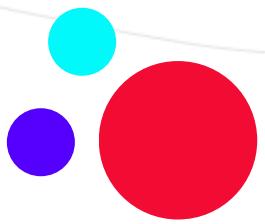


Cisco

JavaScript intro



HELLO

Dariusz Sibik

Lead Frontend Software Engineer (Spyrosoft)

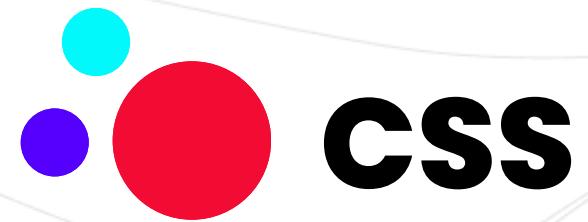




HTML (HyperText Markup Language) to podstawowy język znaczników do tworzenia i prezentowania stron internetowych.

- definiuje strukturę i zawartość strony (nagłówki, akapity, obrazy, linki, tabele itp.)
- osadzanie multimedów (wideo, audio)
- budowa aplikacji w połączeniu z innymi technologiami:
 - strony internetowe
 - szablony e-mail
 - aplikacje webowe
 - aplikacje mobilne
 - interfejsy aplikacji desktopowych
 - szablony cms



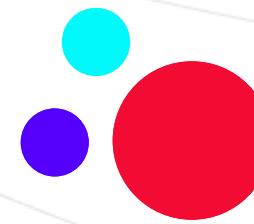


CSS (Cascading Style Sheets) – używany do definiowania wyglądu i sposobu prezentacji stron internetowych.

Kluczowe właściwości:

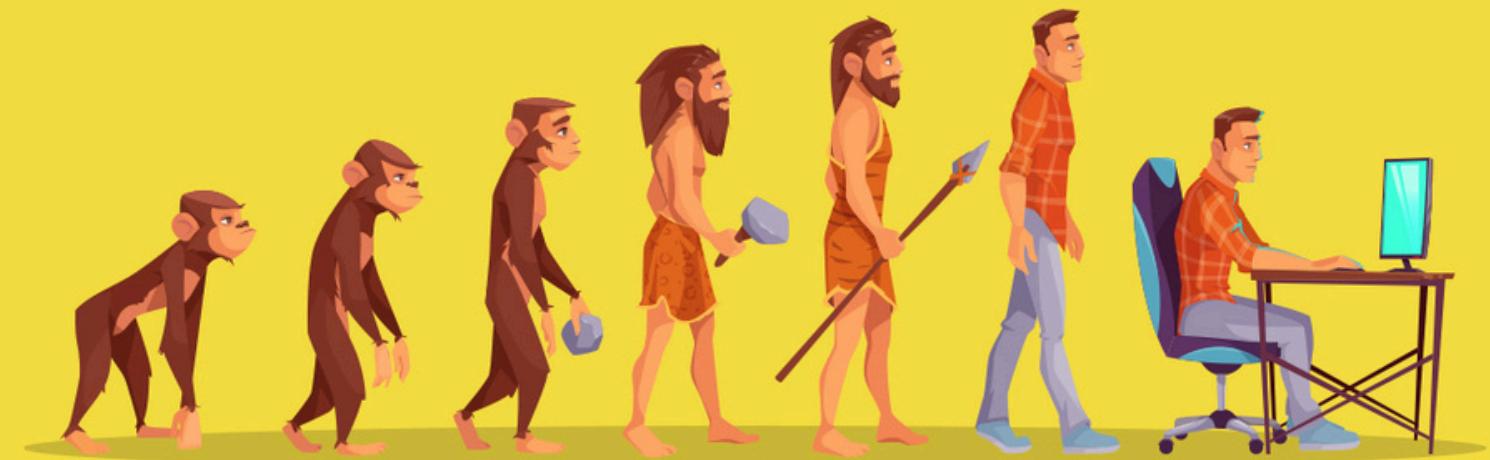
- Pozwala precyzyjnie zarządzać kolorami, czcionkami, marginesami i rozmieszczeniem elementów na stronie.
- Oddziela warstwę treści (HTML) od warstwy wizualnej (css).
- Umożliwia projektowanie stron responsywnych, które automatycznie dopasowują się do rozmiaru ekranu.
- Daje możliwość tworzenia animacji i różnorodnych efektów wizualnych.
- Stanowi podstawę nowoczesnych, atrakcyjnych interfejsów użytkownika.

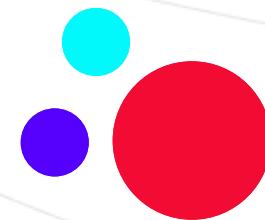




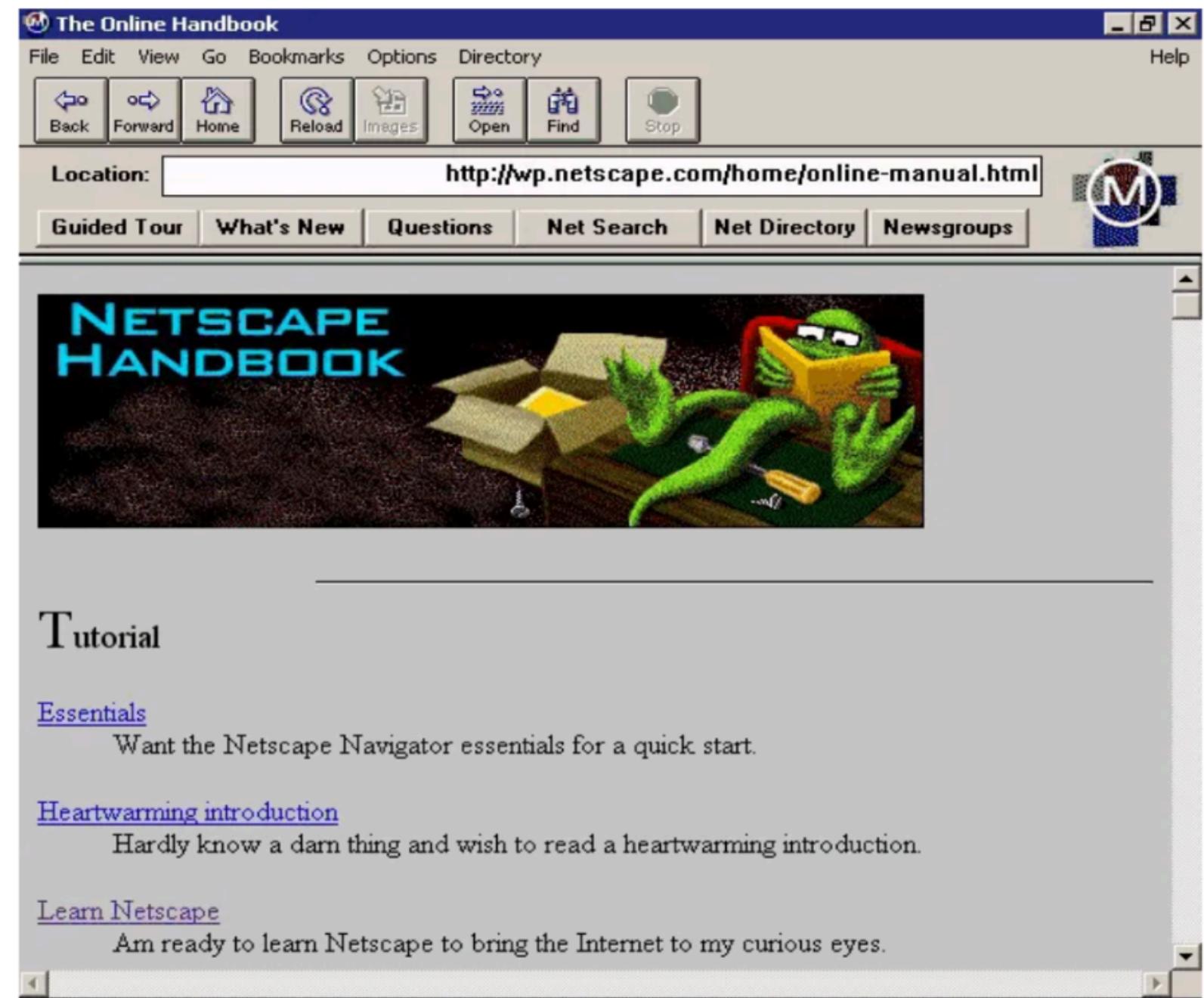
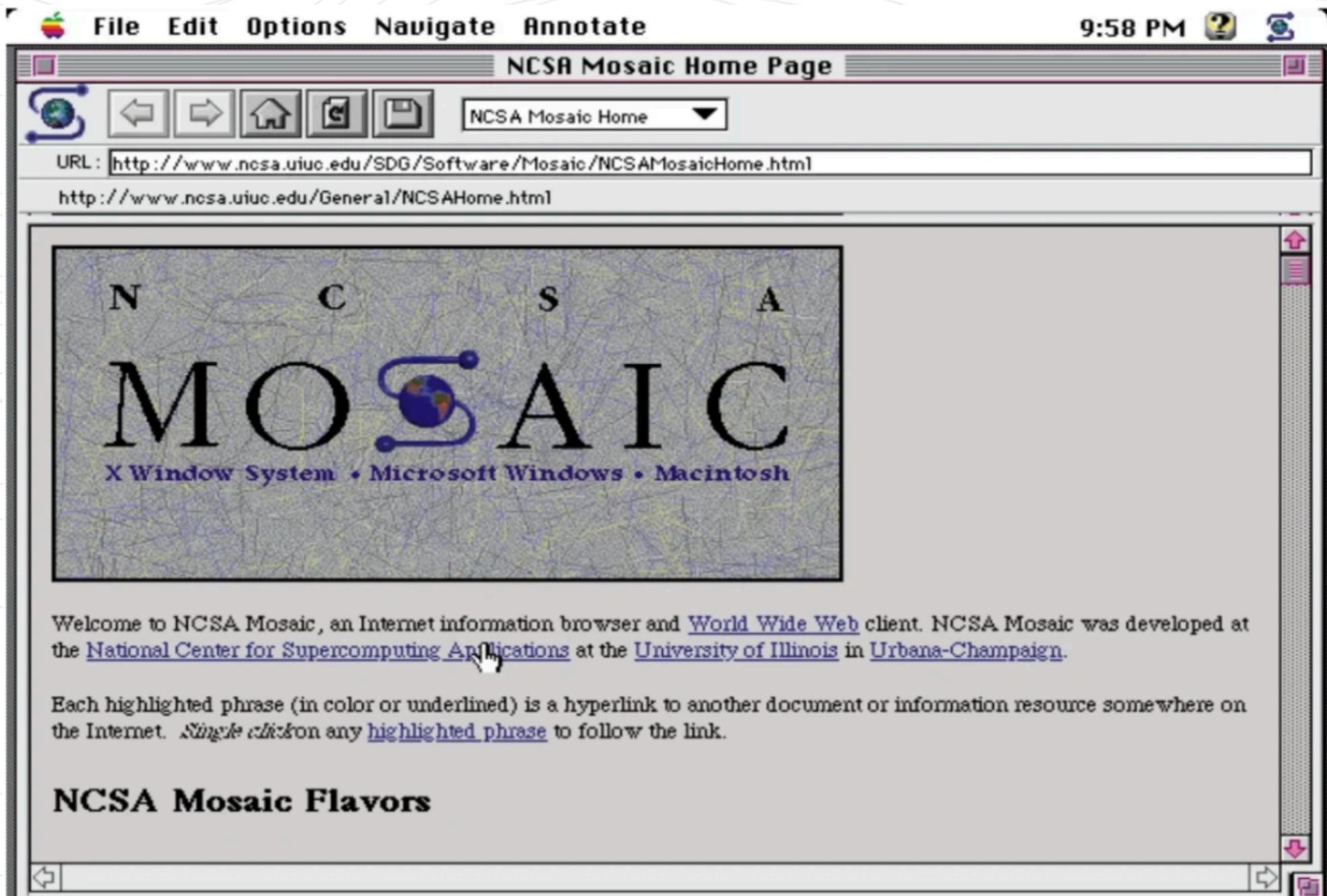
Podstawy teori - Historia JavaScript

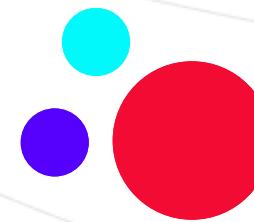
JS



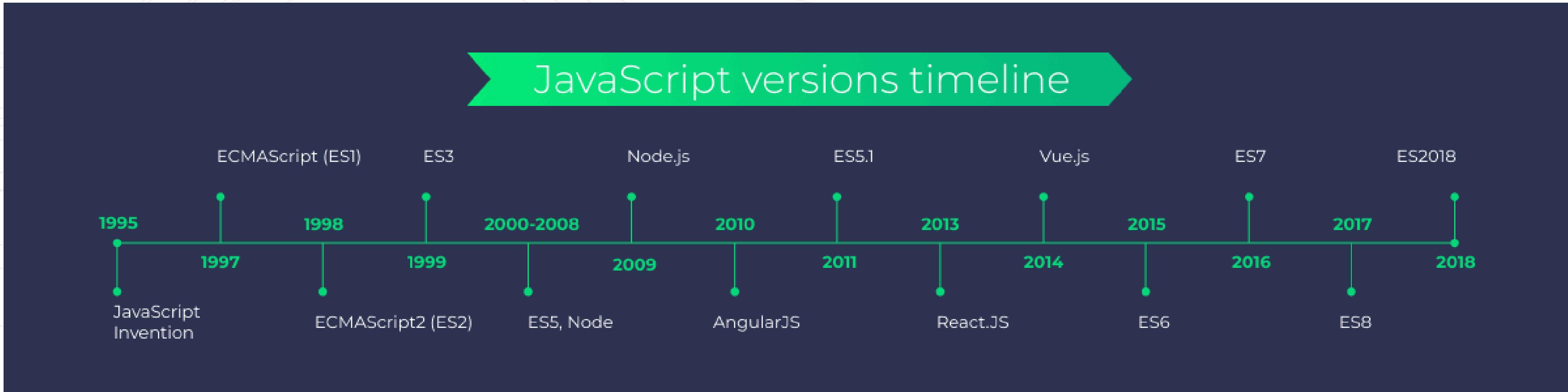


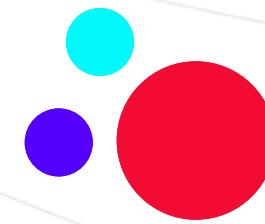
Podstawy teori - Historia JavaScript



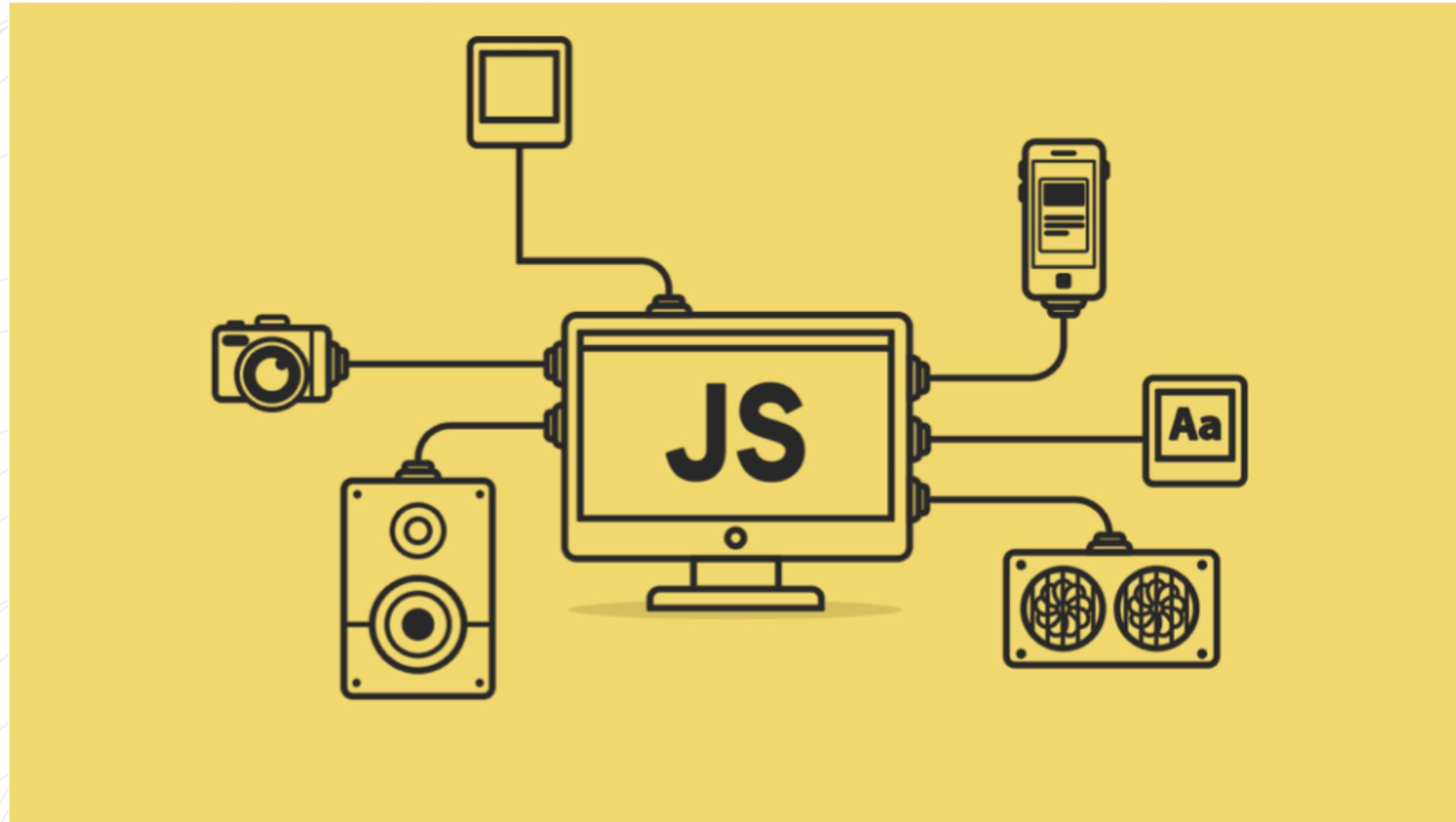


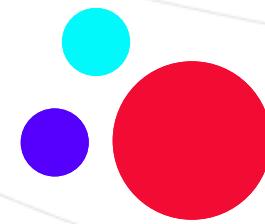
Podstawy teori - Historia JavaScript





Podstawy teori - środowiska uruchomieniowe





Podstawy teori - środowiska uruchomieniowe

- Przeglądarki internetowe.
- Serwer (Node.js)
- IoT (np. JerryScript)
- Mobile (React Native, Flutter)
- Desktop (Electron)



Zmienna to "pojemnik" na dane.

Pozwalają one nazwać pewne wartości, dzięki czemu nie trzeba ich później znowu podawać.

Zmienne mogą przechowywać liczbę, łańcuch znaków (napis), wartość logiczną (true/false), funkcje itd.

Zasady dotyczące nazw zmiennych:

1. mogą składać się ze znaków alfanumerycznych (cyfr, liter),
oraz znaku podkreślenia (_) i dolara (\$)
2. nie mogą zaczynać się od cyfr
3. nie mogą zawierać spacji
4. nazwy są wrażliwe na wielkość znaków (case sensitive)
5. nie można używać słów kluczowych jako nazw
zmiennych jak (return, for, break, if itp) - pełna lista



var

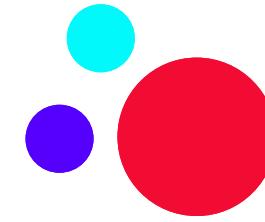
- zasięg funkcyjny
- można ponownie zadeklarować
- można używać zmiennej przed jej zadeklarowaniem (**hoisting**)
- deklaracja var poza ciałem funkcji rozszerza obiekt window
- można przypisać nową wartość do zmiennej

let

- zasięg blokowy
- niemożliwe jest ponowne zadeklarowanie
- niemożliwe jest używanie zmiennej przed jej zadeklarowaniem
- można przypisać nową wartość do zmiennej

const

- zasięg blokowy
- niemożliwe jest ponowne zadeklarowanie
- niemożliwe jest używanie zmiennej przed jej zadeklarowaniem
- do stałej nie możemy przypisać nowej wartości, ale można zmieniać właściwości obiektu



Typy proste w JavaScript

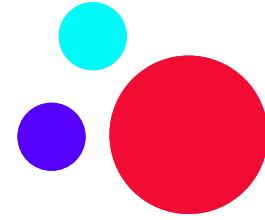
- Null
- Undefined
- Boolean
- Number
- BigInt
- String
- Symbol

Null

- jest wartością, która może być przypisana do zmiennej. Reprezentuje celowy brak wartości, referencji do obiektu.
- oznacza, że zmienna istnieje, jest zadeklarowana oraz zainicjalizowana (posiada przypisaną wartość null, choć nie posiada jeszcze docelowej wartości).
- celowo wskazuje na nieistniejący lub nieprawidłowy obiekt lub adres
- jego zachowanie jest z pozoru prymitywne, jednak w przypadku użycia `typeof` operatora zwraca "object"

Undefined

- wartość zmiennej, która nie została jeszcze zdefiniowana
- wartość zmiennej, która została zadeklarowana, ale nie została przypisana wartość
- jedyną wartością typu `undefined` jest wartość `undefined`
- w przypadku użycia `typeof` operatora zwraca `undefined`

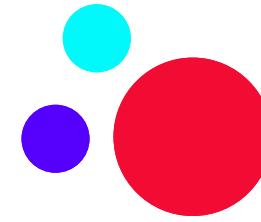


Number

W JavaScript występuje tylko jeden typ liczbowy. Wewnętrznie jest on reprezentowany jako 64-bitowa liczba zmiennoprzecinkowa.

Floating point calculations precision – Liczby binarne zmiennoprzecinkowe nie nadają się zbyt dobrze do operacji na ułamkach dziesiętnych, więc $0.1 + 0.2$ nie jest równe 0.3 . Jest to jeden z najczęściej zgłaszanych błędów w JavaScript. Wynika to z przyjęcia standardu IEEE 754 (IEEE Standard for Binary Floating-Point Arithmetic).





Podstawowe operacje matematyczne

Dodawanie i odejmowanie – operatory (+ i -)

- let suma= 4 + 5
- let roznica= 3 - 10

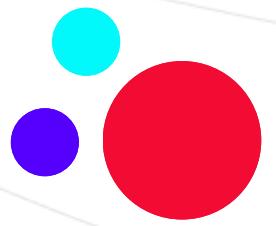
Mnożenie i dzielenie – operatory (* i /)

- let iloczyn= 2 * 3
- let iloraz= 10 / 2

Dzielenie przez 0

- let pamietaj= 4 / 0

Wynik dzielenia przez zero to Infinity



Podstawowe operacje matematyczne

```
// Liczby całkowite  
const n = 5;  
  
// Liczby zmiennoprzecinkowe tj. ułamki  
const m = 3.14;  
  
// Obiekt nieskończoność  
let infinity = Infinity;  
  
// Obiekt not a number  
let takNieRobimy = 'test' / '2'  
  
// Wartości binarne  
let binary = 0b1010101;
```

```
// Dodawanie  
const sum = 2 + 4;  
  
// Odejmowanie  
const diff = 4 - 2;  
  
// Mnożenie  
const product = 2 * 4;  
  
// Dzielenie  
const quotient = 4 / 2;  
  
// Modulo  
const modulo = 5%2;  
  
// Potęgowanie  
const pow = 2**2;
```



String

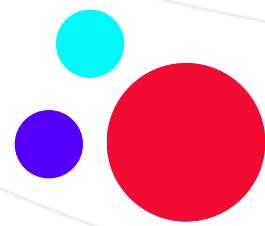
Obiekt **String** służy do reprezentowania i manipulowania sekwencją znaków.

Ciągi są przydatne do przechowywania danych, które można przedstawić w formie tekstowej.

Niektóre z najczęściej używanych operacji na łańcuchach to sprawdzanie ich length, budowanie i łączenie ich za pomocą operatorów łańcuchowych + i +=

Możemy definiować za pomocą znaków:

- '
- "
- `



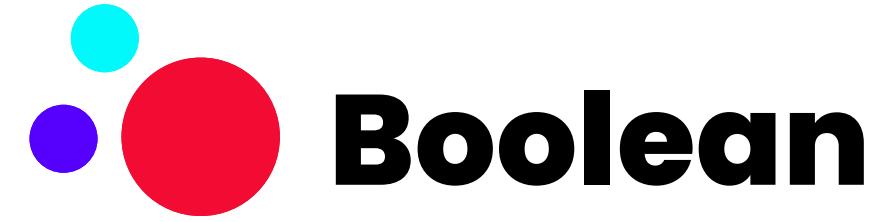
Konkatenacja czyli łączenie ciągów znaków

```
// wynik: "Adam Kowalski"  
let user = 'Adam' + ' ' + 'Kowalski';
```

```
// wynik: "Adam Kowalski"  
let user2 = 'Adam '.concat('Kowalski');
```

```
// wynik: "MojaNazwa"  
let text = "Moja" + " Nazwa";
```

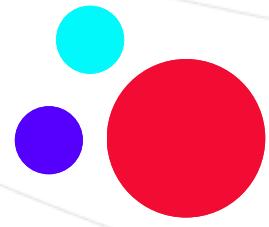
```
// wynik: "true"  
let flag = true + "";
```



Typ boolean może mieć dwie wartości tj. **true** i **false**.

Wartości tego typu wykorzystujemy np. do warunkowania stanu aplikacji.

```
let shouldBeRed = true;  
const color = shouldBeRed ? 'red' : 'black';
```



Operatory logiczne i porównania

Operacje w wykorzystaniem operatorów porównania zwracają wartości typu boolean

- ==** jest równe bez konwersją typów
- !=** nie jest równe bez konwersji typów
- ===** jest równe z konwersją typów
- !==** nie jest równe z konwersją typów

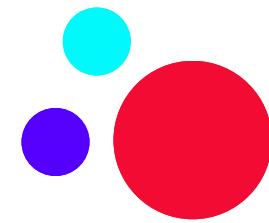
Logiczne AND -operator **&&**

Logiczne OR -operator **||**

Logiczne NOT -operator **!**

Nullish coalescing:

?? – nowe logiczne lub, zwraca drugą wartość w przypadku, gdy pierwsza jest nullish



Jawna i niejawna konwersji typów (koercja)

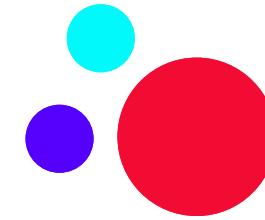
Jest to proces konwersji wartości z jednego typu na inny (taki jak ciąg znaków na liczbę, obiekt na wartość logiczną itd.)

JavaScript automatycznie zmienia typy danych, aby wykonać operację.

```
const value1 = "5";
const value2 = 9;
Let sum = value1 + value2; // 59

sum = Number(value1) + value2; // 14
```

info Share
A C A D E M Y



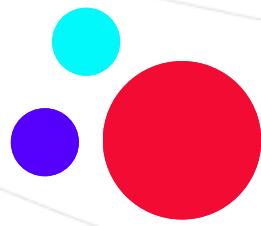
Instrukcja warunkowa (conditional statement)

if...else to podstawowa instrukcja, która wykona operację jeśli określony warunek jest prawdziwy.

Jeśli warunek jest fałszywy, zostanie wykonana operacja w opcjonalnej klauzuli else.

```
if (warunek) {  
    // wykona się jeśli warunek jest prawdziwy  
}  
  
if (warunek) {  
    // wykona się jeśli warunek jest prawdziwy  
} else {  
    // wykona się jeśli warunek jest fałszywy  
}
```





Instrukcja warunkowa (conditional statement)

Instrukcje warunkowe możemy zagnieżdzać, oraz możemy sprawdzić więcej warunków.

Warunki będą sprawdzane od góry do dołu i jeśli żaden z nich nie zostanie spełniony to zostanie wykonana operacja w else.

```
if (warunek) {  
    // wykona się jeśli warunek jest prawdziwy  
} else if (warunek2) {  
    // wykona się jeśli warunek2 jest prawdziwy  
} else if (warunek3) {  
    // wykona się jeśli warunek3 jest prawdziwy  
} else {  
    // wykona się jeśli warunek jest fałszywy  
}
```

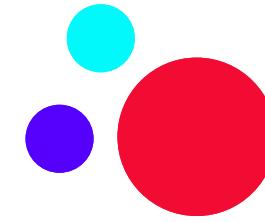


Instrukcja switch ocenia warunek do serii klauzul case i wykonuje operacje kiedy case spełni warunek, aż do napotkania break.

Klauzula default zostanie wykonana, jeśli pozostałe klauzule case nie spełniają warunku.



```
switch (warunek) {  
    case 1:  
        // wykona się jeśli wartość sprawdzana równa jest 1  
        console.log("1");  
        break; // nie pozwoli sprawdzać kolejnych warunków  
    case 2:  
    case 3:  
        // wykona się jeśli wartość jest równa 2 lub 3  
        // 2 nie ma break dlatego przejdzie dalej  
        console.log(" rowne 2 lub 3");  
        break;  
    default:  
        console.log(  
            "żaden case nie spełnił warunek"  
        );  
}
```



Pętla while

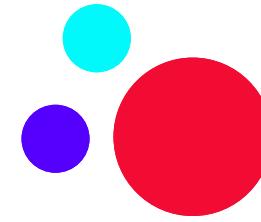
Instrukcja **while** tworzy pętlę, która wykonuje określoną operację, o ile warunek testowy ma wartość **true**.

Warunek jest oceniany przed wykonaniem instrukcji.

```
let n = 0;

while (n < 3) {
    console.log('Hello World!')
    n++;
}
```





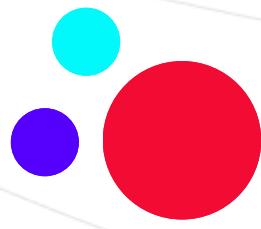
Pętla do while

W przeciwieństwie do zwykłej pętli while, w której warunek sprawdzany jest przed wykonaniem instrukcji, pętla do...while sprawdza warunek po wykonaniu operacji.

```
let end = false;

do {
    console.log('start');
    end = true;
} while (end === true)
```



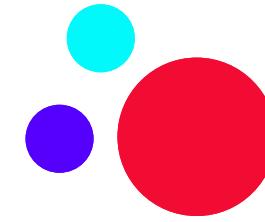


Pętla for

Pętla warunkowa for wykonuje szereg operacji.

Pętle takie najczęściej stosuje się w sytuacjach, kiedy dokładnie znamy liczbę powtórzeń.

```
for (zainicjowanie_zmiennych; warunek_kończący_wykonywanie_pętli; zmiana_zmiennych) {  
    // kod który zostanie wykonany pewną ilość razy  
}  
  
//pętla od 0 do 99  
for (let i=0; i<100; i++) {  
    console.log("Hello World!");  
}
```

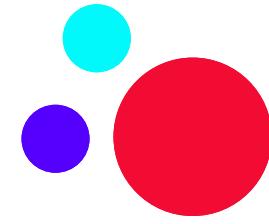


Słowa kluczowe break oraz continue

Na wykonywania pętli możemy wpłynąć instrukcjami break oraz continue.

Break – przerwuje wykonanie całej pętli.

Continue – przerwuje wykonanie bieżącego kroku i przechodzi do następnego.



Iteracja z użyciem for..of i for..in

Iteracja jest procesem przechodzenia przez elementy tablicy lub obiektu, aby wykonać na nich operacje lub uzyskać dostęp do ich wartości.

for...of – jest używane do iteracji po elementach kolekcji, takich jak tablice lub ciągi znaków.

for...in – jest używane do iteracji po właściwościach obiektu.

Przechodzi przez wszystkie właściwości w obiekcie, w tym te zapisane w prototypie obiektu.

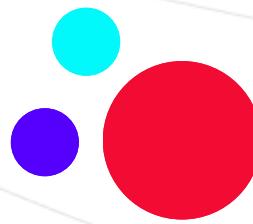
```
const heroNames = ['Hulk', 'Black Widow', 'Ms. Marvel', 'Deadpool'];
```

Zmiennne pomocnicze można dodać ich więcej po przecinku

Warunek, który musi zwrócić true, by nastąpiła kolejna iteracja

Kod, który wykonuje się po wykonaniu bloku kodu i kończy iterację

```
for (let index = 0; index < heroNames.length; index++) {  
    heroNames[index]; // access to array elements one by one;  
  
    continue; // use keyword continue to skip current iteration  
    break; // use keyword to stop loop;  
}
```



for...of vs for..in

```
const heroNames = ['Hulk', 'Black Widow', 'Ms. Marvel', 'Deadpool'];

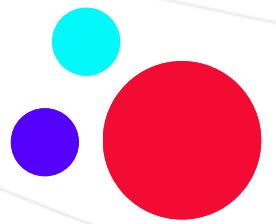
// the number of iteration is exactly the same as the array length in both loop

for (let index in heroNames) {
    heroNames[index]; // access to current array element

    continue; // use keyword continue to skip current iteration
    break;    // use keyword to stop loop;
}

for (let hero of heroNames) {
    hero; // access to current array element

    continue; // use keyword continue to skip current iteration
    break;    // use keyword to stop loop;
}
```



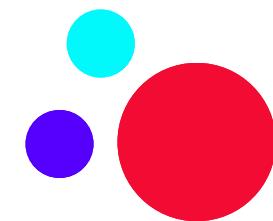
for...of vs for..in

```
const numbers = [1, 2, 3, 4, 5];

for (const number of numbers) {
  console.log(number); // 1, 2, 3...
}

const person = {
  name: 'John',
  age: 30,
  job: 'developer'
};

for (const key in person) {
  console.log(key); // name, age, job
  console.log(key, person[key]); // name John, age 30...
}
```



Tablice

Tablica to uporządkowana struktura danych, w której poszczególne elementy dostępne są za pomocą kolejno numerowanych indeksów, zaczynając od indeksu 0.

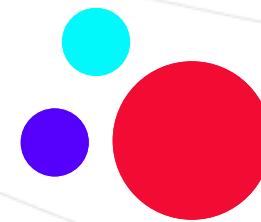
Wyobraź sobie, że przechowujesz użytkowników Twojej aplikacji w zmiennych user1, user2, user3 itd. Tworzy to oczywisty problem – jak mielibyśmy tak pisać aplikacje dla tysięcy użytkowników?

Tymczasem każdy użytkownik naszej aplikacji mógłby znaleźć się w tablicy, czyli "dynamicznym kontenerze" w którym możemy manipulować dowolnie użytkownikami.

```
// najczęściej stosowany sposób na utworzenie tablicy
const array = ["Adam", "Bartek", "Cezary"];

// tablica to typ object
typeof [] // "object"
```

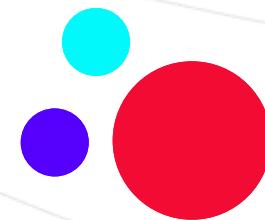




Co możemy przechowywać w tablicy?

Tablica jest odpowiednikiem ciągu, a tablica dwuwymiarowa jest odpowiednikiem macierzy znanych z matematyki.

```
// array przechowuje wszystkie typy danych w JS
// w tablicy mamy definicję funkcji
// tablice mogą przechowywać inne tablice - tablice wielowymiarowe
const arr = [
  0,
  "Adam",
  true,
  undefined,
  null,
  Infinity,
  { name: "Marek", role: "Admin" },
  [true, false],
  function fnInsideArr(args) {
    console.log(args);
  },
  Symbol("foo"),
  1n
]
```



Metody tablicowe

```
// tworzenie tablicy
let users = ['Adam', 'Anna', 'Pawel', 'Patrycja'];
let newArray = Array(5); // tworzy tablicę 5 elementów

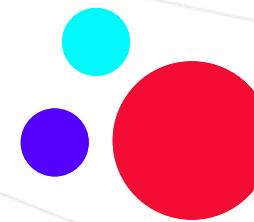
Array.isArray(users); // true

// tablica posiada właściwość Length
users.length; // 4

// dodawanie elementów
users.push('Karol'); // dodawanie elementu na koniec tablicy
users.unshift('Bartek'); // dodaje element na początek tablicy

// kasowanie elementów
users.pop(); // usuwa ostatni element tablicy
users.shift(); // usuwa pierwszy element tablicy

users.reverse(); // odwraca kolejność elementów tablicy
users.sort(); // pozwala posortować elementy tablicy
```

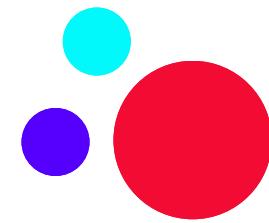


Iterowanie po tablicy

Iteracja oznacza powtarzanie tej samej operacji z góry określoną liczbę razy lub aż do spełnienia określonego warunku.

Wyobraź sobie, że posiadasz listę uczniów w klasie. Twoim zadaniem jest wyczytanie po kolei w ten sam sposób imienia i nazwiska każdego z uczniów. Wiesz, że w klasie jest 20 uczniów. Czytasz imię i nazwisko po czym, przechodzisz do kolejnej osoby i w identyczny sposób wyczytujesz kolejnego ucznia, aż dojdziesz do ostatniego i wtedy zakończysz wyczytywanie. I w ten oto sposób przeiterowałaś się przez listę uczniów 😊

```
let icons = ['🍏', '🍌', '🥕', '🍩'];  
  
// iterować po tablicy możemy poprzez dodanie w warunku length tablicy  
// zaczynamy od elementu 0 bo pierwszym elementem w tablicy jest 0  
for (let i = 0; i < icons.length; i++) {  
    console.log(icons[i]);  
}
```



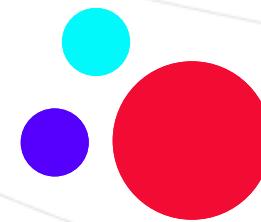
Deklarowanie i definiowanie funkcji

Funkcje są jednym z podstawowych elementów składowych języka JavaScript. Funkcje to zbiór instrukcji, które możemy odpalić poprzez podanie ich nazwy. Celem funkcji jest wykonanie operacji, zadania lub obliczenie wartości.

Każda funkcja po wywołaniu wykonuje swój wewnętrzny kod i może na końcu zwrócić jakąś wartość. Jeśli funkcja jest powiązana z obiektem (jest jego częścią) to nazywamy ją metodą.

```
function hello() {  
    console.log('Hello world!');  
}  
  
function sum(a, b) {  
    return a + b;  
}  
  
// wywołanie funkcji  
hello();  
sum(2, 3);
```





Funkcje – parametry i argumenty

Parametr to nazwa zmiennej przekazanej do funkcji.

Parametr jest wymieniony w definicji funkcji.

Argumenty funkcji to rzeczywiste wartości przekazywane do funkcji.

Parametry są inicjowane wartościami podanych argumentów.

```
function example(parameter) {  
  console.log(parameter); // foo  
}  
  
const argument = "foo";  
  
example(argument);
```



Zasoby

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean
- <https://developer.mozilla.org/en-US/docs/Glossary/Primitive>
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null>
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_AND
- <https://dev.to/damxipo/javascript-versus-memes-explaining-various-funny-memes-2o8c>
- <https://www.freecodecamp.org/news/js-type-coercion-explained-27ba3d9a2839/>



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics>

<https://javascript.info/object-basics>

<https://javascript.info/data-types>

https://www.youtube.com/watch?v=9ooYYRLdg_g

<https://www.youtube.com/watch?v=UgEaJBz3bjY>

for...in

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in>

for...of

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of>



Zasoby

if...else

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else>

while

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

do while

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/do...while>

for

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

continue

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

break

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



THANK YOU FOR YOUR ATTENTION

infoShareAcademy.com