



</>

JavaScript

Typy, liczby, zmienne, operatory, instrukcje kontrolne



```
document.getElementById(div).innerHTML = " " + s[i] + " " + s[i+1];  
else if (i==2)  
{  
    var atpos=i-1;  
    var dotpos=i+1;  
    if (atpos>0 && dotpos<s.length-1)  
        document.getElementById(div).innerHTML = " " + s[0:i-1] + " " + s[i+1:s.length];  
    else  
        document.getElementById(div).innerHTML = " " + s[0:i-1] + " " + s[i+1:s.length];  
    i+=2;  
}
```

O historii słów
kilka...

Najczęściej używany język...



Badania przeprowadzono na ponad 87 000 aktywnych programistów w
2023 roku

01

Typy

Typy danych w JavaScript

number

Liczby całkowite i
zmiennoprzecinkowe

string

ciągi znaków

boolean

wartości logiczne
true/false

null

oznaczenie braku
wartości

undefined

oznaczenie braku
zdefiniowanej
wartości

object

obiekty złożone,
takie jak tablice,
obiekty

Typy danych w JavaScript

```
// Przypisanie wartości liczbowej do zmiennej
```

```
let age = 27;
```

```
// Przypisanie ciągu znaków do zmiennej
```

```
let name = "Piotr";
```

```
// Przypisanie wartości logicznej do zmiennej
```

```
let isTired = true;
```

```
// Przypisanie wartości null do zmiennej
```

```
let emptyValue = null;
```

```
// Przypisanie wartości undefined do zmiennej
```

```
let unassignedValue;
```

```
// Tworzenie obiektu (tablicy) i dodawanie elementów
```

```
let colors = ["red", "green", "blue"];
```

```
colors.push("yellow");
```

```
// Wyświetlanie zawartości zmiennych
```

```
console.log(age); // 27
```

```
console.log(name); // "Piotr"
```

```
console.log(isTired); // true
```

```
console.log(emptyValue); // null
```

```
console.log(unassignedValue); //
```

```
undefined
```

```
console.log(colors); // ["red", "green",  
"blue", "yellow"]
```

02

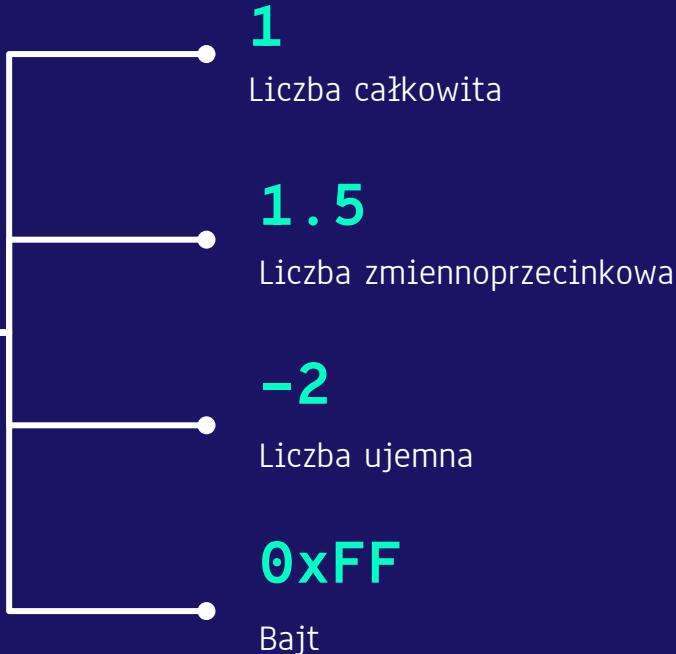
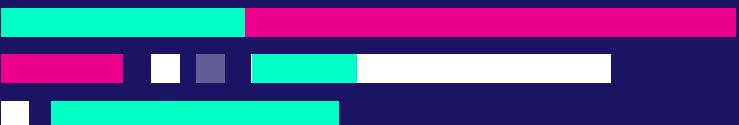
Liczby

Liczby w JS

Number

Właściwie typ *double* z C++

Używany do każdego rodzaju liczby



Konwersja na liczby

Przy próbie wartości logicznych na liczbę, JavaScript używa następujących reguł:

Wartość	Number
null	0
true	1
false	0

Jeśli chcemy przekonwertować *string* na liczbę, interpreter próbuje znaleźć liczbę złożoną z cyfr, kropki (dla zmienoprzecinkowych) oraz znaku +/- na początku. Spacje i zera na początku tekstu są ignorowane. Jeżeli nie uda mu się tego wykonać, zwracana jest specjalna wartość NaN - skrót od *Not a Number*. Podobnie dzieje się przy próbie zmiany typu *undefined*.

03

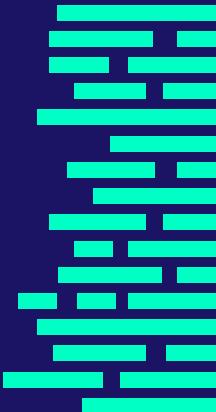
Zmienne

Jak deklarowało się zmienne kiedyś?

Używało się słowa kluczowego `var`. (`var txt = "Ala ma kota";`)

Jak deklaruje się zmienne teraz?

Używając `let` i `const`.



let i const

let - zmienna

```
let txt = "Przykładowy tekst"; //zmienna  
txt = "Inny tekst"; //zmieniamy wartość
```

const - stała

```
const nr = 102; //stała  
nr = 103; //błąd - nie można przypisać nowej wartości
```

JS JEST TYPOWANY DYNAMICZNY

Nie ma konieczności wyraźnego określania typów danych zmiennej przy jej deklaracji!!!

Typowanie dynamiczne

```
// Deklaracja zmiennej bez określenia jej typu  
let dynamicValue;  
  
// Przypisanie liczby do zmiennej (typ number)  
dynamicValue = 10;  
  
// Wyświetlanie typu zmiennej  
console.log(typeof dynamicValue); // "number"  
  
// Przypisanie ciągu znaków do zmiennej (typ string)  
dynamicValue = "Hello, world!";  
  
// Wyświetlanie typu zmiennej  
console.log(typeof dynamicValue); // "string"
```

```
// Przypisanie wartości logicznej do zmiennej  
(typ boolean)  
dynamicValue = true;
```

```
// Wyświetlanie typu zmiennej  
console.log(typeof dynamicValue); //  
"boolean"
```

```
// Przypisanie wartości null do zmiennej  
dynamicValue = null;
```

```
// Wyświetlanie typu zmiennej  
console.log(typeof dynamicValue); // "object"
```

04

Operatory

Operatory przypisania

Czyli operatory, które służą do przypisania do zmiennej jakiejś wartości, pola, obiektu itp.

```
{  
    let x = 5;  
    x += 3; //równoznaczne z x = x + 3;  
    console.log(x);  
}  
{  
    let x = 5;  
    x -= 3; //równoznaczne z x = x - 3;  
    console.log(x);  
}
```

Operatory porównania

Możemy je znaleźć między innymi w instrukcjach warunkowych. Służą one do porównywania lewej strony równania do prawej, w wyniku której zawsze zwracana jest prawda albo fałsz (true/false).

```
{  
    //== - porównuje obie wartości bez porównania ich typów  
    let a = 10;  
    console.log(a == 10) //true  
    console.log(a == "10") //true  
}  
  
{  
    //!= - czy wartości są różne, bez sprawdzenia typu  
    let a = 10;  
    console.log(a != 20) //true  
    console.log(a != 10) //false  
    console.log(a != "10") //false  
}
```

Operatory porównania

```
{  
    // === - porównuje obie wartości i ich typ  
    let a = 10;  
    console.log(a === 10) //true  
    console.log(a === "10") //false  
}  
  
{  
    // !== - czy wartości lub typy są różne  
    let a = 10;  
    console.log(a !== 10) //false  
    console.log(a !== "10") //true  
}
```

```
{  
    // < i > - mniejsze i większe  
    let a = 10;  
    let b = 20;  
    console.log(a < 20) //true  
    console.log(a < b) //true  
    console.log(a > b) //false  
}  
  
{  
    // <= i >= - mniejsze-równe i większe-równe  
    let a = 10;  
    let b = 20;  
    let c = 10;  
    console.log(a <= b) //true  
    console.log(a <= c) //true  
}
```

Operatory logiczne

Operatory logiczne używane są głównie w instrukcjach warunkowych. Służą do sprawdzania czy dane warunki są spełnione zwracając w wyniku true lub false.

Operator	Opis	Przykład	Wynik
<code>&&</code>	and (i)	<code>(x < 10 && y > 1)</code>	Prawda, bo x jest mniejsze od 10 i y jest większe od 1
<code> </code>	or (lub)	<code>(x > 8 y > 1)</code>	Prawda, bo x nie jest większe od 8, ale y jest większe od 1
<code>^</code>	xor (jeden z, ale nie dwa równocześnie)	<code>(x === 6 ^ y === 3)</code>	Fałsz, bo obydwa są prawdziwe
<code>!</code>	not (negacja)	<code>!(x === y)</code>	Prawda, bo negujemy to, że x === y

05

Instrukcje kontrolne

Instrukcja if, else i else if

```
const nr = Math.random() * 10;

if (nr > 5) {
    console.log("Liczba nr jest większa od 5");
}
```

Do każdej instrukcji możemy zastosować również else i else if

```
const nr = Math.random() * 10;

if (nr < 3) {
    console.log("Liczba jest mniejsza od 3");
} else if (nr <= 6) {
    console.log("Liczba jest mniejsza lub równa 6");
} else {
    console.log("Liczba jest większa od 6");
}
```

Switch

Instrukcja **switch** jest kolejnym sposobem tworzenia warunków - tym razem na zasadzie przymówiania wyniku do konkretnych wartości.

```
const animal = prompt("Wpisz jakiego masz zwierzaka");

switch (animal) {
    case "pies":
        console.log("Psy są najlepsze");
        break;
    case "kot":
        console.log("Koty są lepsze od psów");
        break;
    case "chomik":
        console.log("Każdy chomik jest super");
        break;
    default:
        console.log("Jakiś dziwny ten zwierzak");
}
```

```
<!DOCTYPE html>
<html>
<body>

<h1> BIBLIOGRAFIA </h1>

1. D.Flanagan, JavaScript
   przewodnik, Wydawnictwo Helion
2. https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages
   (dostęp 9.11.2023)

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
```

Dziękuję za
uwagę !

```
</body>
</html>
```

JavaScript

OBIEKTY
TABLICE
FUNKCJE
FUNKCJA STRZAŁKOWA



JavaScript

OBIEKTY
TABLICE
FUNKCJE
FUNKCJA STRZAŁKOWA



OBIEKTY

zmienne kumulujące wiele atrybutów w sobie, mogą to być inne zmienne a nawet metody, albo nawet inne obiekty.



PORPERTY	METHODS
name: Volkswagen model: Golf weight: 1383 color: blue	start drive stop brake

car.name = "Volkswagen"
car.model = "Golf"
car.weight = 1383
car.color = "blue"

car.start()
car.drive()
car.stop()
car.brake()

```
const car = {name: "Volkswagen", model: "Golf", weight: 1383, color: "blue"};
```

```
JS  
const ryszard = {  
    imie: "Ryszard",  
    nazwisko: "Wójcik",  
    wiek: 25,  
    kolorOczu: "brązowy",  
    imieNazwisko: function() {  
        return this.imie + " " + this.nazwisko;  
    }  
}  
  
console.log(ryszard);
```

```
// [object Object]  
{  
    "imie": "Ryszard",  
    "nazwisko": "Wójcik",  
    "wiek": 25,  
    "kolorOczu": "brązowy",  
    "imieNazwisko": function () {\n        return this.imie + \" \" + this.nazwisko;\n    }  
}
```

```
console.log(ryszard);  
  
// wyciągnięcie pojedynczej wartości  
const wiekRyszarda = ryszard.wiek;  
console.log(wiekRyszarda);  
  
const wiekRyszarda2 = ryszard['wiek'];  
console.log(wiekRyszarda2);
```

25
25

```
// obiekt jako klasa - szablon  
function Person(imie, wiek){  
    this.imie = imie;  
    this.wiek = wiek;  
}  
  
const ryszard2 = new Person("Ryszard", 25);  
console.log(ryszard2);
```

```
// [object Object]  
{  
    "imie": "Ryszard",  
    "wiek": 25  
}
```

JavaScript

OBIEKTY
TABLICE
FUNKCJE
FUNKCJA STRZAŁKOWA



TABLICE

zmienna do przechowywania
wiele wartości - uporządkowaną
listę

selectedColors[5]

selectedColors[0] → red
selectedColors[1] → blue
selectedColors[2] → pink
selectedColors[3] → white
selectedColors[4] → yellow

```
const tablica1 = new Array("red", "blue", "pink", "white", "yellow");
const tablica2 = new Array(5);
const tablica3 = ["red", "blue", "pink", "white", "yellow"];
```

0	1	2	3	4
red	blue	pink	white	yellow

value1 value2 value3 value4 value5

J S TABLICE - METODY

[].find()	→	[]
[].findIndex()	→	2
[].push()	→	[]
[].unshift()	→	[]
[].pop()	→	[]
[].shift()	→	[]
[].filter()	→	[]
[].map(() =>)	→	[]
[].forEach(() =>)	→	[]
[].fill(, 1)	→	" - - - - "
[].join("-")	→	[]
[].concat([])	→	[]
[[]].flat()	→	[]
[].slice(1, 3)	→	[]

```
const tablica = new Array("Poniedzialek", "Wtorek", "Sroda", "Czwartek");

console.log("Pierwszy element: ", tablica[0]);
console.log("Cala tablica: ", tablica);

tablica[4] = "Piatek";
console.log("Cala tablica: ", tablica);

const dlugoscTablicy = tablica.length;
console.log(dlugoscTablicy);

tablica[6] = "Niedziela";
tablica[5] = "Sobota";
console.log("Cala tablica: ", tablica);
```

"Pierwszy element: " "Poniedzialek"

"Cala tablica: " // [object Array] (4)
["Poniedzialek", "Wtorek", "Sroda", "Czwartek"]

"Cala tablica: " // [object Array] (5)
["Poniedzialek", "Wtorek", "Sroda", "Czwartek", "Piatek"]

5

"Cala tablica: " // [object Array] (7)
["Poniedzialek", "Wtorek", "Sroda", "Czwartek", "Piatek", "Sobota", "Niedziela"]

```
// wyświetlenie tablicy - 2 sposoby
for (let i = 0; i < tablica.length; i++) {
  console.log(tablica[i]);
}

tablica.forEach(x => console.log(x));
```

"Poniedzialek"
"Wtorek"
"Sroda"
"Czwartek"
"Piatek"
"Sobota"
"Niedziela"

```
// [object Array] (3)
[100, false, "JS"]
```

```
// inne sposoby na deklaracje tablicy
const tablica2 = new Array(3);
console.log(tablica2);
```

```
// [object Array] (3)
[,,]
```

```
tablica2[0] = 100;
tablica2[1] = false;
tablica2[2] = "JS";
console.log(tablica2);
```

```
const tablica3 = ["JS", "CSS", "HTML"];
console.log(tablica3);
```

```
// [object Array] (3)
["JS", "CSS", "HTML"]
```

JavaScript

OBIEKTY
TABLICE
FUNKCJE

FUNKCJA STRZAŁKOWA



FUNKCJE

blok kodu przeznaczony do wykonania określonego zadania, wykonywany gdy “coś” go wywołuje

CIAŁO FUNKCJI

```
NAZWA FUNKCJI           PARAMETRY
{                         }
function funcName(param1, param2...) {
    statement 1;
    statement 2;
    ...
}
return output; }           WYNIK FUNKCJI
```

WYWOŁANIE FUNKCJI

const funcInvoke = funcName(param1, param2...);
funcName(param1, param2...); → JEŻELI FUNKCJA NIC NIE
ZWRACA (typ zwracany void - nic)

JavaScript

OBIEKTY
TABLICE
FUNKCJE

FUNKCJA STRZAŁKOWA



FUNKCJA STRZAŁKOWA

wprowadzone w ES6, aby umożliwić krótszą składnię funkcji

PRZED "STRZAŁKĄ"	UŻYCIE "STRZAŁKI"
<pre>function dodawanie(a, b) { return a + b; }</pre>	<pre>const dodawanie = (a, b) => { return a + b; }</pre>
	<pre>const dodawanie = (a, b) => a + b;</pre>
	<pre>const mnozenie = a => a * a;</pre>

**NORMALNA FUNKCJA, a
FUNKCJA STRZAŁKOWA**

This

```
const zmienna = "Zmienna Globalna";
const obiekt = {
    zmienna: "Zmienna poziomu Obiektu",
    funkcjaStrzalkowa: () => {
        console.log(this.zmienna);
    },
    funkcjaNormalna() {
        console.log(this.zmienna);
    }
};
obiekt.funkcjaStrzalkowa();
obiekt.funkcjaNormalna();
```

The diagram illustrates the execution context stack for the arrow function `funkcjaStrzalkowa`. It shows two frames. The bottom frame is labeled "undefined" and contains the code `this.zmienna`. An arrow points from this frame up to the top frame, which is labeled "Zmienna poziomu Obiektu" and also contains the code `this.zmienna`. This visualizes how the `this` keyword refers to the object's internal state rather than the global variable.

undefined

"Zmienna poziomu Obiektu"

Arguments

```
function funkcja() {  
    console.log(arguments[0]);  
    console.log(arguments[1]);  
    console.log(arguments[2]);  
};  
funkcja(1,2,3);
```

1
2
3

```
const funkcja2 = {  
    printArguments: () => {  
        console.log(arguments);  
    }  
};  
funkcja2.printArguments(1,2,3);
```

Uncaught ReferenceError: arguments is not defined
at https://cdpn.io/cpe/boomboom/pen.js?key=pen.js-f37fd273-39ce-094d-6fb6-2f9e3a06a211:99

```
const funkcja3 = {  
    printArguments: (...args) => {  
        console.log(...args);  
    }  
};  
funkcja3.printArguments(1,2,3);
```

1 2 3

Konstruktor / “new”

```
function Article(topic) {  
  this.topic = topic;  
}  
const article = new Article("JS");  
console.log(article);
```

```
// [object Object]  
{  
  "topic": "JS"  
}
```

```
const Article2 = (topic) => {  
  this.topic = topic;  
}  
const article2 = new Article2("JS");  
console.log(article2);
```

```
Uncaught TypeError: Article2 is not a constructor  
at https://cdpn.io/cpe/boomboom/pen.js?key=pen.js-165236df-c127-e971-7234-6fff6c82d4d0:129
```

Return

```
function przykladFunkcji(){
    return 10;
}
const zmienna1 = przykladFunkcji();
console.log(zmienna1);
```

10

```
function przykladFunkcji2(){
    var liczba = 10;
}
const zmienna2 = przykladFunkcji2();
console.log(zmienna2);
```

undefined

```
function przykladFunkcji3() {
    var liczba = 10;
    return;
}
const zmienna3 = przykladFunkcji3();
console.log(zmienna3);
```

undefined

```
const dodajJeden = (liczba) => liczba + 1;
console.log(dodajJeden(10));
```

11

Koniec

Dziękuję za uwagę

Kinga Kuś ETI rok 3

Bibliografia

1. Kurs JavaScript - Tablice, <https://youtu.be/ybRZcGigGuQ?si=v-5whN8KZbvZhyWZ> [dostęp: 12.10.2023]
2. Kurs JavaScript - Tablice, https://youtu.be/ybRZcGigGuQ?si=sHEi5o0_ZmwBW_bu [dostęp: 16.10.2023]
3. <https://www.w3schools.com/js/> [dostęp: 21.10.2023]
4. <https://blog.bitsrc.io/arrow-functions-vs-regular-functions-in-javascript-458ccd863bc1> [dostęp: 21.10.2023]

jQuery, obiektowy model dokumentu

Radosław Żyła

jQuery to popularna biblioteka JavaScript, która ułatwia manipulację elementami HTML, obsługę zdarzeń i wiele innych operacji.

Biblioteka jQuery zawiera funkcje takie jak:

- Manipulacja HTML/DOM CSS
- HTML metody zdarzeń
- Efekty i animacje
- AJAX

Sposoby dodawania biblioteki jQuery

```
<script src="jquery-3.7.1.min.js"></script>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

Od czego zacząć?

```
$(document).ready(function(){  
    // nasz kod jQuery  
});
```

zmianę tekstu, atrybutów, stylów i innych właściwości elementów HTML.



```
$( "element" ).text("Nowa treść")
```

dodawanie i obsługę zdarzeń, takich jak kliknięcia, najazdy myszy itp.



```
$(document).mousemove(function(event){  
    $("span").text("X: " + event.pageX + ", Y: " + event.pageY);  
});
```

wiele wbudowanych funkcji do tworzenia animacji, takich jak **fadeIn**, **fadeOut**, **slideUp**, **slideDown**.



```
$( "#clickme" ).on( "click", function() {  
    $( "#book" ).fadeIn( "slow", function() {  
    });  
});
```

jQuery Effects

- [jQuery Hide/Show](#)
- [jQuery Fade](#)
- [jQuery Slide](#)
- [jQuery Animate](#)
- [jQuery stop\(\)](#)
- [jQuery Callback](#)
- [jQuery Chaining](#)

jQuery HTML

- [jQuery Get](#)
- [jQuery Set](#)
- [jQuery Add](#)
- [jQuery Remove](#)
- [jQuery CSS Classes](#)
- [jQuery css\(\)](#)
- [jQuery Dimensions](#)

jQuery Traversing

- [jQuery Traversing](#)
- [jQuery Ancestors](#)
- [jQuery Descendants](#)
- [jQuery Siblings](#)
- [jQuery Filtering](#)

jQuery AJAX

- [jQuery AJAX Intro](#)
- [jQuery Load](#)
- [jQuery Get/Post](#)

jQuery Misc

- [jQuery noConflict\(\)](#)
- [jQuery Filters](#)

jQuery Examples

- [jQuery Examples](#)
- [jQuery Editor](#)
- [jQuery Quiz](#)
- [jQuery Exercises](#)
- [jQuery Certificate](#)

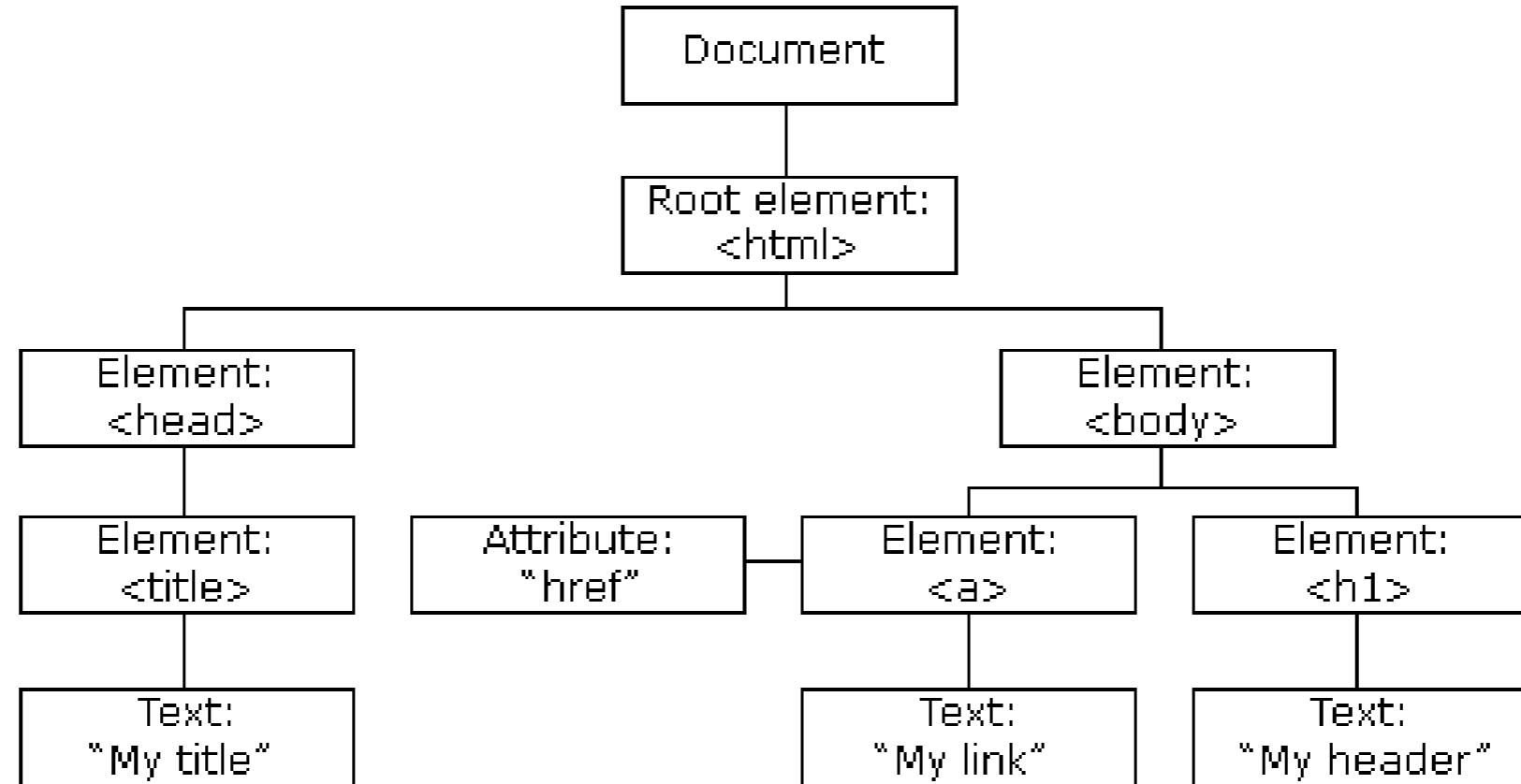
jQuery References

- [jQuery Overview](#)
- [jQuery Selectors](#)
- [jQuery Events](#)
- [jQuery Effects](#)
- [jQuery HTML/CSS](#)
- [jQuery Traversing](#)
- [jQuery AJAX](#)
- [jQuery Misc](#)
- [jQuery Properties](#)

HTML DOM

Za pomocą HMTL DOM JavaScript może uzyskiwać dostęp i zmieniać wszystkie elementy HTML. Odbywa się to za pomocą akcji zwanych metodami.

The HTML DOM Tree of Objects



document.GetElementBy

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

addEventListener

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

```
element.addEventListener("click", myFunction);

function myFunction() {
    alert ("Hello World!");
}
```

Walidacja formularzy

```
function validateForm() {  
  
    let x = document.forms["myForm"]["fname"].value;  
    if (x == "") {  
  
        alert("Name must be filled out");  
        return false;  
    }  
}
```

KONIEC

Prezentację wykonał:
Radosław Żyła

Domknięcia, zdarzenia w języku JavaScript

Arkadiusz Frydrych

Czym są zdarzenia?

W języku JavaScript, **zdarzenia** odnoszą się do interakcji użytkownika z przeglądarką lub innym środowiskiem, w którym działa skrypt. Zdarzenia reprezentują różne akcje, takie jak kliknięcia myszką, wprowadzanie tekstu, zmiany rozmiaru okna przeglądarki, czy wiele innych interakcji użytkownika.

```
const button = document.querySelector(".btn")

button.addEventListener("click", event => {
  console.log("Hello!");
})

button.addEventListener("click", event => {
  console.log("How are you?");
})

// This wil log
// "Hello!"
// "How are you?"
// to the console
```

Przykładowe zdarzenia

Kliknięcie myszką (**click**): To zdarzenie występuje, gdy użytkownik kliknie na element HTML, na przykład przycisk, link lub inny element interaktywny.

```
document.getElementById("myButton").addEventListener("click", function() {
    // Obsługa kliknięcia przycisku
});
```

Zmiana wartości pola formularza (**change**): To zdarzenie występuje, gdy użytkownik zmienia wartość pola formularza, takiego jak pole tekstowe lub checkbox.

```
document.getElementById("myInput").addEventListener("change", function() {
    // Obsługa zmiany wartości pola formularza
});
```

Naciśnięcie klawisza klawiatury (**keydown**): To zdarzenie występuje, gdy użytkownik naciska klawisz na klawiaturze.

```
document.addEventListener("keydown", function(event) {
    // Obsługa naciśnięcia klawisza klawiatury
});
```

Sposoby rejestracji zdarzeń

Metoda `addEventListener` pozwala rejestrować zdarzenia na elementach DOM i jest najczęściej stosowaną techniką. Przyjmuje trzy argumenty: typ zdarzenia, funkcję obsługi zdarzenia (callback) i opcjonalnie flagę mówiącą o tym, czy zdarzenie ma być przechwytywane w fazie przechwytywania czy w fazie wywoływania.

Można również używać atrybutów HTML takich jak `onclick` i `onchange` itp., aby bezpośrednio przypisać funkcje obsługi zdarzeń do elementów HTML. Jest to starszy sposób i mniej zalecany z powodu swojego ograniczonego zastosowania i braku oddzielenia logiki od warstwy prezentacji.

```
const button = document.getElementById("myButton");
button.addEventListener("click", function() {
    // Obsługa kliknięcia przycisku
});
```

```
<button id="myButton" onclick="myFunction()">Kliknij mnie</button>
```

Inne zdarzenie warte uwagi

Obsługa zdarzeń w JavaScript jest kluczowym elementem tworzenia dynamicznych i interaktywnych aplikacji internetowych. Programiści mogą reagować na zdarzenia, wykonywać określone akcje i zmieniać zawartość strony w odpowiedzi na działania użytkownika.

Rodzaj zdarzenia	Opis
click	element kliknięty
change	opuszczamy element, który zmienił zawartość
mouseover	kursor na elemencie
mouseout	kursor opuścił element
mouseenter	kursor wjechał na element
mouseleave	kursor zjechał z elementu
dblclick	podwójne kliknięcie elementu
submit	wysłanie formularza
resize	zmiana wielkości okna
focus	element został wybrany
blur	element przestał być aktywny
keydown	naciśnięto klawisz na klawiaturze
keyup	puszczono klawisz
input	wciśnięty jest klawisz
load	obiekt został załadowany
contextmenu	wciśnięto prawy przycisk myszy i pojawiło się menu kontekstowe
wheel	kręcone jest koło myszy
select	zaznaczono zawartość obiektu
unload	strona jest opuszczana
animationstart	rozpoczyna się animacja

Domknięcia

Domknięcia (closures) to pojęcie w języku JavaScript, które odnosi się do zdolności funkcji do przechowywania dostępu do zmiennych z otaczającego zakresu (scope) nawet po zakończeniu działania tej funkcji. Dzięki domknięciom można tworzyć funkcje, które zachowują stan i dostęp do zmiennych poza swoim własnym zakresem.

```
1 | var a = "outside of a closure";
2 | function myClosure() {
3 |   var a = "inside a closure";
4 | }
5 | myClosure();
```

Przykłady zastosowania domknięcia

W tym przykładzie funkcja `createCounter` tworzy i zwraca funkcję wewnętrzną, która działa jako licznik. Ta wewnętrzna funkcja korzysta z domknięcia, aby przechowywać i aktualizować stan licznika, co pozwala na tworzenie niezależnych instancji liczników.

```
function createCounter() {
  let count = 0;

  return function () {
    count++;
    console.log(count);
  };
}

const counter1 = createCounter();
counter1(); // Wyświetli 1
counter1(); // Wyświetli 2

const counter2 = createCounter();
counter2(); // Wyświetli 1 (inny stan od counter1)
```

Funkcje typu callback

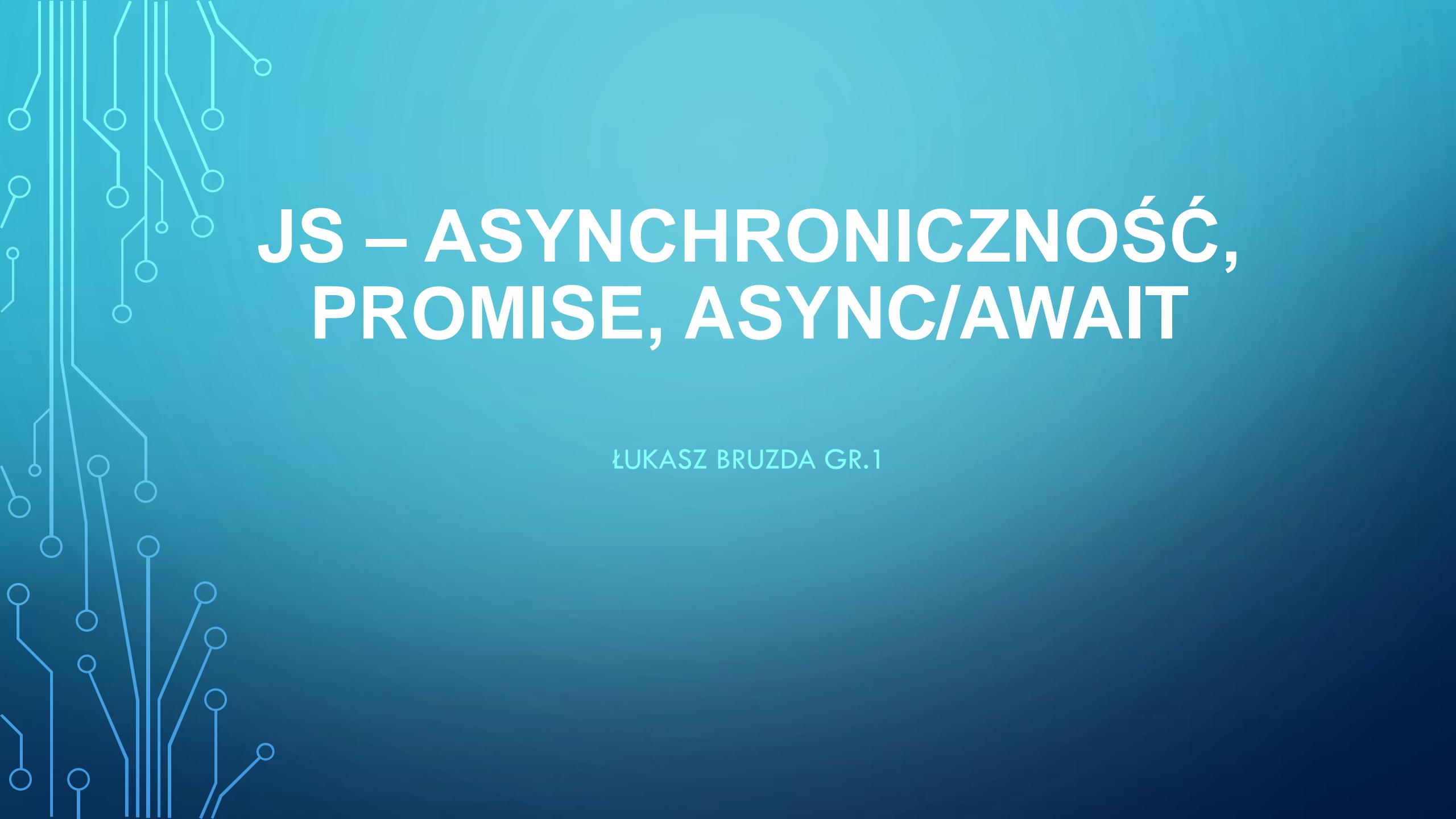
Domknięcia są używane w JavaScript w wielu kontekstach, takich jak obsługa zdarzeń, czy funkcje zwrotne (callbacks). Pozwalają na zachowanie stanu i dostępu do zmiennych poza zakresem funkcji, co jest bardzo przydatne w tworzeniu bardziej elastycznego i modularnego kodu.

```
function operateOnNumbers(a, b, operationCallback) {
  const result = operationCallback(a, b);
  console.log(`Wynik operacji: ${result}`);
}

function add(a, b) {
  return a + b;
}

function subtract(a, b) {
  return a - b;
}

operateOnNumbers(5, 3, add); // Wynik operacji: 8
operateOnNumbers(5, 3, subtract); // Wynik operacji: 2
```



JS – ASYNCHRONICZNOŚĆ, PROMISE, ASYNC/AWAIT

ŁUKASZ BRUZDA GR.1

ASYNCHRONICZNOŚĆ

- Asynchroniczność pozwala nam wykonywać kilka zadań równocześnie. Dzięki temu możemy odpalić kilka zajmujących czas funkcjonalności na raz, a następnie zareagować na ich zakończenie.

Przykład:

```
setTimeout(MojeFunkcja, 3000);

function MojeFunkcja() {
  document.getElementById("demo").innerHTML = "Czas upłynął!!";
}
```

Przykład:

```
setTimeout(function() { MojaFunkcja("Czas upłynął !!"); }, 3000);

function MojaFunkcja(value) {
  document.getElementById("demo").innerHTML = value;
}
```

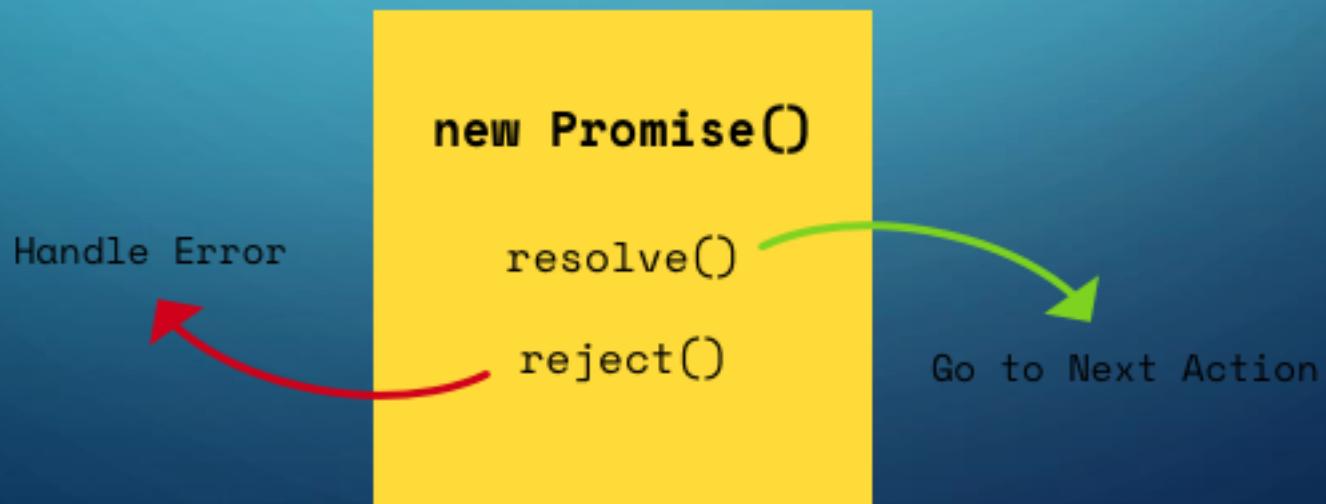
Przykład:

```
setInterval(MojaFunkcja, 1000);

function MojaFunkcja() {
  let d = new Date();
  document.getElementById("demo").innerHTML=
    d.getHours() + ":" +
    d.getMinutes() + ":" +
    d.getSeconds();
}
```

ASYNCHRONICZNOŚĆ/ PROMISE

Dzięki Asynchronicznośc JavaScript może rozpoczynać długotrwałe zadania i równolegle kontynuować wykonywanie innych zadań. Jednak Asynchroniczność jest trudna napisania i trudna do debugowania. Dlatego w JavaScript programowanie asynchronous rozwiązuje się za pomocą Promises.



PROMISE

Praca z **promise** składa się z 2 kroków.

I za pomocą konstruktora Promise tworzymy obiekt.

II za pomocą odpowiedniej funkcji które nam udostępnia reagujemy na jej zakończenie.

1. create

Tworzymy obietnicę

```
const load = new Promise((resolve, reject) {  
    if (dataIsOk) {  
        resolve(data);  
    } else {  
        reject("error");  
    }  
});
```

gdy sukces

gdy porażka

2. consume

Reagujemy na jej wykonanie

```
load  
    .then(result => {  
        console.log(result);  
    })  
    .catch(err => {  
        console.log(err);  
    });
```

UŻYWANIE PROMISE

Do stworzenia Promise korzystamy z konstruktora `Promise()` który w parametrze przyjmuje funkcję, do której przekazujemy referencję do dwóch funkcji, które będą wywoływane w przypadku zwrócenia poprawnego lub błędного kodu.

```
myPromise.then(  
  function(value) { /* code if successful */ },  
  function(error) { /* code if some error */ }  
)
```

Każda obietnica może zakończyć się na dwa sposoby - powodzeniem (`resolve`) i niepowodzeniem (`reject`).

Przykład:

```
let myPromise = new Promise(function(myResolve, myReject) {  
  let req = new XMLHttpRequest();  
  req.open('GET', "mycar.htm");  
  req.onload = function() {  
    if (req.status == 200) {  
      myResolve(req.response);  
    } else {  
      myReject("Plik nie znaleziony");  
    }  
  };  
  req.send();  
});  
  
myPromise.then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}  
)
```

ASYNC/AWAIT

Async powoduje, że funkcja zwraca Promise :

```
async function myFunction() {  
    return "Hello";  
}  
myFunction().then(  
    function(value) {myDisplayer(value);},  
    function(error) {myDisplayer(error);}  
);
```

Await powoduje, że funkcja czeka na Promise:

```
async function Wyswietl() {  
    let myPromise = new Promise(function(resolve, reject) {  
        resolve("Text 234 !");  
    });  
    document.getElementById("demo").innerHTML = await myPromise;  
}  
  
Wyswietl();
```

AWAIT

- Słowa **await** możemy używać tylko wewnątrz funkcji poprzedzonej słowem

async:

```
async function renderPage() {  
  const city = await render();  
}
```

- Instrukcja **await** oznacza to, że kolejna operacja rozpoczęcie się dopiero, gdy poprzednia się zakończy.

Przykład:

```
(async () => {  
  const bar1 = new Bar({place: test});  
  const bar2 = new Bar({place: test});  
  const bar3 = new Bar({place: test});  
  const bar4 = new Bar({place: test});  
  const bar5 = new Bar({place: test});  
  
  await bar1.animate()  
  await bar2.animate()  
  await bar3.animate()  
  await bar4.animate()  
  await bar5.animate()  
  alert("Zakończono");  
})()
```

BIBLIOGRAFIA

- <https://fsgeek.pl/post/asynchronicznosc-w-javascript/>
- <https://kursjs.pl/kurs/ajax/promise>
- <https://www.w3schools.com/js/js.promise.asp>
- https://www.w3schools.com/js/js_asynchronous.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- https://www.w3schools.com/js/js_async.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- <https://kursjs.pl/kurs/ajax/async-await>

JS – CALLBACK

ŁUKASZ BRUZDA GR.1

CALLBACK

Callback jest to funkcja przekazywana jako parametr innej funkcji w celu późniejszego jej wywołania w tejże funkcji. Funkcja Callback może zostać uruchomiona po zakończeniu innej funkcji

Funkcja z Callback

```
function PokazanieWyn(something) {  
    document.getElementById("demo").innerHTML = something;  
}  
  
function ObliczenieWyn(num1, num2, CALLBACK) {  
    let sum = num1 + num2;  
    CALLBACK(sum);  
}  
  
ObliczenieWyn(5, 5, PokazanieWyn);
```

Funkcja bez Callback

```
function PokazanieWyn(something) {  
    document.getElementById("demo").innerHTML = something;  
}  
  
function ObliczenieWyn(num1, num2) {  
    let sum = num1 + num2;  
    return sum;  
}  
  
let result = ObliczenieWyn(5, 5);  
PokazanieWyn(result);
```

PO CO UŻYWAĆ CALLBACK

- Callback sprawdza się w funkcjach asynchronicznych, gdzie jedna funkcja musi czekać na inną , pozwalając określić, co powinno się stać po zakończeniu tych operacji asynchronicznych, zapewniając wykonanie kodu we właściwej kolejności.
- Umożliwiają tworzenie funkcji wielokrotnego użytku, które można wykonywać przy różnych funkcjach.

PRZYKŁAD

Funkcja z zaznaczonym Callback

```
// tworzę przycisk
const button = document.createElement("button");
button.innerText = "Click me";
document.body.appendChild(button)

// nasłuchuję na kliknięcie
button.addEventListener('click', changeColor)

// definiuję, co ma się stać, gdy kliknę przycisk
function changeColor(e) {
  e.target.style.backgroundColor = "red"
}
```

Może to też być tak zapisane:

```
// funkcja nazwana
button.addEventListener('click', function changeColor(e) {
  e.target.style.backgroundColor = "red"
})
```

Może to też być tak zapisane:

```
// funkcja anonimowa
button.addEventListener('click', function(e) {
  e.target.style.backgroundColor = "red"
})
```

1. Callback to funkcja przekazywana do drugiej funkcji przez parametr.
2. Callback to funkcja uruchamiana przez funkcję, do której ją przekazaliśmy.



DZIĘKUJĘ ZA UWAGĘ

BIBLIOGRAFIA

- https://www.w3schools.com/js/js_callback.asp
- https://developer.mozilla.org/en-US/docs/Glossary/Callback_function
- <https://www.nafrontendzie.pl/javascript-metody-obiektu-jako-callback>
- <https://devmentor.pl/b/callback-functions-w-javascript>

TypeScript

Czyli JavaScript ze składnią dla typów.



PLAN PREZENTACJI



01

Wprowadzenie

Ogólne informacje, cechy i historia powstania języka.

03

Zalety i wady języka TypeScript

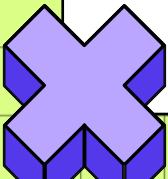
02

TypeScript vs JavaScript

Różnice między tymi językami.

04

Przykładowy program w TS i JS

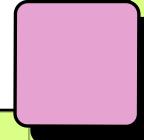
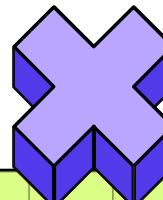




01

Wprowadzenie

Ogólne informacje, cechy i historia
powstania języka.





Ogólne informacje



- **Jest nadzbiorem języka JavaScript**
(dodaje np. interfejsy czy statyczne typowanie)
- Pierwsze pojawienie się: **1 października 2012**
- Rozszerzenia plików: **.ts, .tsx, .mts, .ct**
- Zaprojektowany i wydany przez firmę **Microsoft**



02

TypeScript vs JavaScript

Różnice między tymi językami.



Różnice

TS

JS

Typy danych

statyczne typowanie

dynamiczne typowanie

Kompilacja

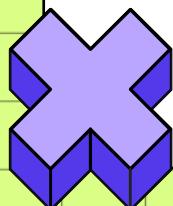
wymaga komplikacji lub transpilacji,
aby przekształcić kod źródłowy na
kod JavaScript

można go uruchamiać bezpośrednio

Narzędzia deweloperskie

oferuje bardziej zaawansowane
narzędzia deweloperskie

mniej rozbudowane narzędzia
deweloperskie

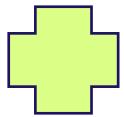


03

Zalety i wady języka TypeScript



Zalety i wady



Bezpieczeństwo typów

Bogaty system typów

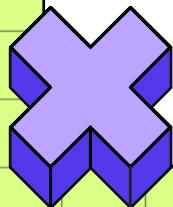
Zaawansowane narzędzia deweloperskie

Czas nauki

Dodatkowy etap kompilacji



Zwiększone zużycie zasobów



04

Zalety i wady języka TypeScript

Przykładowy program w TS i JS

```
// TypeScript
function dodaj(a: number, b: number): number {
    return a + b;
}

const wynik: number = dodaj(5, 3);
console.log("Wynik dodawania: " + wynik);

// JavaScript (bez deklaracji typów)
function dodajJS(a, b) {
    return a + b;
}

const wynikJS = dodajJS(5, 3);
console.log("Wynik dodawania: " + wynikJS);
```

Przykładowy program w TS i JS

TypeScript

```
let x: number = 5;  
console.log(x);
```

Statyczna deklaracja typu

JavaScript

```
let x = 5;  
console.log(x);
```

Dynamiczna deklaracja typu

Dziękuję za uwagę!

Więcej informacji: <https://www.typescriptlang.org/>





{

Porównanie JavaScriptu I TypeScriptu

JS



}



Czym jest TypeScript?

TypeScript to język typu open source stworzony przez firmę Microsoft, który szybko kompiluje JavaScript. TypeScript to zestaw technologii, który oprócz tego, że jest językiem skryptowym, ułatwia tworzenie stron internetowych.

Dzięki dodatkowym możliwościom dodanym specjalnie w celu obejścia ograniczeń bibliotek JavaScript, TypeScript jest JavaScriptem. Dla programistów korzystne jest posiadanie podstawowej wiedzy na temat definicji typów statycznych i zarządzanie złożonością kodu przed użyciem TypeScript. Kluczowe jest zrozumienie różnicy między TypeScript i JavaScript.





{ . . Do czego jest używany

Uproszczenie kodu JavaScript,
sprawiając że aplikacje będą
łatwiejsze do odczytania i
debugowania.

Szybkie kodowanie dla
wszystkich złożonych
i dużych Aplikacji

Pozwala korzystać
ze wszystkich zalet
ES6(ECMAScript 6)

Umożliwianie budowy
skalowalnych aplikacji
internetowych, które są
łatwiejsze do utrzymania
i rozwijania





Czym jest JavaScript?

JavaScript to dynamiczny język programowania komputerowego. Jego implementacje umożliwiają skryptowi po stronie klienta interakcję z użytkownikami i tworzenie dynamicznych stron i jest najczęściej używany jako komponent stron internetowych. Jest to obiektowy język programowania, który można interpretować.



LiveScript była pierwotną nazwą JavaScript, ale Netscape zmienił ją na JavaScript, być może w odpowiedzi na szum, jaki wywołała Java. LiveScript, poprzednik JavaScript, zadebiutował w Netscape 2.0 w 1995 roku. Netscape, Internet Explorer i inne przeglądarki internetowe zawierają rdzeń języka ogólnego przeznaczenia.





{ . . Do czego jest używany

Łatwe dodawanie
interaktywności do stron
internetowych

Tworzenie aplikacji
webowych i
mobilnych.

Budowa serwerów WWW
i tworzenie
aplikacji

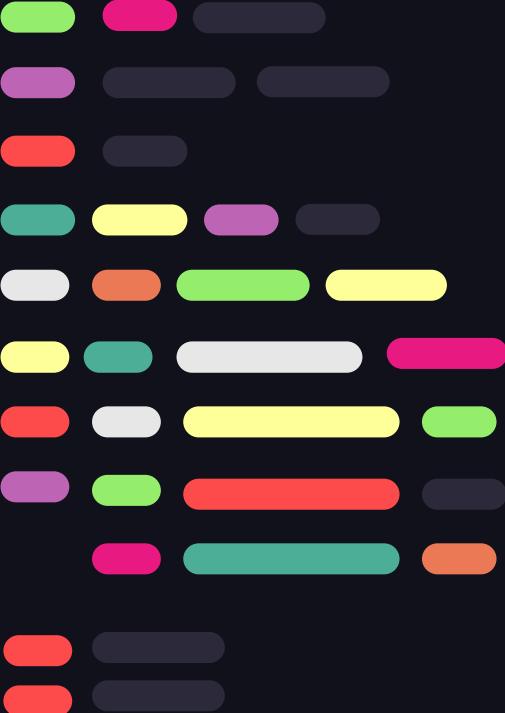
Przyspieszenie
działania
aplikacji

rozwój front-endu i
rozwój back-endu

Wykonywanie struktury
danych i walidacja w
samej przeglądarce
internetowej, a nie na
serwerze.



Czym TypeScript różni się od JavaScriptu?



- TypeScript to nadzbiór JavaScriptu, natomiast JavaScript to język skryptowy, który umożliwia tworzenie dynamicznych stron internetowych.
- • Kod JavaScript nie wymaga komplikacji, podczas gdy kod TypeScript tak.
- • Kiedy porównuje się TypeScript i JavaScript, TypeScript obsługuje prototypowanie, podczas gdy JavaScript nie.
- • W przeciwieństwie do JavaScriptu, Typescript wykorzystuje pojęcia takie jak typy i interfejsy do opisu używanych danych.
- • JavaScript jest najlepszym wyborem dla małych projektów, podczas gdy TypeScript zapewnia solidny system typów z rodzajami generycznymi i możliwościami JS dla większych projektów.

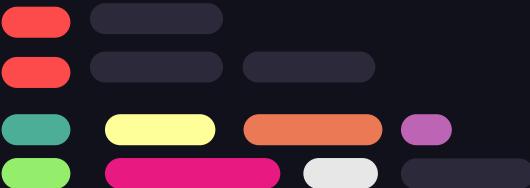




Plusy i minusy JavaScriptu

{ Plusy

- Kompatybilny ze wszystkimi nowoczesnymi przeglądarkami,
- Można go używać z innymi językami programowania takimi jak Node Js, Angular JS i inne,
- Wszechstronny i dynamiczny,
- Minimalizuje długość kodu



Minusy }

- Kod jest widoczny dla użytkowników, przez co może zostać wykorzystany do szkodliwych celów,
- Przed publikacją kod musi zostać uruchomiony na różnych platformach,
- Pojedynczy błąd w kodzie może zatrzymać renderowanie całego kodu JavaScript na stronie,





Plusy i minusy TypeScriptu

{ Plusy

- Łatwo wykrywalne błędy na etapie kompilacji,
- Lepsze wsparcie dla narzędzi,
- Integracja z istniejącym kodem JavaScript,
- Wsparcie dla nowych funkcji ECMAScript.

Minusy }

- Dodatkowy czas kompilacji,
- Zazwyczaj wymaga kompilacji kodu na każdym etapie
- Złożoność integracji z niektórymi narzędziami.



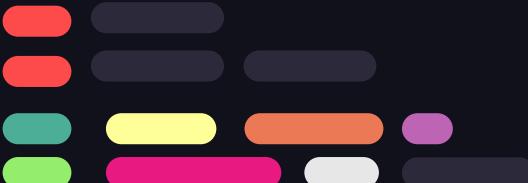
Porównanie kodu

JS

{



W czystym JavaScript nie ma deklaracji typów, co oznacza, że funkcja może przyjmować różne rodzaje danych jako argumenty, a wynik może być dowolnego typu. JavaScript nie sprawdza typów przed wykonaniem, więc nie zgłosi błędu, jeśli do funkcji zostaną przekazane argumenty innego typu.



```
typescript

// Przykładowa funkcja dodawania w TypeScript
function dodaj(a: number, b: number): number {
    return a + b;
}

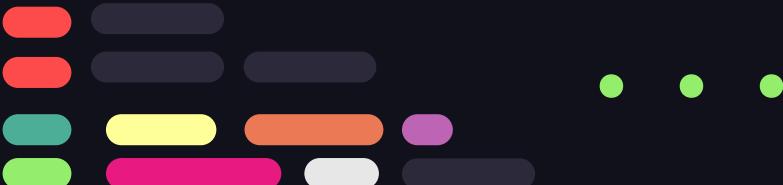
// Wywołanie funkcji dodawania
console.log(dodaj(3, 5)); // Zwróci: 8
```

}

{



W TypeScript deklaracje typów umożliwiają programistom określanie oczekiwanych typów dla argumentów funkcji oraz dla zwracanej wartości. Dzięki temu TypeScript może wykryć błędy typów w trakcie komplikacji, co pomaga zapobiegać błędom działania programu.



```
javascript

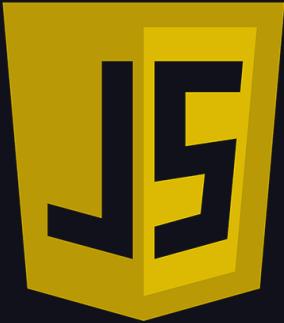
// Przykładowa funkcja dodawania w JavaScript
function dodaj(a, b) {
    return a + b;
}

// Wywołanie funkcji dodawania
console.log(dodaj(3, 5)); // Zwróci: 8
```

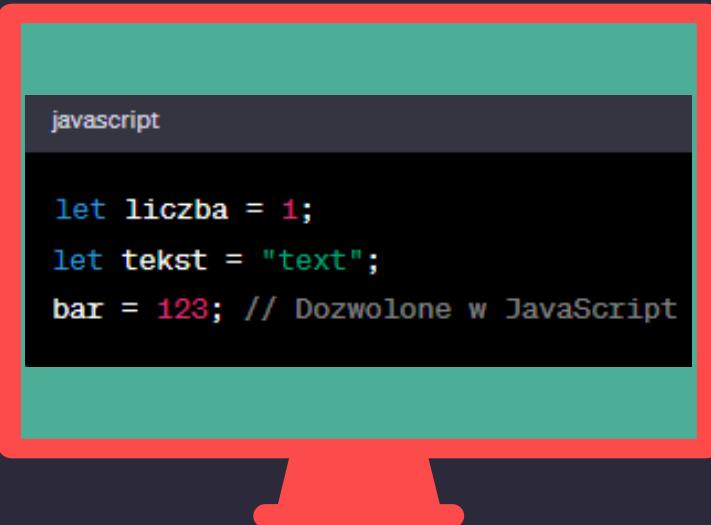
}

JS

{



Zmienne liczba i tekst są deklarowane bez określania ich typów, co oznacza, że ich typy mogą być zmienione dynamicznie podczas działania programu.



A monitor icon with a red border and a red base, containing a code snippet.

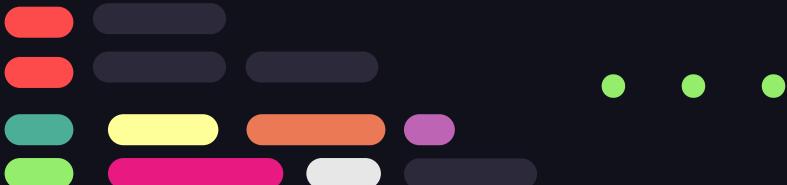
```
javascript
let liczba = 1;
let tekst = "text";
bar = 123; // Dozwolone w JavaScript
```

}

{



Zmienne liczba i tekst mają określone typy (odpowiednio number i string) podczas deklaracji, co uniemożliwia przypisanie wartości innego typu niż określony, zapewniając większe bezpieczeństwo typów i wykrywanie błędów już na etapie kompilacji.



typescript

```
let liczba: number = 1;  
let tekst: string = "text";  
bar = 123; // Nie dozwolone w TypeScript,
```

}



{

Koniec

Wykonał: Jakub Skorupa



}

Źródła:

{

-typescriptlang.org

-javatpoint.com/javascript-vs-typescript

}

