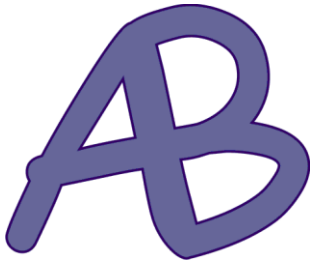
Aplikacje frontendowe



Wykład

Część 1 – Poznajemy JavaScript

Czym jest JavaScript

- ✓ JavaScript jest wysokopoziomowym, interpretowanym językiem programowania, dynamicznie typowanym, obiektowym.
- ✓ Stosowany głównie w kontekście programowania aplikacji webowych (stron WWW i aplikacji internetowych).
- ✓ JavaScript działa po stronie klienta (ang. client-side scripting), ale od czasu pojawienia się Node.js również po stronie serwera (ang. server-side scripting).

Czym jest JavaScript

JavaScript to nie Java

- ✓ Oprócz nazwy, JavaScript i Java nie mają ze sobą prawie nic wspólnego. Java to kompletny język programowania, opracowany i promowany przez firmę Sun Microsystems.
- ✓ Za pomocą Javy, potomka języków C i C + +, programiści mogą tworzyć kompletne aplikacje. W przeciwieństwie do tych ostatnich, Java kusi jednak perspektywą braku ograniczeń sprzętowych. Oznacza to, że programista piszący w Javie po stworzeniu programu będzie mógł uruchomić go na dowolnym sprzęcie, bez względu na to, czy działa on pod kontrolą systemu Windows, Mac OS czy Unix
- ✓ Obok tworzenia całych, rozbudowanych aplikacji, zasadniczym zastosowaniem Javy jest tworzenie apletów, niewielkich programów, które są pobierane z internetu i uruchamiane w przeglądarkach internetowych.
- ✓ Aplety Javy umieszcza się na stronach WWW, używając znacznika <object> języka HTML,

Podstawowe różnice pomiędzy językami Java a JavaScript

JavaScript	Java
Język interpretowany na komputerze klienta	Język kompilowany do tzw. b-kodu, wykonywanego następnie za pomocą wirtualnej maszyny Javy na komputerze klienta
Język oparty na predefiniowanych obiektach, niepozwalający jednak na stosowanie mechanizmów programowania obiektowego jak np. dziedziczenie	Język zorientowany obiektowo z obsługą wszystkich mechanizmów obiektowości
Kod programu jest zagnieżdżony w kodzie HTML	Kod programu jest niezależny od kodu HTML i znajduje się w oddzielnych plikach
Zmienne i ich typ nie muszą być deklarowane przed użyciem	Zmienne i ich typ muszą być zadeklarowane przed ich użyciem w programie
Odwołania do obiektów i funkcji są wykonywane podczas uruchamiania programu	Wszystkie odwołania do obiektów i funkcji są sprawdzane na etapie kompilacji
Ze względów bezpieczeństwa nie ma możliwości zapisu na dysk twardy	Ze względów bezpieczeństwa aplety, (w przeciwieństwie do aplikacji) nie mają możliwości zapisu na dysk twardy

Co potrafi JavaScript?

1. Manipulacja dokumentem HTML/CSS (DOM API)
   a) JavaScript umożliwia dynamiczną manipulację strukturą, treścią oraz stylami dokumentu HTML.
   b) Możemy dodawać, usuwać lub modyfikować elementy DOM w odpowiedzi na zdarzenia użytkownika (event-driven programming).
2. Obsługa zdarzeń użytkownika - reakcja na kliknięcia, ruchy myszy, zmiany wartości pól formularzy, naciśnięcia klawiszy itp.
3. Asynchroniczna komunikacja z serwerem (AJAX, Fetch API) - komunikacja z serwerem bez konieczności przeładowywania strony,
4. Tworzenie aplikacji jednostronicowych (Single Page Applications) dzięki dynamicznej aktualizacji DOM bez przeładowywania strony.
5. Tworzenie animacji obiektów HTML/CSS. (można sterować animacjami ręcznie lub używać gotowych bibliotek jak GSAP czy Anime.js).

Co potrafi JavaScript?

6. Dostęp do API przeglądarkiJavaScript może komunikować się z różnymi Web API, takimi jak:
    a. LocalStorage i SessionStorage (przechowywanie danych w przeglądarce).
    b. Geolocation API (pozyskiwanie lokalizacji użytkownika).
    c. Canvas API (grafika 2D, np. gry lub wizualizacje).
    d. WebRTC (komunikacja wideo i audio peer-to-peer).
    e. Notifications API, Clipboard API i wiele innych
7. Programowanie serwerowe - dzięki Node.js JavaScript funkcjonuje także po stronie serwera.
8. Programowanie mobilne i desktopowe z użyciem frameworków takich jak:
    a. React Native – tworzenie aplikacji mobilnych na Android i iOS.
    b. Electron – budowanie aplikacji desktopowych (np. Slack, Visual Studio Code)

Czego JavaScript nie zrobi

JavaScript jest językiem strony klienta, co oznacza, że został zaprojektowany do pracy na naszym komputerze, a nie na serwerze. Z tego powodu w język JavaScript zostało wbudowanych kilka ograniczeń, wynikających przede wszystkim ze względów bezpieczeństwa:

a) JavaScript nie pozwala na odczytywanie i zapisywanie plików na komputerze klienckim.

b) JavaScript nie pozwala na zapisywanie plików na serwerach. Oznacza to, że musimy samodzielnie tworzyć programy po stronie serwera obsługujące mechanizmy zapisywania takich informacji. Mogą to być na przykład moduły napisane w PHP albo programy w języku Java lub Python.

c) Skrypt w języku JavaScript nie może zamknąć okna, którego sam nie otworzył. Ma to zapobiegać sytuacji, w której strona WWW przejmuje kontrolę nad przeglądarką i zamyka okna wszystkich innych stron.

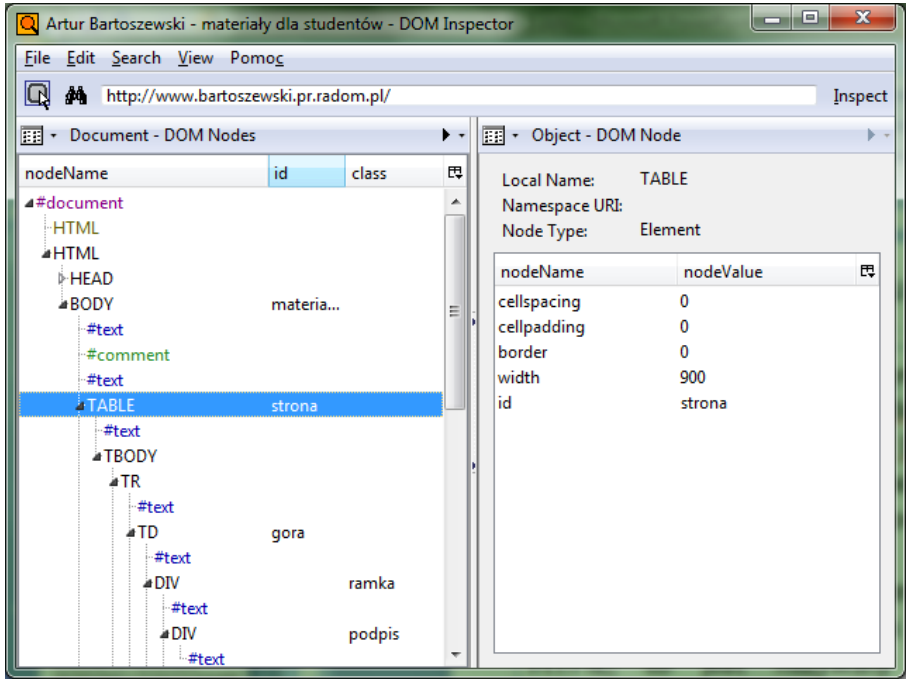
d) Język JavaScript nie może odczytywać informacji z pozostałych stron WWW otwartych w przeglądarce, pochodzących z innych serwerów.

Wprowadzenie modelu DOM

- ✓ Na stronie WWW obiekty składające się na stronę (lub dokument) reprezentowane są za pomocą struktury drzewiastej.
- ✓ Język JavaScript uznaje każdy z elementów takiego drzewa za osobny obiekt, dzięki czemu możemy stosować skrypty JavaScript do manipulowania dowolnym z tych obiektów. Taka reprezentacja obiektów tworzących dokument nazywana jest modelem DOM (Document Objęci Model — model obiektów dokumentu).
- ✓ Każdy z obiektów znajdujących się w drzewie nazywany jest węzłem (ang. node).
- ✓ Za pomocą języka JavaScript można dowolnie modyfikować drzewa poprzez dodawanie, edytowanie i usuwanie poszczególnych węzłów.
- ✓ Każdy z obiektów w drzewie jest węzłem. Jeżeli węzeł zawiera znacznik HTML, to jest nazywany węzłem elementu (ang. element node). Jeżeli jednak zawiera tylko tekst, to nazywany jest węzłem tekstowym (ang. text node). Oczywiście, węzły elementów mogą zawierać w sobie węzły tekstowe.

Wprowadzenie modelu DOM

- ✓ Każdy z obiektów znajdujących się w drzewie nazywany jest węzłem (ang. node).
- ✓ Za pomocą języka JavaScript można dowolnie modyfikować drzewa poprzez dodawanie, edytowanie i usuwanie poszczególnych węzłów.



Zdarzenia

- ✓ Zdarzenia to czynności, które wykonuje użytkownik podczas odwiedzania strony.
- ✓ JavaScript obsługuje zdarzenia za pomocą poleceń zwanych funkcjami obsługi zdarzeń. Czynność wykonywana przez użytkownika na stronie wywołuje, umieszczoną w skrypcie, funkcję obsługi zdarzenia.

Zdarzenie	Co obsługuje
onabort	Zaniechano ładowania strony.
onblur	Obiekt przestał być aktywny (wybrany).
onchange	Stan obiektu uległ zmianie.
onclick	Kliknięto obiekt.
onerror	Błąd w skrypcie.
onfocus	Obiekt został uaktywniony (wybrany).
onload	Zakończyło się ładowanie obiektu.
onmouseover	Obiekt został wskazany myszką.
onmouseout	Obiekt przestał być wskazywany myszką.
onselect	Zawartość obiektu została zaznaczona.
onsubmit	Zawartość formularza została przesłana do serwera.
onunload	Zmieniono wyświetlaną stronę.

Część 2 – Skrypty

Umieszczanie skryptów w kodzie HTML

- ✓ Skrypty osadzone - umieszczane wewnątrz kodu HMTL przy użyciu znacznika <script>]

```
<script>
        // Twój kod JavaScript
</script>
```

- ✓ Skrypty zewnętrzne – skrypty mogą być przechowywane w osobnych plikach z rozszerzeniem .js. Następnie można je dołączyć do strony za pomocą atrybutu src w tagu <script>.

```
<script src="ścieżka/do/pliku.js"></script>
```

Znacznik <SCRIPT>

Kod JavaScript musi być umieszczony pomiędzy znacznikami HTML <script> i </script>.

Na stronie HTML skrypty można umieszczać w dwóch miejscach:

- pomiędzy znacznikami <head> i </head> — skrypt taki nazywamy skryptem nagłówka
- pomiędzy znacznikami <body> i </body> w treści dokumentu.

Bloki <SCRIPT>

- ✓ Umieszczanie skryptu w nagłówku (przy pobieraniu strony będzie ładowany jako pierwszy) lub ciele dokumentu (niewskazane!)
- ✓ Zaleca się umieszczanie skryptów JavaScript na końcu strony, tuż przed zamknięciem tagu </body>.
- ✓ Zapewnia to, że pozostała zawartość strony zostanie wczytana przed wykonaniem skryptów, co może przyspieszyć ładowanie strony.

Bloki <SCRIPT>

- ✓ Dowolna liczba bloków <script> na stronie
- ✓ Kolejność osadzania skryptów ma znaczenie – będą wykonywane w kolejności umieszczenia na stronie
- ✓ Przeglądarka może wstrzymywać ładowanie i wyświetlanie strony póki nie wykona skryptu

Instrukcja document.write

- ✓ Instrukcja document.write() pozwala na wyprowadzenie tekstu na ekran przeglądarki. Tekst, który chcemy wyświetlić, należy ująć w nawiasy i cudzysłowy i podać zaraz za document.write() np.

```
document.write ("Jaki miły mamy dzień!")
```

WSKAZÓWKA:
Warto zwrócić uwagę, że w celu poprawnej interpretacji polskich liter przez przeglądarkę dodaliśmy w sekcji HEAD znacznik:

```
<meta charset="UTF-8" />
```

Pierwszy Skrypt

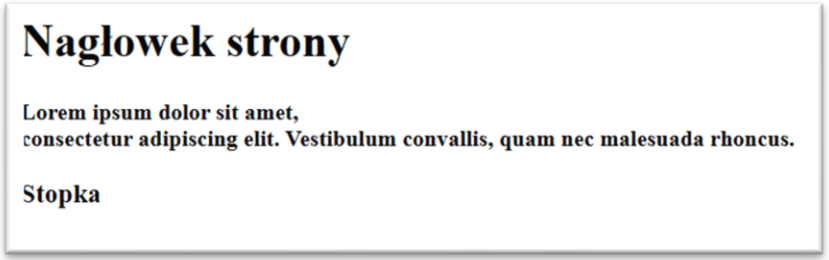
Tradycja to rzecz ważna i należy ją kultywować – więc zgodnie z odwieczną tradycją kursów programowania na początek będzie „Witaj świecie!”

Witaj Świecie!

```
</head>
<body>
    <script>
        document.write("Witaj Świecie!");
        console.log("WItaj Świecie w konsoli");
    </script>
</body>
</html>
```

Formatowanie tekstu

```
  <body>
    <script>
      document.write("<h1>Nagłowek strony</h1>");
      document.write(
        "<b>Lorem ipsum dolor sit amet, <br> consectetur adipiscing elit.
         Vestibulum convallis, quam nec malesuada rhoncus.</b>"
      );
      document.write("<h3>Stopka</h3>");
    </script>
  </body>
</html>
```



- ✓ Polecenie document.write() może zawierać także znaczki HTML – będą one interpretowane przez przeglądarkę jako normalny kod.

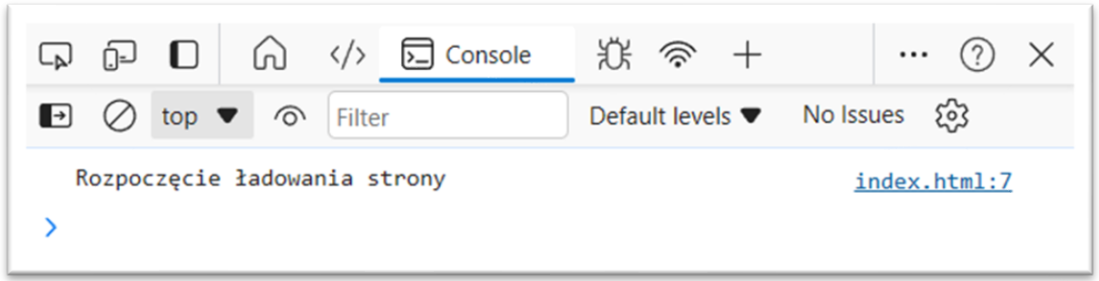
Formatowanie tekstu

Oprócz znaczników HTML w wyświetlanych łańcuchach znakowych mogą też pojawić się znaki specjalne, takie jak np. rozpoczęcie nowego wiersza. Jeśli chcemy wyświetlić znak specjalny, musimy zastosować sekwencję — ukośnik plus litera symbolizująca dany znak

Sekwencja znaków specjalnych	**Znaczenie**
\b	Backspace
\f	wysunięcie kartki (ang. *form feed*)
\n	nowy wiersz (ang. *new line character*)
\r	enter (ang. *carriage return*)
\t	tabulator (ang. *tab character*)

Skrypty w sekcji <head>

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script>
      console.log("Rozpoczęcie ładowania strony");
    </script>
    <title>Document</title>
  </head>
  <body>
  </body>
</html>
```



Metoda document.write sekcji skrypt w <head>

WSKAZÓWKI

Stosowanie znaku średnika na końcu instrukcji JavaScript jest opcjonalne, o ile każdą instrukcję umieszczać będziemy w osobnym wierszu.

Tu stosujemy je w celu podniesienia czytelności kodu.

Na stronie można umieścić dowolną liczbę znaczników <script>, a co za tym idzie, dowolną ilość skryptów.

Komentarze

- ✓ Znacznik <SCRIPT> nie jest częścią specyfikacji HTML 2.0, ani wcześniejszych, więc niektóre przeglądarki mogą go nie rozpoznać. W takiej sytuacji mogą one wyświetlić tekst skryptu na stronie. Chcielibyśmy oczywiście tego uniknąć. Z pomocą przyjdą komentarze, które można umieszczać w kodzie HTML. Konstrukcja wygląda następująco:

```
<!--
Tekst komentarza
-->
```

  - ✓ W skryptach stosujemy dwa typy komentarzy:
    - Liniowy:

```
// Tekst komentarza
```

    - Blokowy

```
/*
    Tekst komentarza
    Tekst komentarza
*/
```

<NOSCRIPT>

Użycie znacznika <noscript> do poinformowania użytkownika, że jego przeglądarka nie obsługuje JavaScriptu.

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <script>
      <!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
      document.write("To jest tekst wstawiony przez JavaScript");
      // Koniec kodu JavaScript -->
    </script>
    <noscript>
      Twoja przeglądarka nie obsługuje JavaScriptów.<br />
      Sugerujemy użycie innej przeglądarki.
    </noscript>
  </head>
  <body></body>
</html>
```

Wstawianie zewnętrznych skryptów

- ✓ oddzielny plik zwykle z rozszerzeniem *js*

```
<script type="text/javascript"
        src="lokalizacja skryptu">
</script>

<script type="text/javascript"
        src="http://nazwa.domeny/skrypty/skrypt.js">
</script>
```

- ✓ zamykający </script> jest niezbędny i nie należy go pomijać.

Część 3 – Zmienne i operatory

Typy zmiennych

- liczbowy (*number*) - nie ma rozróżnienia na typy całkowitoliczbowe i rzeczywiste (zmiennopozycyjne). Liczby zapisywane są za pomocą literałów liczbowych
- łańcuchowy (*string*) - w pojedynczym lub podwójnym cudzysłowie
- logiczny (*boolean*) : true,false
- specjalny typ null – wartość pusta
- obiektowy

- Język JavaScript rozróżnia wielkość liter. Oznacza to, że „Liczba” to nie to samo, co ”liczba”
- Nazwy zmiennych nie mogą zawierać spacji lub znaków przestankowych i polskich liter, a dodatkowo nie mogą zaczynać się od cyfry.

Litarały (wartości zmiennych)

- Literały – wartości zmiennych. np.
  - literał znakowy "złoty", 'kolor'
  - literał liczbowy 2.35e-4, 0.1E-1
- Znaki specjalne w literałach znakowych (stałe znakowe) - jak w C \b , \f, \n, \r, \t, \', \\
- Znak w szesnastkowym kodzie Unicode \uXXXX np. "\u0041" - znak"A"
- Przykłady literałów liczbowych
  - 3.142 liczba zmiennoprzecinkowa
  - 314E-2 liczba zmiennoprzecinkowa
  - 26 liczba całkowita
  - 076 liczba ósemkowa
  - 0x9F liczba szesnastkowa

- 0 - liczba która zaczyna się od „0” jest traktowana jako ósemkowa
- 0x - liczba szesnastkowa

Deklarowanie zmiennych

- Zmienna nie ma ustalonego typu – może przyjmować dowolną typ wartości.
- Ta sama zmienna może być wykorzystywana w różnych momentach do przechowywania liczb i łańcuchów

- Uwaga na zasięg zmiennych!

Deklarowanie zmiennych

Deklarowanie zmiennych może odbywać się za pomocą trzech słów kluczowych:

- var – deklaruje zmienną o zasięgu funkcji lub globalnym.
- let – deklaruje zmienną o zasięgu bloku.
- const – deklaruje stałą (niezmienną referencję).

Każde z tych słów kluczowych ma swoje zastosowanie i różni się od pozostałych w sposobie deklaracji i przypisywania wartości do zmiennych.

Deklarowanie zmiennych

Deklarowanie zmiennych za pomocą var

Deklaracja zmiennej globalnej lub lokalnej

```
{
  var x = 2;
}
document.write(x);
// x może być użyte poza
blokiem
```

- Zmienne zadeklarowane za pomocą var **zawsze mają zakres globalny**
- Zmienne zadeklarowane za pomocą słowa kluczowego var **NIE mogą mieć zakresu bloku**
- Zmienne zadeklarowane za pomocą słowa kluczowego var **mogą być ponownie zadeklarowane**

Deklarowanie zmiennych

Deklarowanie zmiennych za pomocą var
Zmienne istnieje także PRZED miejscem, w którym ją zadeklarowano

```
<script>
  document.write(x);
  {
    var x = 2;
  }
</script>
```

```
var x: number
```

```
> document.write(x);
  {
    var x = 2;
  }
< undefined
```

Przypisanie wartości nie przenosi się „w górę”

- Nie ma potrzeby definiowania zmiennych na początku funkcji – ale jest to zalecane
- Uwaga na zasięg zmiennych!

Deklarowanie zmiennych

Deklarowanie zmiennych za pomocą let

Deklaracja zmiennej o zakresie bloku

```
{
  let x = 2;
}
document.write(x);
// x NIE może być użyte poza blokiem
```

- Przed ES6 (2015) JavaScript nie miał zakresu bloku - miał zakres globalny i zakres funkcji.
- W ES6 wprowadzono dwa nowe słowa kluczowe JavaScript: let i const zapewniły one zakres bloku

Deklarowanie zmiennych

Deklarowanie zmiennych za pomocą let
Zmienne nie istnieje poza blokiem, w którym ją zadeklarowano.

```
> document.write(x);
  {
    let x = 2;
  }
▶ Uncaught ReferenceError: x is not   VM231:1
```

- Nie ma potrzeby definiowania zmiennych na początku funkcji – ale jest to zalecane
- Uwaga na zasięg zmiennych!

Deklarowanie zmiennych

Ponowna deklaracja zmiennych

```
 var x = 10;
// Tu x wynosi 10
{
  var x = 2;
  // Tu x wynosi 2
}
// Tu x wynosi 2
```

Ponowne zadeklarowanie zmiennej przy użyciu słowa kluczowego var może spowodować problemy.

Ponowna deklaracja zmiennej wewnątrz bloku spowoduje również ponowne zadeklarowanie zmiennej poza blokiem

```
let x = 10;
// Tu x wynosi 10
{
  let x = 2;
  // tu x wynosi 2
}
// tu x wynosi 10
```

Ponowne zadeklarowanie zmiennej przy użyciu słowa kluczowego let może rozwiązać ten problem.

Ponowna deklaracja zmiennej wewnątrz bloku nie spowoduje ponownego zadeklarowania zmiennej na zewnątrz bloku

Operatory

- Operatory arytmetyczne: +, -, /, *, %
- Operatory zwiększania i zmniejszania: -- , ++
- Operatory porównania: ==, !=, >, <, >=, <=
  === równe wartością i typem
  !== różne wartością i typem
- Operatory logiczne: &&, ||, !
- Operatory przypisania: =, +=, -=, *=, /=, %=
- Operatory bitowe: >>, <<, &, |, ^, ~

Konwersja typów i konkatenacja

- ✓ JavaScript - język o łagodnej kontroli typów
- ✓ Zmienne i literały różnych typów można bez potrzeby wcześniejszego uzgadniania typów porównywać, łączyć itp.
- ✓ Możliwe jest łączenie typu znakowego i liczbowego operatorem "+" – wynik typu znakowego

Przykład - najprostszy przypadek konwersji typów:

```
t="2000";
t+=10;   //  -> Wynik: t="200010"
```

Operatory === i ==

Operator ===

a === b

typ(a) jest równe typ(b)
AND
a jest równe b

Operator === zwraca wartość prawda jeżeli zmienne są takiego samego typu i mają taką samą wartość

Operatory === i ==

Operator ==

```
if( typ(a) jest równe typ(b) )
{
        a === b
}
else
{
        numer(a) === numer(b)
}
```

Operator ==

- jeżeli zmienne są takiego samego typu - porównuje ich wartości
- Jeżeli zmienne różnią się typem, próbuje przekonwertować je na typ numer i dopiero wtedy dokonuje porównania

Operatory === i ==

Operator ==

- jeżeli zmienne są takiego samego typu - porównuje ich wartości
- Jeżeli zmienne różnią się typem, próbuje przekonwertować je na typ numer i dopiero wtedy dokonuje porównania

```
> let x = 1;
  if (x) console.log("TAK");
  else console.log("NIE");
TAK
```

```
> let x = 1;
  if (x == true) console.log("TAK");
  else console.log("NIE");
TAK
```

```
> let x = 1;
  if (x === true) console.log("TAK");
  else console.log("NIE");
NIE
```

Wartość NaN

Wartość NaN w języku JavaScript oznacza "Not-a-Number" i jest używana do reprezentowania wartości, która nie jest liczbą w sensie arytmetycznym. W sytuacjach, gdy operacja arytmetyczna nie może być przeprowadzona na zwykłej liczbie, JavaScript zwraca wartość NaN.

Główne sytuacje, w których wartość NaN może wystąpić, to:

- Dzielenie przez zero: W JavaScript próba dzielenia przez zero zwraca NaN

```
let wynik = 10 / 0; // zwróci NaN
```

- Operacje arytmetyczne na wartościach, które nie są liczbami: Jeśli wykonujesz operacje arytmetyczne na nieprawidłowych wartościach, wynik będzie NaN.

```
let wynik = "abc" * 10; // zwróci NaN
```

- Nieudane parsowanie tekstu na liczbę: Próba parsowania tekstu, który nie jest liczbą, do liczby również zwróci NaN.

```
let wynik = Number("abc"); // zwróci NaN
```

Wartość NaN

Wartość NaN ma pewne unikalne właściwości. Na przykład, każda operacja porównania (także z samym sobą) zwraca false:

```
console.log(NaN === NaN); // false
```

Aby sprawdzić, czy wartość jest NaN, można użyć funkcji isNaN():

```
let x = NaN;
if (isNaN(x)) {
  console.log("Wartość x jest NaN");
} else {
  console.log("Wartość x nie jest NaN");
}
```

Wartość NaN jest przydatna w przypadku wykrywania błędów lub nieprawidłowych operacji arytmetycznych w kodzie JavaScript.

Łańcuchy znaków

Stringi

W JavaScript istnieją trzy rodzaje znaków cudzysłowu do oznaczania łańcuchów znaków (stringów):

- Pojedynczy apostrof (’)
- Podwójny cudzysłów (”)
- Backtick (tylda odwrotna) (`)

```
const a = 'tekst';
const b = "tekst";
const c = `tekst`;
```

Rodzaj	Można użyć zmiennych (interpolacja)?	Wieloliniowość	Zastosowanie
'...'	nie	nie	standardowy string
"..."	nie	nie	standardowy string
`...`	TAK	TAK	szablony tekstowe, HTML, generatory kodu

Stringi

Przykład interpolacji zmiennych (template literals)

```
const imie = "Anna";
const powitanie = `Witaj, ${name}!`;

console.log(powitanie); // Witaj, Anna!
```

Przy '...' lub "..." to by nie zadziałało – należałoby łączyć stringi ręcznie:

Stringi

Dostęp do znaków w stringach:

```
const word = "JavaScript";
console.log(word[0]); // "J"
console.log(word.charAt(1)); // "a"
```

Stringi

Metody przydatne w pracy na stringach:

length	Długość łańcucha
toLowerCase()	Zmienia wszystkie litery na małe
toUpperCase()	Zmienia na wielkie litery
includes(sub)	Sprawdza, czy zawiera podciąg
startsWith(sub)	Czy zaczyna się od
endsWith(sub)	Czy kończy się na
indexOf(sub)	Indeks pierwszego wystąpienia
slice(start, end)	Wycina fragment
substring(a, b)	Podłańcuch bez indeksów ujemnych
replace(old, new)	Zastępuje pierwszy napotkany fragment
trim()	Usuwa białe znaki z początku i końca
split(delimiter)	Rozdziela string na tablicę

Stringi

Przykład:

```
const input = "  JavaScript jest super!  ";
const clean = input.trim().toUpperCase();

console.log(clean); // "JAVASCRIPT JEST SUPER!"
```

- Stringi w JS są niemodyfikowalne – każda operacja zwraca nowy string, nie zmienia oryginału.

Literatura:

- Negrino Tom, Smith Dori, ***Po prostu JavaScriopt 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