

Wykład

Część 3 – Instrukcje sterujące



Instrukcje wyboru

```
if (warunek) {  
    instrukcje;  
} else {  
    instrukcje;  
}
```

(warunek)? wartość1:wartość2

```
switch (zmienna){  
    case wartość Stała1 : instrukcje;break;  
    case wartość Stała2 : instrukcje;break;  
    //.....  
    default:instrukcje;  
}
```

Pętle



```
while (warunek) {  
    instrukcje  
}
```

```
do {  
    instrukcje;  
} while (warunek);
```



for dla tablic (przechodzi przez wszystkie elementy tablicy):

```
// Przykładowa tablica
let liczby = [1, 2, 3, 4, 5];

// Instrukcja for iterująca przez tablicę
// i wypisująca jej zawartość

for (let i = 0; i < liczby.length; i++) {
  console.log(liczby[i]);
// Wyświetlenie kolejnych elementów tablicy
}
```



Wersja pętli for przeznaczona od obsługi tablic (przechodzi przez wszystkie elementy tablicy):

```
// Pętla for...of iterująca przez tablicę  
i wypisująca jej zawartość  
  
for (const liczba of liczby) {  
    console.log(liczba);  
    // Wyświetlenie kolejnych elementów tablicy  
}
```



Pętle

Wersja pętli for dla obiektów (przechodzi przez wszystkie właściwości (pola) obiektu:

```
// Przykładowy obiekt
```

```
let samochod = {  
  marka: "Toyota",  
  model: "Corolla",  
  rokProdukcji: 2020,  
  kolor: "czarny",  
};
```

```
// Pętla for...in iterująca przez właściwości obiektu  
i wypisująca ich nazwy i wartości
```

```
for (const klucz in samochod) {  
  console.log(klucz + ":" + samochod[klucz]);  
  // Wyświetlenie nazwy właściwości i jej wartości  
}
```

Część 4 – Funkcje

Funkcje



W języku JavaScript funkcje mogą być tworzone na kilka różnych sposobów.

Deklaracja funkcji (funkcje deklarycyjne):

```
function nazwaFunkcji(parametr1, parametr2) {  
    // Ciało funkcji  
    return wynik;  
    // Opcjonalne, jeśli chcesz zwrócić wartość  
}
```

Funkcje deklarycyjne są podniesione (hoisted), co oznacza, że można je wywołać przed ich faktycznym zadeklarowaniem w kodzie.

Są czytelne i łatwe do debugowania ze względu na to, że posiadają nazwę. Nadają się do tworzenia funkcji, które są używane w różnych miejscach w kodzie.



Funkcje

arguments to obiekt, który jest dostępny we wszystkich funkcjach i zawiera wszystkie przekazane do funkcji argumenty w postaci tablicopodobnej. Jest to użyteczne, gdy nie znamy liczby argumentów przekazanych do funkcji lub chcemy stworzyć funkcję, która może przyjmować różną liczbę argumentów.

```
function showArguments() {  
  for (let i = 0; i < arguments.length; i++) {  
    console.log(arguments[i]);  
  }  
}
```

```
showArguments(1, 2, 3); // Wyświetli: 1, 2, 3
```

Mimo że obiekt arguments zachowuje się jak tablica, nie jest on tablicą. Nie posiada metod typowych dla tablic, takich jak map, filter czy reduce.

Funkcje



Wyrażenia funkcyjne:

```
let nazwaFunkcji = function (parametr1, parametr2) {  
  // Ciało funkcji  
  return wynik;  
  // Opcjonalne, jeśli chcesz zwrócić wartość  
};
```

Wyrażenia funkcyjne są przypisywane do zmiennych, co oznacza, że mogą być wywoływane tylko po zainicjowaniu zmiennej.

Mogą być używane tam, gdzie potrzebujesz funkcji jako wartości.
Przykładowo: przekazywanie ich jako argumenty do innych funkcji.



Funkcje

Funkcje strzałkowe (*ang. Arrow Functions*) zostały wprowadzone w ES6 (ECMAScript 2015) jako skrócona składnia dla funkcji anonimowych. Pozwalają one pisać bardziej zwięzły i czytelny kod, zwłaszcza w przypadku funkcji przekazywanych jako argumenty

```
let nazwaFunkcji = (parametr1, parametr2) => {  
  // Ciało funkcji  
  return wynik;  
  // Opcjonalne, jeśli chcesz zwrócić wartość  
};
```

Funkcje



```
// tradycyjna funkcja anonimowa
let suma = function (a, b) {
  return a + b;
};

// Funkcja strzałkowa
let suma = (a, b) => a + b;
```

Funkcja strzałkowa nie posiada wskaźnika `this` oraz obiektu `arguments`.

Funkcje



Przykłady składni:

```
// Jedna linia - domyślny return
const square = x => x * x;

// Funkcja bez argumentów
const sayHello = () => console.log("Hello!");

// Funkcja z blokiem kodu
const max = (a, b) => {
  if (a > b) return a;
  return b;
};
```



Najważniejsza różnica między funkcjami strzałkowymi a tradycyjnymi to sposób wiązania `this`. Funkcje strzałkowe nie mają własnego `this` – dziedziczą je z otaczającego kontekstu leksykalnego.

```
function Counter() {  
  this.count = 0;  
  setInterval(() => {  
    this.count++;  
    console.log(this.count);  
  }, 500);  
}
```

tworzy nowy obiekt (`this`) i przypisuje mu pole `count = 0`.

`setInterval()` uruchamia podaną funkcję co 500 ms. Wewnątrz tej funkcji inkrementujemy `this.count` i wypisujemy jego wartość.

Konstrukcja nie jest możliwa przy zastosowaniu zwykłej funkcji która utworzy własną instancję `this`



Ograniczenia funkcji strzałkowych:

- nie mają własnych: `this`, `arguments`, `super`, `new.target`;
- nie nadają się jako metody obiektów, jeśli `this` ma mieć znaczenie dynamiczne;
- nie mogą być używane jako funkcje konstruktora (`new MyArrowFunction()` powoduje błąd).

Część 4 – Tablice

Tablice



Tablica (array) to struktur danych przechowująca uporządkowaną kolekcję elementów. Może zawierać dane dowolnego typu, **także mieszane**.

```
const przykład = [1, "tekst", true, null];
```

W JavaScript tablica to obiekt specjalnego typu. Charakteryzuje się:

- dynamiczną długością
- numerycznym indeksem kluczowym.



Tablice

Tworzenie tablic:

```
const fruits = ["apple", "banana", "cherry"];  
const numbers = new Array(1, 2, 3); // mniej zalecane  
const empty = []; // pusta tablica  
  
let zmienna = 10; // zmienna  
const mixed = [1, "apple", true, null, zmienna]; // tablica mieszana  
  
const nested = [[1, 2], [3, 4]]; // tablica zagnieżdżona
```

Dostęp do elementów:

```
fruits[0] = "orange"; // zmiana wartości w tablicy  
nested[0][1] = 5; // zmiana wartości w tablicy zagnieżdżonej
```

Rozmiar tablicy:

```
fruits.length; // 3
```

Tablice



Tablice jako kolejka:

```
const empty = []; // pusta tablica
empty.push("orange"); // dodanie elementu do pustej tablicy
empty.pop(); // usunięcie ostatniego elementu z pustej tablicy
empty.unshift("kiwi"); // dodanie elementu na początek pustej tablicy
empty.shift(); // usunięcie pierwszego elementu z pustej tablicy
```



Iteracja po tablicy:

for `for (let i = 0; i < fruits.length; i++) {
 console.log(fruits[i]);
 }`

for.. of `for (const fruit of fruits) {
 console.log(fruit);
 }`

forEach `fruits.forEach((fruit, index) => {
 console.log(`${index}: ${fruit}`);
 });`




Przydatne metody:

<code>push(x)</code>	Dodaje element na końcu tablicy
<code>pop()</code>	Usuwa ostatni element
<code>shift()</code>	Usuwa pierwszy element
<code>unshift(x)</code>	Dodaje element na początek
<code>slice(a, b)</code>	Zwraca fragment tablicy
<code>splice(a, b)</code>	Dodaje/usuwa elementy od indeksu a
<code>map(fn)</code>	Tworzy nową tablicę przez transformację
<code>filter(fn)</code>	Zwraca elementy spełniające warunek
<code>reduce(fn)</code>	Redukuje tablicę do jednej wartości
<code>find(fn)</code>	Zwraca pierwszy pasujący element



Przykłady zastosowania metod:

```
const nums = [1, 2, 3, 4, 5];  
const squared = nums.map(n => n * n); // [1, 4, 9, 16, 25]  
const even = nums.filter(n => n % 2 === 0); // [2, 4]
```



Właśnie w takich sytuacjach
najbardziej przydatne są funkcje
strzałkowe opisywane wyżej



Przykłady zastosowania metod:

```
const numbers = [1, 2, 3, 4, 5];  
const sum = numbers.reduce((accumulator, currentValue) => {  
  return accumulator + currentValue;  
}, 0);  
console.log(sum); // → 15
```

reduce() przetwarza tablicę od lewej do prawej, agregując wartości zgodnie z funkcją przekazaną jako argument.

- accumulator – zgromadzona wartość z poprzednich kroków (tutaj: suma).
- currentValue – bieżący element przetwarzany w tablicy.
- 0 – wartość początkowa dla accumulator.



UWAGA: == i === w kontekście tablic

```
[1, 2] === [1, 2]; // false (porównanie referencji)
[1, 2] == [1, 2]; // false (porównanie referencji)
```

```
38 1. a).
```

```
39
```

```
40
```

```
41
```

```
42
```

Ten warunek zawsze będzie zwracać wartość „false”, ponieważ język JavaScript porównuje obiekty według odwołania, a nie wartości. ts(2839)

Wyświetl problem (Alt+F8) Szybka poprawka... (Ctrl+.) Poprawianie przy użyciu funkcji Copilot (Ctrl+I)

```
[1, 2] === [1, 2]; // false (porównanie referencji)
```

```
↑↓
```

```
42 [1, 2] == [1, 2]; // false (porównanie referencji)
```

To też zawsze daje FALSE, chociaż VSC nie informuje o problemie

Wynika to z faktu, jak działa porównanie obiektów (a tablice w JS to obiekty). W JavaScript tablice są obiektami, a obiekty porównywane są przez referencję, a nie przez wartość.

Część 5 – Metoda `forEach`



Metoda forEach

forEach jest metodą dostępną dla tablic (arrays) oraz niektórych innych kolekcji (jak NodeList), która pozwala wykonać określoną funkcję dla każdego elementu w kolekcji.

```
let liczby = [1, 2, 3, 4, 5];  
liczby.forEach(wypisz);  
  
function wypisz(liczba)  
{  
    console.log(liczba);  
}
```

Tej wersji parametrem metody forEach jest nazwa funkcji która wykonana będzie dla każdego elementu tablicy



Metoda forEach

Funkcję iterującą wszystkie elementy tablicy piszemy częściej jako funkcję anonimową

```
let liczby = [1, 2, 3, 4, 5];  
liczby.forEach(function (liczba) {  
    console.log(liczba);  
});
```

Lub funkcję strzałkową

```
let liczby = [1, 2, 3, 4, 5];  
liczby.forEach((liczba) => {console.log(liczba)});
```

// lub w "zminimalizowanym" zapisie

```
let liczby = [1, 2, 3, 4, 5];  
liczby.forEach(liczba => console.log(liczba));
```



Metoda forEach

Funkcja może przyjmować trzy parametry:

```
array.forEach(function(currentValue, index, array) {  
  // kod do wykonania dla każdego elementu  
});
```

- currentValue: Aktualny element przetwarzany w tablicy.
- index (opcjonalnie): Indeks aktualnego elementu.
- array (opcjonalnie): Tablica, na której metoda forEach została wywołana.

```
let liczby = [1, 2, 3, 4];  
liczby.forEach((liczba, indeks, arr) => {  
  arr[indeks] = liczba * 2;  
});
```

Uwaga – proszę to potraktować jako przykład składni.
Modyfikacja elementów tablicy w ten sposób nie jest zalecana,
lepiej użyć metody map



Metoda forEach

- ✓ Metoda `forEach` nie może być przerwana ani zakończona przedwcześnie (jak w przypadku pętli `for` lub `while` przy użyciu `break`).
- ✓ Sama metoda `forEach` nie modyfikuje oryginalnej tablicy, chyba że zrobimy to ręcznie wewnątrz funkcji.
- ✓ Metoda `forEach` zawsze zwraca `undefined`.
- ✓ Jeśli należy przerwać iterację lub zwrócić zmodyfikowaną tablicę, rozważyć należy użycie metod: `map`, `filter` lub klasycznej pętli `for`.



Metoda forEach

Różnice między forEach, map i filter

forEach:

- Zwraca zawsze undefined.
- Używana do wykonywania operacji pomocniczych (side effects), np. logowanie, modyfikacja DOM.

map:

- Zwraca nową tablicę o tej samej długości, z przetworzonymi wartościami.
- Stosuj, gdy potrzebujesz przekształcić dane i dalej je wykorzystać.

filter:

- Zwraca nową tablicę zawierającą tylko te elementy, które przeszły warunek.
- Idealna do selekcji elementów na podstawie predykatów.

Literatura:

- Negrino Tom, Smith Dori, ***Po prostu JavaScript i Ajax, wydanie VI***, Helion, Gliwice 2007.
- Lis Marcin, JavaScript, Ćwiczenia praktyczne, wydanie II, Helion, Gliwice 2007.
- http://www.w3schools.com/JS/js_popup.asp
- Beata Pańczyk, wykłady opublikowane na stronie <http://www.wykladowcy.wspa.pl/wykladowca/pliki/beatap>