liczba = 5;
console.log(przykladowa_nazwa(liczba));
```

Funkcja może zawierać dowolną liczbę parametrów:

```
function sumowanie(a,b) { //-podajemy w nawiasie parametry
    return a+b;
};
```

A w konsoli wpisz `sumowanie(1,2)` – wpisujemy argumenty

Zakres zmiennych

```
var yourName = "Marcin"
function sayName(name) {
    var yourSurname = "Dłubis"
    return "Cześć, " + yourName + yourSurname;
};
```

w konsoli sprawdź wyświetlanie zmiennych `yourName` oraz `yourSurname`

Zadania

1. Kalkulator podatku VAT

2. Sprawdź, czy liczba jest parzysta

Obiekty

Wstęp

```
const person = {
    name: "Marcin",
    surname: "Dlubis",
    age: 28,
    statement: function() {
        console.log("Cześć" + this.name);
    }
}
```

Dzięki zastosowaniu obiektu, zamiast tworzyć kilka zmiennych (jak. name, surname, age) stworzona została jedna zmienna person.

W pliku `index.js` utwórz ten obiekt.

Dodanie właściwości

```
let person = {
    "my name": "Marcin",
    "my age": 28,
    occupation: "inzynier"
};

person.place = "Katowice";
person["my name"] = "Maciej";
```

Przykład

```
var car = {
manufacturer: "Peugeot",
model: "308",
saySomething: function() {
    console.log("Jestem w warsztacie"); }
};
```

Przykład

```
var car = {
manufacturer: "Peugeot",
model: "307",
saySomething: function() {
    console.log("Jestem w warsztacie" + car.manufacturer);
    /* console.log("Jestem w warsztacie" + this.manufacturer);*/
} };
```