



Krakowska Akademia
im. Andrzeja Frycza Modrzewskiego

Krakowska Akademia im Andrzeja Frycza Modrzewskiego

Programowanie w Javascript – wprowadzenie

Autor: Marcin Dłubis



Spis treści

1. Wstęp	4
2. Typy danych i operatory	5
2.1 var & let & const	6
2.2 Typy wartości:	7
2.2.1 Typy proste	7
2.2.2 Typy złożone**	8
3. Operatory	9
Operatory arytmetyczne	9
3.1 Przypisanie	10
3.2 Operatory porównania	10
3.3 Operatory logiczne	10
3.4 Inkrementacja, dekrementacja	11
3.5 Instrukcje warunkowe	11
3.5.1 instrukcja warunkowa If	11
3.5.2 Instrukcja warunkowa switch	12
4. Pętle	14
4.1 Pętla for	14
4.2 Pętla while	14
4.3 Pętla do while	15
Różnica pomiędzy pętlami	15
Zadania	15
5. Funkcje	16
5.1 Zmienna lokalna a zmienna globalna	16
5.2 Zwracanie wartości	17
5.3 Parametry funkcji	18
5.4 Zakres zmiennych	19
5.5 Argumenty i parametry	19
5.6 Funkcje anonimowe	20
5.7 Wbudowane funkcje JS	20
5.8 Zadania	20
6. Obiekty	20
6.1 Wstęp	21
6.2 Dodanie właściwości	21
6.3 Zadania	22
7. Tablice	24



7.1 Wstęp	24
7.2 Tablice wielowymiarowe	24
7.3 Metody tablicowe	25
7.3 Iterowanie po tablicy	26
7.4 Zadania.....	27

1. 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 działanie poprzez umieszczenie w zewnętrznym 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.

2. 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.

Dobłą 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.

Dobłą 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 - „przykład”
- 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

2.1 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:

Bezpieczniej 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 - „przykład”
- 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

2.2 Typy wartości:

Przykłady zmiennych wraz z przypisaniem:

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

2.2.1 Typy proste

- number - np. 100, 0.92, NaN
- string - np. 'studia podyplomowe', 100
- boolean - true, false



- undefined
- null /* symbol - np. nazwa()

2.2.2 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);
```


3. Operatory

Operatory to symbole, które reprezentują pewną operację.

Operatory możemy podzielić na kilka kategorii:

- arytmetyczne
- przypisania
- porównania
- logiczne

Operatory arytmetyczne

- dodawanie
- odejmowanie
- mnożenie
- dzielenie
- konkatencja
- 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 konkatencją, 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!

3.1 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
```

3.2 Operatory porównania

Rodzaje operatorów:

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

3.3 Operatory logiczne

Rodzaje operatorów logicznych:

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

Przykład 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"); }
```

3.4 Inkrementacja, dekrementacja

Inkrementacja - zwiększenie

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

Dekrementacja - zmniejszenie

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

Zadanie

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

3.5 Instrukcje warunkowe

3.5.1 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

3.5.2 Instrukcja warunkowa `switch`

```
let Number = 5;
```

```
switch (Number) {  
  case 5:  
    console.log("Szczęśliwa liczba to 5 ");  
    break;  
  case 30:  
    console.log("Pechowa liczba");  
    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.

4. Pętle

Pętla to struktura pozwalająca na wykonywanie czynności do momentu osiągnięcia warunku 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
```

4.1 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);
}
```

4.2 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");
```

4.3 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

5. 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. zwyczajowo 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");  
};
```

5.1 Zmienna lokalna a zmienna globalna

Przykład

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




```
console.log(przykladowy_tekst); };  
  
saySomething();
```

5.2 Zwracanie wartości

Utwórz funkcję i otwórz konsolę:

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

Otrzymasz:

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

Aby funkcja coś zwracała:

```
function saySomething() { return "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;
```

Działający przykład

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

5.3 Parametry funkcji

Parametry funkcji to nazwy zmiennych pisane we wnętrzu nawiasów okrągłych, rozdzielonych przecinkiem. Są niezwykle ważne w kontekście dynamizowania kodu.

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

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 w nawiasie argumenty

Zakres zmiennych

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

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

Jeżeli nie podasz wszystkich parametrów, to niepodany parametr traktowany jest `undefined`!

Jeżeli podasz więcej parametrów niż jest argumentów, to funkcja skorzysta tylko z tych, które faktycznie są wykorzystywane.

Przykład

```
function sumowanie(a,b,){
```

```
    return a+b  
};
```

sumowanie (1,2,3,4) dalej zwróci liczbę 3 (suma 1+2).

Czy parametry są przekazywane poprzez kopiowanie czy referencje?

Typy prymitywne są przekazywane do funkcji poprzez kopiowanie! Obiekty są przekazywane do funkcji poprzez referencje!

5.4 Zakres zmiennych

```
function saySomething() {  
    wiadomosc = "Studia podyplomowe";  
    alert(wiadomosc);  
}  
saySomething();
```

Sposób na wyświetlenie wartości zmiennej wewnętrznej: w konsoli sprawdź wyświetlanie zmiennych `yourName` oraz `yourSurname`. Dzięki temu zmienna ta, pomimo że jest wewnątrz funkcji, stała się zmienną globalną!

Aby się do niej odwołać, w konsoli wpisz: `console.log(window.wiadomosc);`

TIPS&TRICKS:

W języku JavaScript istnieje `garbageCollector` - dzięki niemu, gdy wewnętrzna zmienna funkcji zostanie wykorzystana to nie musimy się martwić o usunięcie zmiennej i zwolnieniu pamięci, tylko `garbageCollector` sam zadba o usunięcie jej z pamięci.

5.5 Argumenty i parametry

Spórz na standardowy przykład:

```
function sumowanie(a,b) {  
    return a+b;  
};  
  
sumowanie(1,2);
```

zmienna arguments

```
function sumowanie(a,b) {  
    console.log(arguments)  
    return a+b;  
};  
  
sumowanie(1,2);
```

Jak widzisz, zmienna `arguments` pokazuje wartości przekazane do funkcji. Nie jest to tablica, a obiektem tablicopodobnym (ang. *array-like*).

Tablicopodobnym bo nie możemy skorzystać z takich metod tablic jak np. `pop` czy `push` (ale już z `length` - tak).

5.6 Funkcje anonimowe

```
const exampl = function(a,b) {  
    return a+b;  
};
```

Funkcje anonimowe przydają się, gdy raz chcemy przekazać jakąś funkcję i nie chcemy jej przekazać w innych miejscach.

Przykład zwykłej funkcji, która nie jest anonimowa:

```
function saySomething(imie,nazwisko) {  
    const dane = imie + nazwisko;  
    return dane;  
};  
  
console.log(saySomething("Twoje dane to: ", "Marcin));
```

Zamienienie jej na funkcję anonimową

```
function saySomething(imie,nazwisko) {  
    const dane = imie + nazwisko();  
    return dane;  
};  
  
console.log(saySomething("Twoje dane to: ", "Marcin));
```

5.7 Wbudowane funkcje JS

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

5.8 Zadania

1. Kalkulator podatku VAT
2. Sprawdź, czy liczba jest parzysta

6. Obiekty

6.1 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.

6.2 Dodanie właściwości

```
let person = {  
  "my name": "Marcin",  
  "my age": 28,  
  occupation: "inżynier"  
};  
  
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); */  
} };
```

6.3 Zadania

Zadanie 1.

Utwórz funkcję, która przyjmuje rozmiary pudełka (zawiera klucze szerokości, długości, wysokości) i zwraca objętość pudełka.

```
volumeOfBox({ width: 2, length: 5, height: 1 }) → 10  
volumeOfBox({ width: 4, length: 2, height: 2 }) → 16  
volumeOfBox({ width: 2, length: 3, height: 5 }) → 30
```

Zapisz to zadanie za pomocą funkcji standardowej oraz funkcji strzałkowej: <https://codepen.io/MarcinDI/full/zbjeoN>

Pamiętaj: nie stosuj funkcji strzałkowej przy tworzeniu metod! Wy tłumaczenie poniżej:

W tym wypadku `this` kieruje nas na obiekt `boxSizes` (sprawdź to w konsoli przeglądarki wpisując `console.log(boxSizes)`).

```
const boxSizes = {  
  width: 10,  
  height: 20,  
  length: 10,  
  volume: function() {  
    console.log(this)  
    return this.width*this.length*this.height  
  }  
}
```

W tym wypadku `this` kieruje nas na obiekt globalny `Window` (sprawdź to w konsoli przeglądarki wpisując `console.log(boxSizes)`), dlatego rozwiązanie otrzymuje wartość `undefined`.

```
const boxSizes = {  
  width: 10,  
  height: 20,  
  length: 10,  
  volume: () => {  
    console.log(this)  
    this.width*this.length*this.height  
  }  
}
```

Zapisz to zadanie za pomocą funkcji standardowej oraz funkcji strzałkowej.

Zadanie 2.



Napisz funkcję, która przyjmuje dwa parametry: liczbę oraz obiekt z wartościami min i max. Sprawdź, czy podana liczba zawiera się w przedziale min do max.

```
isInRange(4, { min: 0, max: 5 }) → true  
isInRange(4, { min: 4, max: 5 }) → true  
isInRange(4, { min: 6, max: 10 }) → false  
isInRange(5, { min: 5, max: 5 }) → true
```

Zapisz to zadanie za pomocą funkcji standardowej oraz funkcji strzałkowej.

7. Tablice

7.1 Wstęp

Tablice, z ang. *arrays*, to zbiór danych, np:

```
const arr = [1,2,5,932,1]

const obj = {
  country: "Poland",
  city: "Katowice"
}

const arr2 = [1*8, "dwa", 1988, {name:"Marcin",family:"yes"}, obj]
```

Tablice są indeksowane, tj. ich indeks zaczyna się od 0, dalej 1,2,3 itp. Tablice są obiektami, a więc posiadają właściwości i metody! W przypadku tablic istnieją z góry predefiniowane metody - jak np. długość tablicy (o czym za chwilę).

Odwołanie się do elementów tablicy:

```
arr[1] //console.log(arr[1]) - element drugi o indeksie jeden z tablicy
arr1
arr[3].name //console.log(arr[3].name) - właściwość name w elemencie
czwartym, który ma indeks równy trzy
```

Nadanie wartości poszczególnym elementom tablicy:

```
arr[0] = "nowa wartosc";
arr[1] = 12;
arr2[3].name = "Tomasz"
```

7.2 Tablice wielowymiarowe

```
const tablica_wielowymiarowa = [[5,3,1],["on","ona"],["ono"]]
```

Odwołanie w konsoli:

```
tablica_wielowymiarowa[1][3][0]
```

Ale spójrz na to - jaka jest różnica pomiędzy:

```
tablica_wielowymiarowa[1][3]
tablica_wielowymiarowa[1][3][0]
```


odpowiedź: typ danych [object vs. string]

7.3 Metody tablicowe

```
const tablica = [];  
tablica[0] = "Pierwszy element";
```

Tablice w JS tworzone są dynamicznie, tj. nie trzeba tworzyć elementów tablicy po kolei, można po stworzeniu pierwszego elementu stworzyć dziesiąty.

```
const tablica = [];  
tablica[0] = "Pierwszy element";  
tablica[10] = "Drugi element";
```

Dodanie elementu do tablicy

```
tablica.push("Ostatni element");
```

Ograniczenie wielkości tablicy

Aby ograniczyć wielkość tablicy:

```
const nazwa_tablicy = [1, 2, 3, 4, 5, 6];  
nazwa_tablicy.length = 4;
```

Usunięcie wartości elementu z tablicy

```
const nazwa_tablicy = [1, 2, 3, 4, 5, 6];  
delete nazwa_tablicy[2];
```

Usunięcie ostatniej wartości tablicy

```
nazwa_tablicy.pop();
```

Usunięcie pierwszego elementu tablicy

```
nazwa_tablicy.shift();
```

Usunięcie więcej niż jednego elementu tablicy

```
const nowa_tablica = [1, 2, 3, 4, 5, 6];  
nowa_tablica.splice(3, 2); //Usuwanie od początku tablicy  
nowa_tablica.splice(-2, 2); //Usuwanie od końca tablicy
```

Zapamiętanie usuwanych elementów

```
const zmienna1 = nowa_tablica.pop();  
const zmienna2 = nowa_tablica.unshift();
```

Lista metod

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

7.3 Iterowanie po tablicy

Bardzo często tablice wykorzystywane są wspólnie z pętlami.

ZADANIE:

wyświetl liczby: 6, 7, 8, 23, 34, 43, 56, 50 w następujący sposób:

```
Wylosowana liczba to: 6  
Wylosowana liczba to: 7  
Wylosowana liczba to: 8  
Wylosowana liczba to: 23  
itd
```

Sposób 1

```
const losowanie = [6, 7, 8, 23, 34, 43, 56, 50],  
const wyswietlany_tekst = "Liczba to: ";  
  
for (let i = 0; i < losowanie.length; i++) {  
    console.log(wyswietlany_tekst + losowanie[i]);  
}
```

Sposób 2

```
const losowanie = [6, 7, 8, 23, 34, 43, 56, 50],  
const wyswietlany_tekst = "Liczba to: ";  
  
for (let item of losowanie) {  
    console.log(wyswietlany_tekst + item);  
}
```



```
}
```

7.4 Zadania

Zadanie 1

Wygeneruj 10-elementową tablicę liczb losowych, a następnie wypisz te liczby w konsoli.

Zadanie 2

Utwórz funkcję tworzącą tablicę 10-elementową z wylosowanymi liczbami. Następnie napisz nową funkcję, która przyjmuje nowoutworzoną tablicę jako parametr i zwraca tylko wartości parzyste.

```
noOdds([1, 2, 3, 4, 5, 6, 7, 8]) → [2, 4, 6, 8]  
noOdds([43, 65, 23, 89, 53, 9, 6]) → [6]  
noOdds([718, 991, 449, 644, 380, 440]) → [718, 644, 380, 440]
```