

# JavaScript



# Cześć!

Maciej Kucharski

Frontend/JS Developer @SentiOne

# Agenda

1. Mega skrócona historia JS
2. Słowa kluczowe i osadzanie kodu JS
3. Zmienne i typy
4. Gramatyka
5. Operacje matematyczne
6. Obiekty
7. Listy
8. Q&A

# Krótką historia JavaScriptu

- Powstał w 10 dni (1993r.)
- Brendan Eich
- Netscape Navigator



# Historia JavaScriptu

# Historia JavaScriptu

## ECMAScript

ECMAScript is a language specification standardized by Ecma International in ECMA-262 and ISO/IEC 16262.

- ECMAScript - 1997
- ECMAScript 5.0 - 2009
- ECMAScript 6 - 2015
- ECMAScript 7 - 2016
- ECMAScript 8 - 2017

# Historia JavaScriptu

## Gdzie mogę użyć tego języka?

- Strony internetowe
- Serwery (Node.JS)
- Mobilne aplikacje (Cordova, React Native)
- IoT (internet of things, raspberry pi, arduino)

# Historia JavaScriptu

## Silnik

JavaScript jest językiem **interpretowanym**, dlatego potrzebuje interpretera do działania.

“In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.” - *Wikipedia*

Silnik JavaScript jest interpreterem, który interpretuje i wykonuje kod JavaScript



# Historia JavaScriptu

## Silniki JavaScript

**SpiderMonkey** nazwa kodowa pierwszego silnika JavaScriptu napisanego przez **Brendana Eich** z firmy **Netscape Communications**, silnik napisany w C++.

**V8** jest najpopularniejszym silnikiem rozwijanym przez firmę Google, napisany w C++.

**Rhino** zarządzany przez Mozilla Foundation, opensource

# Historia JavaScriptu

## Środowisko JavaScript

Głównym środowiskiem dla programów/skryptów w języku JavaScript jest przeglądarka i NodeJS.

## “ Zadanie 0 - Hello World

- *Otwórz przeglądarkę*
- *Otwórz narzędzia deweloperskie*
- *Przejdź do zakładki “console”:  
***alert('hello world')****

# Słowa kluczowe i osadzanie kodu na stronie

# Słowa kluczowe w języku

## Zarezerwowane słówka

- [break](#)
- [case](#)
- [catch](#)
- [class](#)
- [const](#)
- [continue](#)
- [debugger](#)
- [default](#)
- [delete](#)
- [do](#)
- [else](#)
- [export](#)
- [extends](#)
- [finally](#)
- [for](#)
- [function](#)
- [if](#)
- [import](#)
- [in](#)
- [instanceof](#)
- [new](#)
- [return](#)
- [super](#)
- [switch](#)
- [this](#)
- [throw](#)
- [try](#)
- [typeof](#)
- [var](#)
- [void](#)
- [while](#)
- [with](#)
- [yield](#)

# Osadzania

## Osadzanie JS w kodzie strony HTML

---



```
1 <script>
2   ...
3 </script>
4
5 <script src="script.js"></script>
```

# Osadzania

## Osadzanie JS w kodzie strony HTML

Do prawidłowego działania kodu JS istnieją dwa miejsca, w które możemy umieścić osadzenie kodu JS.

- `<head> . . . . . </head>` tag
- `<body> . . . . . </body>` tag

Które jest “lepsze”?



## *Zadanie 1 - Hello World 2*

- *Dodaj “alert(‘hello world’)”  
najpierw w pliku HTML,  
następnie stwórz nowy plik  
\*.js a następnie osadź go na  
stronie HTML*



# Zmienne i typy

# Gramatyka i słowa kluczowe

## Typy danych

### Typy proste:

- undefined,
- null,
- booleans,
- numbers,
- strings

Wszystkie inne typy są **obiektami**, włączając w to listy i **funkcje**.

**Typy proste są przekazywane przez wartość, a obiekty przez referencje.**

# Gramatyka i słowa kluczowe

## Literały (ustalone wartości tekstowe lub liczbowe)



```
1 // Null literal
2 null
3
4 // Boolean literal
5 true
6 false
7
8 // Numeric literals
9 1234567890
10 42
```

# Gramatyka i słowa kluczowe

## Literały (ustalone wartości tekstowe lub liczbowe)



```
1 // String literals
2 'foo'
3 "bar"
4
5 // Array literal
6 [1954, 1974, 1990, 2014]
7
8 // Object literals
9 { a: 'foo', b: 'bar', c: 42 }
```

# Gramatyka i słowa kluczowe

## Gramatyka

Możemy użyć operatora **typeof** aby sprawdzić typ zmiennej.

## “ Zadanie 2

- *Otwórz konsolę w narzędziach deweloperskich*
- *Stwórz dowolną zmienną*
- *Sprawdź jej typ*

# Gramatyka i słowa kluczowe

## Zmienne

**Zmienna** jest synonimem pewnego obszaru pamięci, służącego do przechowywania danych.

Zmienna ma nazwę oraz wartość. Nazwa pozwala na jednoznaczną identyfikację zmiennej. Nazwa ma formę orzekającą.

# Gramatyka i słowa kluczowe

## Zmienne

Nazwa zmiennej

Wartość

*name*

*Maciej*

streetName

*Grunwaldzka*

*age*

*27*



# Gramatyka i słowa kluczowe

## Zmienne

**Deklaracja zmiennej** jest stworzeniem miejsca dla jakiejś wartości, która zostanie zapisana później.

**Inicjalizacja zmiennej** jest przypisanej danej wartości do pamięci. Dalsze modyfikowanie danej zmiennej nazywamy **przypisywaniem**.

# Gramatyka i słowa kluczowe

## Zmienne

Mamy 3 rodzaje deklaracji zmiennych

**var** - pozostałość z poprzednich wersji JS, lepiej nie używać ale koniecznie trzeba znać

**let** - nowszy i lepszy sposób na deklarowanie i inicjalizowanie zmiennych

**const** - najlepszy sposób na tworzenie zmiennych, z tym, że po użyciu tego słowa nie możemy przypisać nowej wartości do raz zainicjalizowanej zmiennej!

# Gramatyka i słowa kluczowe

## Zmienne

Jeśli nie wiesz jakiego słowa użyć, niech to będzie

**const!**

# Gramatyka i słowa kluczowe

## Zmienne

---



```
1 let x; // declaration
2 let y;
3
4 x = 10; // initialization
5 y = 10;
6
7 x = y + 2; // assignment
```

# Gramatyka i słowa kluczowe

## Zmienne - ponowne deklaracje

---



```
1 var x = 5
2 var x
3
4 alert(x) // result will be 5
```

# Gramatyka i słowa kluczowe

## Zmienne - ponowne deklaracje



```
1 // re-declaration is only possible with vars
2
3 let x = 5
4 let x // result will be Error
```

# Gramatyka i słowa kluczowe

## Zmienne - nazywanie

Zasady na nazywanie zmiennych:

- Nazwy mogą zawierać litery, cyfry, \_, \$
- Zmienne muszą zaczynać się od litery, \$ lub \_
- Zmienne są **case sensitive**
- Słowa kluczowe nie mogą być nazwami dla zmiennych

# Gramatyka i słowa kluczowe

## Zmienne - dobre praktyki

- Deklaracja zmiennych na samej górze bloku (pliku, funkcji, warunku)
- Nie deklaruj ponownie zmiennych
- Nazwy zmiennych w camelCase
- Nazywaj zmienne w taki sposób, aby było wiadomo co przechowują



## “ Zadanie 3

- *Zadeklaruj zmienną i przypisz do niej jakąś wartość*
- *Sprawdź typ tej zmiennej*

# Gramatyka i słowa kluczowe

## Truthy and falsy

Wartość “Truthy” jest wtedy kiedy wartość jest traktowana jako wartość “true” kiedy jest ewaluowana do wartości boolean (true/false).

Większość wartości jest traktowana jako truthy, chyba, że jest zdefiniowana jako falsy.

# Gramatyka i słówka kluczowe

## Truthy and falsy

### Wartości Falsy:

false,

0,

""

,

null,

undefined,

NaN.

# Gramatyka

Więcej - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical\\_grammar](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar) .

# Gramatyka i słowa kluczowe

## Gramatyka

Białe znaki nie mają znaczenia



```
1 // Lines below do the same:  
2  
3 result = a + b  
4 result=a+b  
5 result = a      +      b
```

# Gramatyka i słowa kluczowe

## Gramatyka

Koniec linii (instrukcji) powinien być zakończony znakiem “;”, jednak JS ma mechanizm automatycznego dodawania średników (ASI)

Możemy pomijać średniki

Należy uważać na znaki nowe linii.

# Gramatyka i słowa kluczowe

## Gramatyka

Zmienne są case sensitive!



```
1 const result = 10
2 const RESULT = 11
3
4 // Lines above are different variables!
```

# Gramatyka i słowa kluczowe

## Komentarze

---



```
1 // One line comment
2
3 /*
4 Multi-line comment
5 Multi-line comment
6 Multi-line comment
7 */
```



# Gramatyka i słowa kluczowe

## Komentarze

Nie powinno się traktować komentarzy jako rozwiązanie dla ułatwienia zrozumienia kodu.

Kod powinien być pisany tak, aby był zrozumiały (np. korzystając poprzez nazywanie zmiennych w przejrzysty sposób).

Pozostawienie komentarza w kodzie to ostateczność.

# Operacje matematyczne

# Operacje matematyczne

## + i - (to jedyne, czym żyje...)

---



```
1 const a = 1 + 1  
2  
3 const b = 2 - 3  
4  
5 const c = a + b
```

# Operacje matematyczne

+ i -

Uwaga! + to też znak konkatencji napisów!



```
1 const result1 = 'ala' + ' ' + 'ma kota'  
2  
3 const result2 = 'b' + 'a' + + 'n' + 'a'
```

# Operacje matematyczne

\* i /



```
1 const a = 2 / 2  
2  
3 const b = a * 2  
4  
5 const c = a * b
```

# Operacje matematyczne

## Modulo %

---



```
1 const a = 2 % 2 // result is 0
2
3 const b = 4 % 3 // result is 1
4
5 const c = 2 % 4 // result is 2
```

# Operacje matematyczne

**+=** **-=**



```
1 let a = 2
2 let b = 4
3
4 a += a // result is 4
5 b -= b // result is 0
```

# Operacje matematyczne

## Zwiększanie ++ i zmniejszanie --

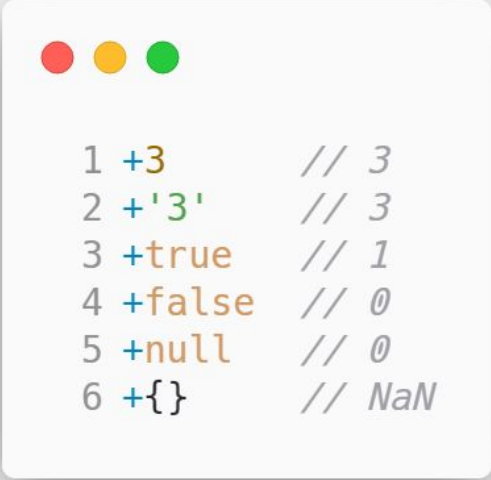
```
1 let a = 2
2 let b = 4
3
4 // Checkout on console:
5
6 a--
7 a
8 --a
9 a
10
11 b++
12 b
13 ++b
14 b
```



# Operacje matematyczne

## Unary plus (+)

---



```
1 +3      // 3
2 +'3'    // 3
3 +true    // 1
4 +false   // 0
5 +null    // 0
6 +{}      // NaN
```

# Obiekty

# Obiekty

## Obiekty w JavaScript

**Obiekt** jest zbiorem **własności**, i własność jest powiązaną nazwą i wartością. (klucz-wartość)

Obiekty mogą przechowywać wszystkie poznane typy danych.

Jeżeli typem własności jest **funkcja**, nazywamy taką własność **metodą**.

Przeglądarka ma wiele obiektów już zdefiniowanych, jednak ty możesz tworzyć swoje.

# Obiekty

## Obiekty w JavaScript

Własność obiektów może być tłumaczona jako zmienna przypisana to danego obiektu.

Możemy dostać się do własności przez metodę kropkową:

```
objectName.propertyName
```

Lub z nawiasami:

```
objectName[ 'propertyName' ]
```

# Obiekty

## Tworzymy pierwszy obiekt



```
1 const myCar = {} // or new Object()  
2  
3 // dot notation  
4 myCar.make = 'Ford';  
5 myCar.model = 'Mustang';  
6 myCar.year = 1969;  
7  
8 // square brackets notation  
9 // these 3 lines has the same effect as 3 above  
10 myCar['make'] = 'Ford'  
11 myCar['model'] = 'Mustang'  
12 myCar['year'] = 1969
```

# Obiekty

## Tworzymy pierwszy obiekt



```
1 // You can read properties same way
2
3 // dot notation
4 myCar.make // 'Ford'
5 myCar.model // 'Mustang'
6 myCar.year // 1969
7
8 // square brackets
9 myCar['make'] // 'Ford'
10 myCar['model'] // 'Mustang'
11 myCar['year'] // 1969
```

# Obiekty

## Tworzymy pierwszy obiekt



```
1 // square brackets notation
2 // has a little advantage
3 // it can use nroperty names
4 // from variables
5
6 const property1 = 'make'
7 const property2 = 'model'
8 const property3 = 'year'
9
10 myCar[property1] // 'Ford'
11 myCar[property2] // 'Mustang'
12 myCar[property3] // 1969
```

## “ Zadanie 4

*Stwórz obiekt opisujący twój  
wymarzony samochód lub dom.  
Wydrukuj wszystkie własności  
do konsoli*



# Obiekty

## Literał obiektowy



```
1 const myCar = {  
2     make: 'Ford',  
3     model: 'Mustang',  
4     year: 1969  
5 }
```

## “ Zadanie 5

*Stwórz obiekt z poprzedniego  
slajdu i wydrukuj wszystkie  
wartości.*

# Obiekty

## Obiekty zawierające obiekty i funkcje

```
1 const myCar = {  
2   make: 'Ford',  
3   model: 'Mustang',  
4   year: 1969,  
5   parts: {  
6     engine: [...],  
7     body: [...],  
8     ...  
9   },  
10  sound: function () { alert('Wrrrrr!') }  
11 }
```



# Listy

# Listy

## Deklaracje



```
1 const a = [] // these are arrays with the same content
2 const b = new Array() // and length 0
3
4 const c = ['a', 'b'] // these are arrays with the same content
5 const d = new Array('a', 'b') // and length 2 - with 2 strings
6
7 const e = [3], // two completely different arrays
8 const f = new Array(3) // one with length 1 and one with 3
```

# Listy

## .length

Własność **length** zwraca ilość elementów w danej tablicy.



```
1 const array = [1, 2, 3]
2
3 array.length // 3
4
5 array.length = 2
6
7 console.log(array)
```

# Listy

## Iterowanie po elementach listy

---



```
1 const numbers = [1, 2, 3, 4, 5];  
2  
3 const length = numbers.length  
4  
5 for (let i = 0; i < length; i++) {  
6   console.log(numbers[i])  
7 }
```

## “ Zadanie 6

*Stwórz tablicę zawierającą 1000 losowych wartości.*

*Spróbuj użyć metody obiektu Math (Math.random()).*



# Listy

## pop()



```
1 // The pop() method removes the last element from an array and returns that element.  
2  
3 //This method changes the length of the array.  
4  
5  
6 const array = [1, 2, 3]  
7  
8 array.pop()  
9  
10 console.log(array) // [1, 2]
```

# Listy push()



```
1 // The push() method adds one or more elements to the end of an array
2 // and returns the new length of the array.
3
4 const numbers = [1, 2, 3]
5 numbers.push(4)
6
7 console.log(numbers) // [1, 2, 3, 4]
8
9 numbers.push(5, 6, 7)
10
11 console.log(numbers) // [1, 2, 3, 4, 5, 6, 7]
```

# Listy shift()



```
1 // The shift() method removes the first element from an array and returns that element.  
2 // This method changes the length of the array.  
3  
4  
5 const a = [1, 2, 3]  
6 const b = a.shift()  
7  
8 console.log(a) // [2, 3]  
9 console.log(b) // 1
```

# Listy unshift()



```
1 // The unshift() method adds one or more elements to the beginning of an array
2 // and returns the new length of the array.
3
4
5 const array = [1, 2, 3]
6 array.unshift(4, 5)
7
8 console.log(array) // [4, 5, 1, 2, 3]
```



## ***Zadanie 7***

*Stwórz tablice z pięcioma losowymi wartościami.*

*Wydrukuj pierwszą i trzecią wartość. Zwiększ drugą wartość o 2.*

# Listy

## slice()

Metoda **slice()** zwraca nową tablice z wybranymi wartościami (zależą od przekazanych argumentów)



```
1 const array = ['zero', 'one', 'two', 'three']
2 const sliced = array.slice(1, 3)
3
4 console.log(array)           // ['zero', 'one', 'two', 'three']
5 console.log(sliced)         // ['one', 'two']
```

# Listy

## indexOf()

Metoda `indexOf()` zwraca pozycję pierwszego znalezionej elementu w liście. Jeżeli takiego elementu nie ma - zwracane jest `-1`.

```
1 const array = [2, 9, 9]
2
3 array.indexOf(2)           // 0
4 array.indexOf(7)           // -1
5 array.indexOf(9)           // 1
6 array.indexOf(9, 2)         // 2
7 array.indexOf(2, -1)        // -1
8 array.indexOf(2, -3)        // 0
```

## “ Zadanie 8

*Znajdź 9 w [1,2,3,4,5,6,7,8,9,10]*



# Porównania i operatory logiczne

# Porównania i operatory logiczne

Operator `==` stara się porównać dwie wartości nawet jeśli nie są one tego samego typu.

Jeżeli po obu stronach operatora występuje obiekt, JS sprawdza czy obiekty mają tę samą referencję (wskazują na to samo miejsce w pamięci)

# Porównania i operatory logiczne

## Porównanie ==

```
1 1 == 1 // true
2 '1' == 1 // true
3 1 == '1' // true
4 0 == false // true
5
6 const object1 = { key: 'value' }
7 const object2 = { key: 'value' }
8
9 object1 == object2 // false
10
11 NaN == NaN // false - WTF ??
```

Tabela równości

# Porównania i operatory logiczne

## Nierówność !=

Odwrotność sprawdzania równości.

Sprawdza czy wartości są **NIE** równe.

# Porównania i operatory logiczne

## Równość (strict inequality) ===

Operator sprawdza czy wartości są równe, jednak bez dodatkowego zmiany typów, aby była możliwość porównania.

```
1 3 === 3 // true
2 3 === '3' // false
3
4 const object1 = { 'value': 'key' }
5 const object2 = { 'value': 'key' }
6
7 object1 === object2 // false
```

# Porównania i operatory logiczne

## Nierówność (strict equality) !==

Działa dokładnie odwrotnie do znaku ===

# Porównania i operatory logiczne

## Znaki większości/mniejszości

Do sprawdzenia pewnych relacji pomiędzy danymi mogą posłużyć nam znaki większości i mniejszości

- <      operator mniejszości
- >      operator większości
- <=     operator mniejszości bądź równości
- >=     operator większości bądź równości

## “ Zadanie 10

*Porównaj różne rodzaje  
zmiennych różnych typów.  
Sprawdź kiedy warto użyć == a  
kiedy nie.*



# Porównania i operatory logiczne

## OR, AND, negation

Operator OR (lub): `||`

Operator AND (i): `&&`

Operator zaprzeczenia: `!`

# Porównania i operatory logiczne

## OR, AND, negation

---



```
1 true || false // true
2
3 true && false // false
4
5 !true // false
```

# Porównania i operatory logiczne

## OR, AND, negation

JS zawsze ewaluuje dane od lewej do prawej. W poniższych przypadkach, prawa strona wyrażenia nie zostanie wykonana.



```
1 false && (anything) // is short-circuit evaluated to false  
2 true || (anything) // is short-circuit evaluated to true
```

# Porównania i operatory logiczne

## OR, AND, negation



```
1 const a = 1
2
3 console.log(a || b) // b is not defined - we should get an error!
```

# Porównania i operatory logiczne

## OR, AND, negation

Operatory OR i AND zwracają pierwszą wartość, która jest **TRUTHY**.



```
1 const a = false || 1           // a is assigned 1
2 const a = 0 || 2               // a is assigned 2
3 const a = 0 || false || 3     // a is assigned 3
```

## “ Zadanie 11

*Stwórz zmienna, która jest falsy.  
Stwórz kolejna zmienną i  
przypisz jej pierwszą zmienna  
OR 1.  
Zweryfikuj wynik*

# JavaScript

# Szybkie przypomnienie

## Obiekt

```
const Car = {  
  name: "Samochodzik",  
  model: "Golf",  
  engine: "1.8"  
}
```



# Szybkie przypomnienie

## Tablica

```
const pokemons = ["Bulbasaur", "Ivysaur", "Venusaur"];
```

# Szybkie przypomnienie

## Operator porównania

==

# Szybkie przypomnienie

## Prawda czy fałsz?

True False

**== VS ===**

1 == 1



# TRUE

**== VS ===**

1 == `1`

TRUE

== VS ===

1 === `1`



**FALSE**

**== VS ===**

**1 !== `1`**

TRUE

**== VS ===**

**1 !== `1`**

**FALSE**

**== VS ===**

``1,2,3` == [1,2,3]`

TRUE

**== VS ===**

`[1,2,3] === [1,2,3]`



**FALSE**

**== VS ===**

`[1,2,3] == [1,2,3]`

**FALSE**

# Truthy

`1`

[1,2,3]

1

{}

function() {}

# Falsy

``

0

null

undefined

NaN

Z definicji, prawdziwe są wszystkie wartości, z wyjątkiem tych, które są zdefiniowane jako Falsy

<https://developer.mozilla.org/pl/docs/Glossary/Truthy>

# Warunki i wyrażenia

# Warunki i wyrażenia

## Instrukcja vs wyrażenie

**Instrukcja (Statement)** jest kawałkiem kodu, który wykonuje pewne akcje.

**Wyrażenie (Expression)** jest fragmentem, który coś zwraca lub tworzy.

# Warunki i wyrażenia

## Instrukcja vs wyrażenie

Kiedy chcemy uzależnić nasz kod od pewnych zmiennych, możemy wykonać różne fragmenty kodu. Do tego możemy użyć instrukcji warunkowej **IF**




```
1 if (condition1) statement1
2
3 if (condition1){
4     statement1
5     statement2
6 }
```

# Warunki i wyrażenia

## Instrukcja vs wyrażenie

Oczywiście możemy budować duże instrukcje (ale to nie jest zalecane)



```
1 if (condition1) {  
2     statement1  
3 } else if (condition2) {  
4     statement2  
5 } else if (condition3) {  
6     statement3  
7     statement4  
8 } ...  
9 else {  
10     statementN  
11 }
```



## Zadanie 12

“

*Stwórz zmienną.*

*Napisz instrukcję warunkową IF,*

*Która wydrukuje:*

*1 kiedy zmienna jest równa 1*

*2 kiedy zmienna jest równa 2*

## Zadanie 13

“

*Stwórz zmienną.*

*Napisz instrukcję warunkową IF,*

*Która wydrukuje:*

*1 kiedy zmienna jest równa 1*

*2 kiedy zmienna jest równa 2*

*3 kiedy zmienna jest równa 3*

*‘Error’ dla każdego innego przypadku*

# Warunki i wyrażenia

## Switch

Kiedy chcemy wykonać różne akcje uzależnione od wielu warunków możemy użyć metody switch.

```
1 switch(expression) {  
2     case n:  
3         code  
4     case n:  
5         code  
6         break;  
7     default:  
8         code  
9 }
```

## Zadanie 14

“

*Stwórz zmienną.*

*Napisz instrukcję warunkową **switch**,*

*Która wydrukuje:*

*1 kiedy zmienna jest równa 1*

*2 kiedy zmienna jest równa 2*

*3 kiedy zmienna jest równa 3*

*‘Error’ dla każdego innego przypadku*

# Warunki i wyrażenia

## Ternary operator (conditional expression)

Jeśli *condition* jest true, operator zwraca wartość *expr1*; w przeciwnym razie zwraca wartość *expr2*



```
1 condition ? expr1 : expr2
```

# Warunki i wyrażenia

## Ternary operator (conditional expression)



```
1 const expressionToCheck = true
2
3 console.log(expressionToCheck ? 'This is true!' : 'This is false!')
4
5 const result = expressionToCheck ? 'This is true!' : 'This is false!'
6
7 console.log(result)
```

# Warunki i wyrażenia

## Ternary operator (conditional expression)



```
1 // We can use multiple ternary operators together:
2
3 const firstCheck = false
4 const secondCheck = false
5
6 const access = firstCheck ?
7   'Access denied'
8   :
9   secondCheck ?
10    'Access denied'
11    :
12    'Access granted'
13
14 console.log(access) // 'Access granted'
```

# Funkcje



# Funkcje

## Jak działają funkcje w JS?

Funkcja to kawałek kody, który możemy wywołać w określonym przez nas momencie.

Kiedy funkcja jest wywoływana, argumenty są przekazywane do funkcji. Funkcja może nam coś zwrócić lub nie.

Funkcja jest obiektem.

Funkcja może być przypisana do zmiennej i przekazana jako parametr.

# Funkcje

## Jak działają funkcje w JS?



```
1 function name(param1, param2, ...) {  
2     statement // <- this is function body  
3     statement // <- this is function body  
4 }
```

**name** - nazwa funkcji

**param** - argumenty funkcji

**statements** - ciało funkcji

# Funkcje

## Jak działają funkcje w JS?



```
1 // An anonymous function is a function without a function name:
2 function (param1, param2, ...) { ...statements }
3
4 // A named function is a function with a function name:
5 function foo(param1, param2, ...) { ...statements }
6
7 // * An arrow function expression has a shorter syntax than a function expression
8 (param1, param2, ...) => { ...statements }
9
10 // * arrow functions was introduced in ES6, and works in modern browsers now
```

# Funkcje

## Jak działają funkcje w JS?



```
1 // Declaring named function:
2 function addOne (a) {
3     return a + 1
4 }
5
6 // Declaring anonymous function and assigning it to a variable:
7 const addOne = function(a) {
8     return a + 1
9 }
10
11 // Using arrow function
12 const addOne = a => a + 1
```

# Funkcje

## Return



```
1 function addOne (a) {  
2   return a + 1  
3   console.log('2') // will not be executed  
4 }
```

**Return kończy działanie funkcji i zwraca wartość działania tej funkcji**

Jeżeli nie ma słowa return, funkcja zwraca undefined

## “ Zadanie 15

*Stwórz funkcję dodającą dwie liczby*

# Funkcje

## Zakres

Zakres definiuje dostępność (widoczność) zmiennych.

W JavaScript mamy dwa typy zakresów

- Lokalny (funkcja lub blok)
- Globalny

W JS głównie spotykamy zakres funkcji (lokalny) tworzony przez każdą funkcję. Zmienne zadeklarowana w środku funkcji nie jest widoczna poza funkcją

W ES6 został wprowadzony zakres blokowy (if)

# Funkcje

## Zakres lokalny

---




```
1 // no variable sum here
2
3 function add(a, b) {
4     const sum = a + b
5     // we can use sum here !
6     return sum
7 }
8
9 // no variable sum here
```



# Funkcje

## Zakres globalny



```
1 let sum
2 // we can use sum here !
3
4 function add(a, b) {
5     sum = a + b; // and here
6     return sum
7 }
8
9 // and here
```

Wszystkie globalne zmienne mają zakres globalny, więc są dostępne w każdej funkcji

# Funkcje

## Wywoływanie funkcji

Kod ze środka funkcji jest wykonywany tylko wtedy kiedy funkcja jest wywoływana “()”.

```
1 function add(a, b) {  
2     const sum = a + b  
3     return sum  
4 }  
5  
6 add(10, 2) // Will return 12  
7 window.add(10, 2) // same as above
```

# Funkcje

## Deklaracje

```
1 // function declaration - declares a function that name is accessible in the scope
2 function example() {
3     return 3
4 }
5
6 //anonymous function expression
7 const variable = function() {
8     return 3
9 }
10
11 //named function expression - the name is only local to the function scope (inside body)
12 const variable = function example() {
13     return 3
14 }
```

# Funkcje

## Dodatkowe informacje

W JavaScript, zakres globalny jest kompletnym środowiskiem JavaScript.  
W przeglądarce zakresem globalnym jest obiekt **window**.

Wszystkie zmienne stworzone z użyciem **var** mogą być znalezione w obiekcie **window**, jako własność tego obiektu

Cykl życia zmiennej zaczyna się kiedy jest ona zadeklarowana.  
Zmienna lokalna jest usuwana kiedy funkcja się kończy.  
Zmienna globalna jest usuwana kiedy zamyka się przeglądarka (Tab).

# Funkcje

## IIFE (Immediately Invoked Function Expression)

IIFE jest funkcją, która wykonuje się zaraz po zadeklarowaniu.



```
1 (function(a, b) {  
2     const sum = a + b  
3     return sum  
4 })(10, 2) // Will return 12
```

## Zadanie 16

“

*Stwórz funkcję, która dodaje 3 liczby.  
Zwróć je.*

*Spraw, aby funkcja sama się  
wywołała.*

# Funkcje

## Funkcje wewnętrzne

Funkcja wewnętrzna jest funkcja, która jest zadeklarowana wewnątrz  
Funkcja wewnętrzna nie jest dostępna z zewnątrz.

```
1 function addSquares(a,b) {  
2     function square(x) {  
3         return x * x  
4     }  
5  
6     return square(a) + square(b)  
7 }
```

## Zadanie 17

“

*Stwórz funkcję, która zwiększa przekazaną do niej liczbę o 1, jeśli ta liczba jest ujemna,*

*Lub zmniejszą tę liczbę, jeśli ta jest dodatnia.*

*Zawsze zwracaj zmodyfikowaną wartość*



# Funkcje

## Rekurencja

Funkcje rekurencyjne są to Funkcję, które wywołują same Siebie.



```
1 function loop(x) {  
2     console.log(x)  
3  
4     if (x >= 10){  
5         return  
6     }  
7  
8     loop(x + 1)  
9 }
```

# Funkcje

## Przez wartość i referencje

W JS typu proste przekazywane są przez wartość. Natomiast obiekty przez referencje.

```
function modifyMember(reference) {  
    reference.foo = "bar"  
}
```

```
var obj = { foo: "foo" }  
modifyMember(obj)  
console.log(obj) // { foo: "bar" }
```

# Funkcje

## Przez wartość i referencje

W JS typu proste przekazywane są przez wartość. Natomiast obiekty przez referencje.

```
function modifyString(stringValue) {  
    stringValue = "new string value";  
    console.log("stringValue", stringValue);  
}
```

```
const someStringValue = "value 123";
```

```
modifyString(someStringValue);  
console.log(someStringValue);
```