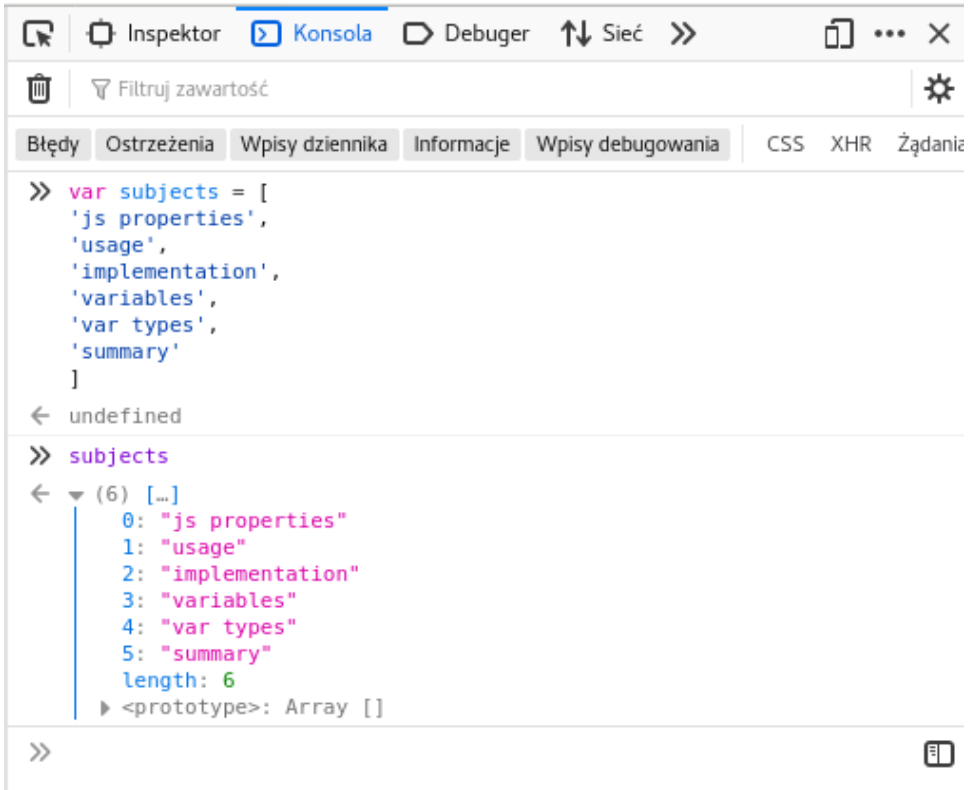
Programowanie Internetowe

- Standard ECMA Script
- Obiektowy model dokumentu
- Data, Czas, Wyrażenia regularne
- Przechowywanie danych
- Przetwarzanie asynchroniczne

Opracował: inż. Grzegorz Petri

Przegląd zagadnień

- JavaScript to nie JAVA
- Standard ECMA
- Cechy, wady i zalety języka
- Zastosowania technologii
- Implementacje standardu
- Załączanie skryptów
- Składowe języka
- Zmienne oraz typowanie



JavaScript ≠ Java

Podobieństwa i różnice języków

JavaScript (LiveScript)	***JAVA***
Kod interpretowany w programie klienckim (*np. przeglądarka internetowa uruchamiająca kod JavaScript - aplikację*)	Kod kompilowany do *bajtkodu*, wykonywanego w maszynie wirtualnej (*dostarczanej klientowi wraz z aplikacją*)
Zorientowany obiektowo (*bez wszystkich mechanizmów obiektowości*)	W pełni obiektowy – zorientowany na klasy
Dynamicznie i słabo typowany	Statycznie i silnie typowany
Użycie zmiennych bez ich deklaracji	Zmienne wymagają deklaracji przed użyciem
Brak modyfikatorów dostępu (*zasięg scopes*)	Modyfikatory dostępu do klas i zmiennych
Odwołania do obiektów oraz funkcji są wykonywane podczas uruchamiania skryptu	Odwołania do obiektów oraz funkcji są sprawdzane na etapie kompilacji programu
Brak dostępu do dysku twardego*	Pełen dostęp do dysku twardego oraz środowiska systemu operacyjnego

Standard ECMA Script

ECMA-262

- Edycja 1: 1997
- Edycja 2: 1998
- Edycja 3: 1999
- Edycja 4: porzucona*
- ***Edycja 5: 2009 (strict mode, struktura JSON)***
- Edycja 6: 2015 (class, notacja strzałkowa ()=>{})

Wady i zalety technologii

Zalety

- ✔ Przeniesienie części działań serwera na stronę klienta
- ✔ Szybsza komunikacja z klientem
- ✔ Zwiększona interakcja z klientem
- ✔ Bogatszy interfejs użytkownika
- ✔ Prostota języka oraz popularność technologii

Wady

- ✗ Mniejsze bezpieczeństwo
- ✗ Możliwość wyłączenia
- ✗ Brak dostępu do dysku klienta
- ✗ Brak wsparcia dla:
  - ✗ aplikacji sieciowych*
  - ✗ wielowątkowości**

** HTML5 WebSockets*
*** HTML5 WebWorkers*

Zastosowania technologii

Client-side (*przeglądarki*)

- Interakcje z użytkownikiem: efekty, wyliczenia, wczytywanie danych
- Tworzenie multi-platformowych aplikacji i gier webowych
- Rozszerzenia (WebExtensions)

Server-side (*serwery*):

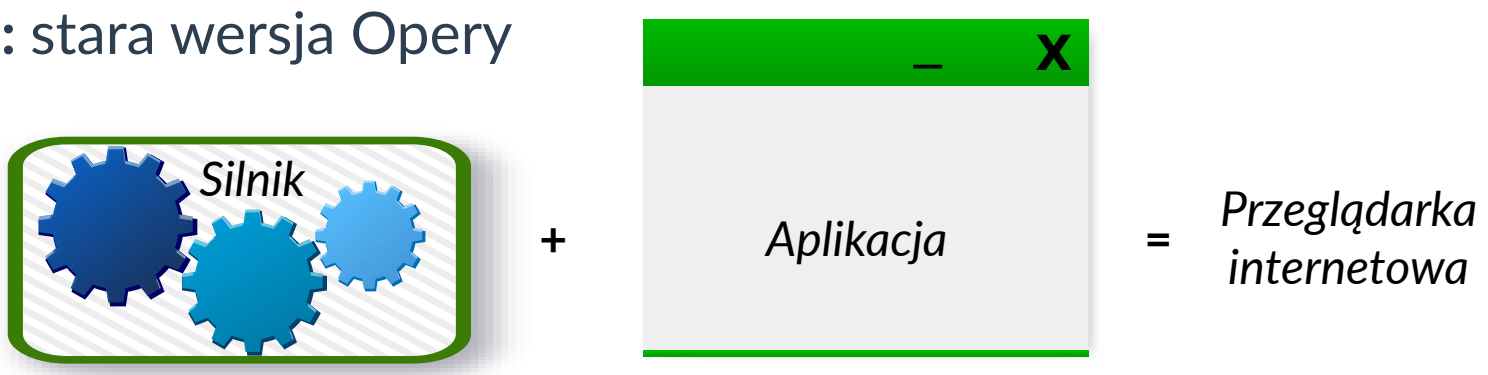
- Node.js
- ASP JScript
- IBM Domino
- Apache CouchDB

- **Electron (Atom Shell)** *framework pozwalający tworzyć aplikacje desktopowe*
- **Apache Cordova** *framework do aplikacji mobilnych niezależny od platformy*
- **Adobe Acrobat** *wsparcie używania skryptów w dokumentach PDF*
- **GNOME Shell** *rozszerzenia powłoki systemowej zwiększające możliwości*
- **Microsoft Excell** *wschodzącą i multiplatformowa alternatywa dla makr VBS*

Implementacje standardu ECMA Script

Silniki JavaScript

1. **SpiderMonkey:** produkty Mozilla (*Firefox*, *Thunderbird*, *itd.*)
2. **Rhino**: open-source'owa implementacja w JAVA osadzona w J2SE 6
3. **Google V8:** przeglądarka Google Chrome, usługi Googla, Node.js
4. **JavaScriptCore:** w części przeglądarek opartych o silnik WebKit (*Apple*)
5. **Chakra/JScript:** MS IE, MS IIS, MS Office
6. **Carakan:** stara wersja Opery



Załączanie skryptów

Przestrzeń globalna

W obrębie dokumentów HTML dla plików JavaScript obowiązują podobne zasady, co do arkuszy stylów CSS:

1. Kod skryptu w znaczniku **<HEAD> … </HEAD>**
2. Kod skryptu w znaczniku **<BODY> … </BODY>**
3. Załączenie skryptu w znaczniku **<HEAD> … </HEAD>**
4. Załączenie skryptu w znaczniku **<BODY> … </BODY>**

Przestrzeń globalna 

Wszystkie załączone skrypty oraz kody umieszczone w dokumencie HTML posiadają wspólna przestrzeń funkcjonowania, czyli zmienne, funkcje i obiekty są dostępne dla wszystkich skryptów danej witryny.

Załączanie skryptów

1. Kod skryptu w znaczniku **<HEAD> … </HEAD>**
2. Kod skryptu w znaczniku **<BODY> … </BODY>**

✗ *Niska przenośność kodu*

✗ *Modyfikacje muszą być wprowadzane w KAŻDYM dokumencie*

✗ *Ingerowanie w dokument – mieszanie warstwy logiki oraz prezentacji*

```
<SCRIPT type="text/javascript">
<!-- Kod skryptu //-->
</SCRIPT>
```

Załączanie skryptów

3. Załączenie skryptu w znaczniku `<HEAD> ... </HEAD>`
4. Załączenie skryptu w znaczniku `<BODY> ... </BODY>`

- ✔ *Wysoka przenośność kodu*
- ✔ *Modyfikacje wprowadzane w JEDNYM dokumencie*
- ✔ *Brak ingerencji w dokument – oddzielenie warstwy logiki od prezentacji*

```
<SCRIPT type="text/javascript"
        src="ścieżka/do/pliku.js">
</SCRIPT>
```

Składowe elementy

- **Typy danych**
- **Zmienne**
- **Operatory**
- **Metody**
- *Obiekty*
- *Właściwości*
- *Zdarzenia*

Typy danych

Istnieje 9 typów danych w JavaScript

Typy danych prymitywnych*:

1. undefined
2. Boolean
3. Number
4. String
5. Symbol *(ES6/ES2015)*
6. BigInt *(ES11/ES2020)*

* *Nie jest Obiektem, nie posiada Metod*

Inne typy:

7. **Null** - *rodzaj specjalnego typu oznaczającego brak wartości.*
8. **Object** – *typ specjalny, struktury, non-data type (typ nie-danych).*
9. **Function** – *pomimo typu struktury (non-data) odpowiada na operator* ***typeof****, a jej konstruktor pochodzi od konstruktora Obiektu.*

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

Sprawdzanie typu

- Aby sprawdzić typ zmiennej lub wartości powstał operator `typeof`, który zwraca tekstową (*string*) reprezentację typu.
- Składnia: `typeof operand` LUB `typeof (operand)`
- Operand: zmienna lub wartość

```
> typeof 13                 // „number”
> typeof ‘ciasteczka’       // „string”
> typeof true               // „boolean”
> typeof brakZminnej        // „undefined”
```

Deklarowanie zmiennych

- `var` *podstawowy sposób deklarowania zmiennych*
  - *zasięg funkcyjny, czyli całość skrypt(ów)*
  - *możliwe użycie zmiennej bez przypisania wartości*
  - *ponowna deklaracja tej samej zmiennej wciąż możliwa*
- `let` *sposób wprowadzony w ES6 celem wyeliminowania niejasności* ***var***
  - *zmienna o zasięgu blokowym*
  - *wymaga deklaracji przed jej użyciem*
  - *ponowna deklaracja tej samej zmiennej nie możliwa*
- `const` *zmienna typu stałego*
  - *Przypisanie wartości uniemożliwia jej zmianę*

Deklarowanie zmiennych

Przykłady

1) var
2) let
3) const

```
var v1 = 'test';
function v1test(){
   var v1 = 'atest';
   console.log(v1);// atest
}
v1test();          // atest
console.log(v1);// test
```

1

```
console.log(v2);// error
let v2 = 'fly';
console.log(v2);// fly
```

2

```
const pi = 3.14;
const g = 9.81;
console.log(pi,g);
g = 11; //redeclaration
console.log(pi,g); // error
```

3

Operatory

+	Dodawanie	-=	Odejmowanie	<	mniejsze od
-	Odejmowanie	*=	Mnożenie	>=	większe bądź równe od
*	Mnożenie	/=	Dzielenie	<=	mniejsze bądź równe od
/	Dzielenie	%=	Reszta z dzielenia	&&	and (i)
%	Reszta z dzielenia	==	równe	||	or (lub)
++	Inkrementacja	!=	różne	^	xor
--	Dekrementacja	===	taki sam typ danych i wartość	!	not (negacja)
=	Przypisanie	!==	różne wartości lub różny typ danych		
+=	Dodawanie	>	większe od		

Deklarowanie metod

- Metoda (funkcja) posiada słowo kluczowe function
- Musi zawierać nazwę funkcji, chyba, że jest to anonimowa metoda
- Może zawierać parametry, oraz domyślne parametry w bloku znaków ( )
- Musi zawierać ciało metody w bloku pomiędzy znakami {}

```
var pierwsza = 1;
function pierwsza()
{  // ciało metody
}
typeof pierwsza;  // ??
```

```
var druga = 2;
var druga = function()
{  // ciało metody
}
typeof druga;  // ??
```

Parametry metod

- ✔ Może zawierać parametry,
- ✗ jednak próba użycia zmiennych niezainicjowanych wartością spowoduje zwrócenie błędu NaN (*Not a Number*).

- ✔ *Stosowanie domyślnych parametrów w bloku argumentów ( ) znosi z programisty ciężar wprowadzania zabezpieczeń*

```
function suma(a,b){
typeof b==='undefined' ?
b=1 : null;
}
suma(1) // wynik?
```

```
function suma(a,b=2)
{
// sprawdzanie jest zbędne
}
suma(1);  // wynik?
```

Wywoływanie metody i jej wynik

- ✔ Może zwracać wynik
- ➔ Wymaga operatora `return`

- ✗ Nie musi zwracać wyniku
- ➔ Wykonywane obliczenia lub działania na elemencie

```
function suma(a=1,b=2)
{
    return a+b;
}
// wywołanie metody
var wynik = suma();
wynik; // wywołanie zmiennej
```

```
var wynik = 0;
function suma(a,b)
{
    wynik = a+b;
}
suma(a,b)// wywołanie metody
wynik; // wywołanie zmiennej
```

Konsola

Proste informowanie o błędach

- ✔ Metoda `Window.alert()`
- ✔ Wywołanie w oknie przeglądarki (*przestrzeń renderowania strony*)
- ✔ Okno modalne blokujące interakcje ze stroną aż do zamknięcia okna przyciskiem OK

```
function suma(a=1,b=2){
    alert(a+b);
    return a+b;
}

alert(suma()); // wynik
```

- ✔ Metoda `log()` obiektu `Console`
- ✔ Obiekt `Console` daje dostęp do debugera przeglądarki (*zwykle przez narzędzia developerskie*)

```
var wynik = 0;

function suma(a,b)

{
    wynik = a+b;
    console.log( wynik );
}
suma(a,b)// wywołanie metody
console.log( wynik );
```

Więcej informacji o możliwościach obiektu Console: https://developer.mozilla.org/en-US/docs/Web/API/Console

Słabe / dynamiczne typowanie

Wybrane przykłady

Lewy operand	*operator*	*Prawy operand*	*Wynik*
`[] (array)`	`+`	`[] (array)`	`"" (empty string)`
`{} (object)`	`+`	`[] (array)`	`0 (number)`
`[] (array)`	`+`	`{} (object)`	`{} (empty object)`
`false (boolean)`	`+`	`[] (array)`	`"false" (string)`
`„123” (string)`	`+`	`1 (number)`	`"1231" (string)`
`„123” (string)`	`-`	`1 (number)`	`122 (number)`

```
dialog(
  ‘Pytania?’
);
```