Tłumaczymy żarty

Typy proste, dodawanie vs konkatenacja, 42, korzystanie z bibliotek zewnętrznych

Olga Bednarczyk

Za: Programista płakał jak commitował

Plan wykładu

1. Typy danych
    - Typy proste
    - Obiekty
2. Wymuszona konwersja typów danych
3. Operacje logiczne
    - Truthy vs falsy
    - Operatory logiczne
4. Const vs var vs let
5. Przydatne biblioteki
6. Co dalej

Typy danych

Typy proste (Primitives)

- Number – liczba: 0, -5, 8.9, NaN
- String – znak lub łańcuch znaków: 'JavaScript rocks!'
- Boolean – typ logiczny: true/false
- Undefined – wartość nie została jeszcze przypisana
- Null – intencjonalny brak wartości

Wszystko inne jest obiektem!

- obiekt (Object) – { name: 'Olga', age: 26 }
- tablica (Array) – ['wow', 'wow', null, 15]
- funkcja – function foo () { return 'Hello' }
- Set – 'tablica' unikalnych elementów
- Date – obiekt daty
- so on and so forth...

Jak sprawdzić typ zmiennej?

```
typeof operand
// OR
typeof (operand)
```

```
typeof 'Christmas time' // => 'string'
typeof 25 // => 'number'
typeof true // => 'boolean'
typeof undeclaredVar // => 'undefined'
typeof (() => console.log('whatever')) // => 'function'
typeof [1, 3, 7] // => 'object'
typeof { a: 1, b: 2 } // => 'object'
typeof null // => 'object'
```

Wymuszona konwersja (rzutowanie) typów danych

Console | What's New

top | Filter

```
> 2 + 2
< 4
> "2" + "2"
< "22"
> 2 + 2 - 2
< 2
> "2" + "2" - "2"
< 20
```

Definicja

"Type coercion is the automatic or implicit conversion of values from one data type to another (such as strings to numbers). Type conversion is similar to type coercion because they both convert values from one data type to another with one key difference — type coercion is implicit whereas type conversion can be either implicit or explicit."

Definicja na 'chłopski rozum'

“Zmiana typu danych, kiedy JavaScript nie umie inaczej wykonać zadanej operacji”

Dodawanie vs konkatenacja (łączenie)

Działania, w których operatorem jest '+'

Dodawanie

```
2 + 5 // => 7
```

Konkatenacja

```
'It is so' + 'useful' // => 'It is so useful'
'2' + '5' // => '25'
```

Nie można dodawać stringów i liczb => konwersja liczby w string => konkatenacja

```
'To nasz ' + 6 + ' wykład' // => 'To nasz 6 wykład'

0 + '3' // => '03'

'23' + 1 // => '231'
```

Nie można odejmować stringów i liczb => konwersja stringa w liczbę => odejmowanie

```
'10' - 5 // => 5

2 - '8' // => -6

'Nope' - 1 // => NaN

34 - 'Another nope' // => NaN
```

JS konwertuje stringi inne niż liczbowe do NaN (Not a Number)

Nie można łączyć stringów i booleanów => konwersja booleana w string => konkatenacja

```
'Staph it!' + true // => 'Staph it!true'

false + '4' // => 'false4'
```

Nie można odejmować stringów i booleanów => konwersja stringa i booleana w liczbę => odejmowanie

```
'33' - true // => 32

'33' - false // => 33

'DaftAcademy' - true // => NaN
```

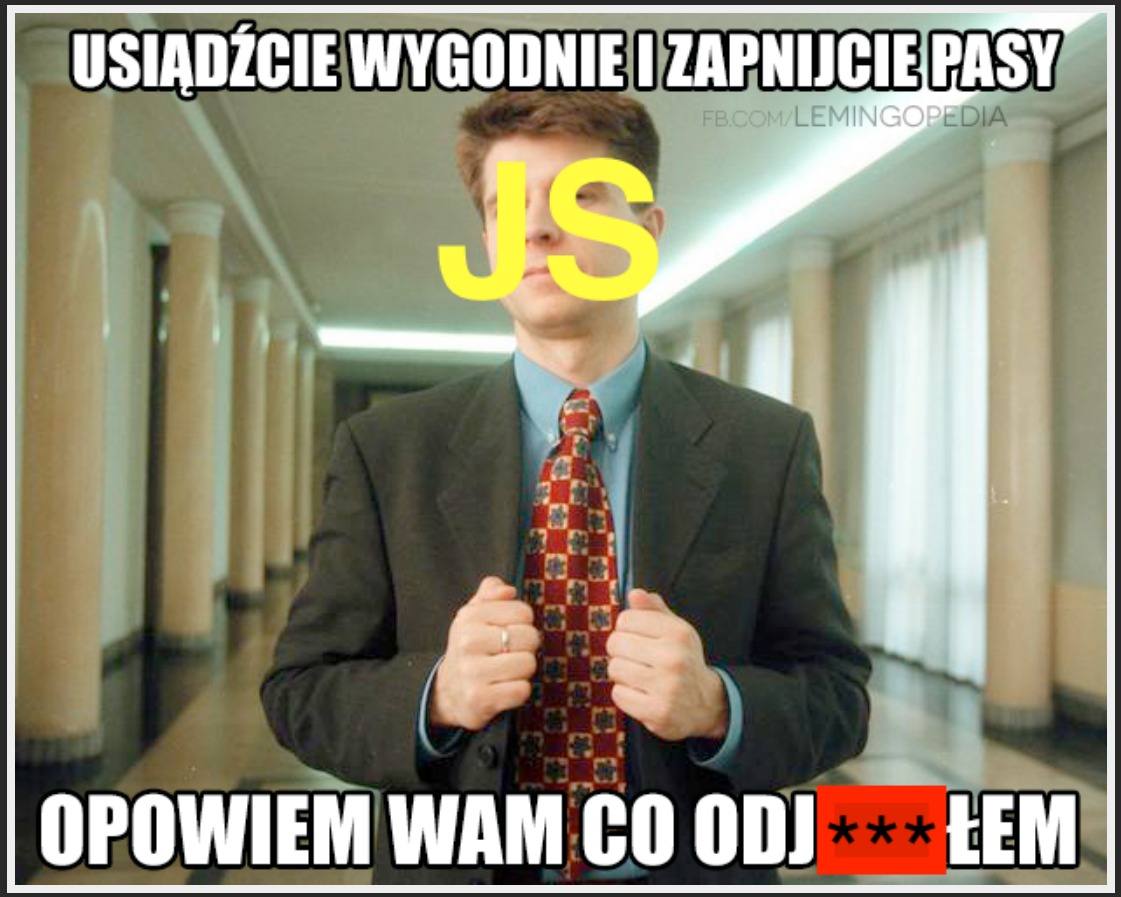
True ewaluuje się do 1, false – do 0

Nie można dodawać/odejmować liczb i booleanów => konwersja booleana w liczbę => dodawanie/odejmowanie

```
33 + true // => 34

33 - false // => 33
```

True ewaluuje się do 1, false – do 0


USIĄDŹCIE WYGODNIE I ZAPNIJCIE PASY
FB.COM/LEMINGOPEDIA
JS
OPOWIEM WAM CO ODJ***ŁEM

Nie można dodawać tablic => konwersja ich elementów i rozdzielających przecinków w stringi => konkatenacja

```
['2', '2'] + ['3', '3'] // => '2,23,3'

[33, '1'] + [2] // => '33,12'

['Ala', 'Ola'] + ['Ela', 'Ula'] // => 'Ala,OlaEla,Ula'

['kot'] + [] + ['ek'] // => 'kotek'

[] + [] // => ''
```

Nie można odejmować tablic => konwersja ich elementów w liczby => odejmowanie

```
[15] - [2] // => 13

['4'] - ['5'] // -1

[4] - [] // => 4

[] - [] // => 0

[2, 8] - [5, 6, 4] // NaN
```

PS. Mnożenie i dzielenie działają analogicznie do odejmowania

Więcej przykładów i wyjaśnień – link

'42'

```
[]+(-~{}-~{}-~{}-~{})+(-~{}-~{}) //=>  '42'
```

{} – podczas odejmowania ewaluuje się do 0

~ – bitowy operator NOT; konwertuje 0 do -1, bo ~x = -(x + 1)

```
-~{} // => 1
(-~{}-~{}-~{}-~{}) // =>  4
(-~{}-~{}) // => 2
```

[] – podczas dodawania konwertuje się do '' (pusty string), co wymusza konwersję innych elementów do stringów

```
[] + 4 + 2 // => '' + '4' + '2' => '42'
```

Czemu akurat '42'?

Odpowiedź tutaj

(tak przynajmniej twierdzi Mikołaj)

Operacje logiczne

Wartości truthy i falsy

Falsy – ewaluują się do false

- undefined
- null
- 0
- '' (pusty string)
- NaN
- false

Truthy – ewaluują się do true

Wszystkie pozostałe, np.:

- 90, 2.3, -5
- 'Wow, wow', 'a', ' '
- [a, b, c], []
- { lectureId: 6 }, {}
- true

Operatory

- && – i
- || – lub
- ! – negacja
- !! – podwójna negacja
- == – równość
- === – identyczność
- != – nierówność
- !== – nieidentyczność

&& – i

Oba warunki muszą być spełnione

```
true && true // => true

true && false // => false

false && true // => false

false && false // => false
```

|| – lub

Przynajmniej jeden z warunków musi być spełniony

```
true && true // => true

true && false // => true

false && true // => true

false && false // => false
```

! – negacja

Zmiana true na false i odwrotnie

```
!true // => false

!false // => true
```

!! – podwójna negacja

Ewaluacja zmiennej do typu Boolean

```
!!true // => true

!!false // => false

!!'' // => false

!!undefined // => false
```

== – równość

Porównanie wartości, ale nie typu (występuje wymuszona konwersja typów)

```
2 == 2 // => true

2 == '2' // => true

2 == 3 // => false

null == undefined // => true
```

`===` – identyczność

Porównanie wartości oraz typu (nie występuje wymuszona konwersja typów)

```
2 === 2 // => true

2 === '2' // => false

null === undefined // => false
```

!= – nierówność

Porównanie nierówności wartości, ale nie typu

```
2 != 2 // => false

2 != '2' // => false

2 != 3 // => true

null != undefined // => false
```

!== – nieidentyczność

Porównanie nierówności wartości oraz typu

```
2 !== 2 // => false

2 !== '2' // => true

2 !== 3 // => true

null !== undefined // => true
```

Bloki warunkowe

- if ... else – poczytać tutaj
- switch statement – poczytać tutaj
- ternary operator (operator warunkowy) – poczytać tutaj

Ternaty operator (operator warunkowy)

```
warunek ? jeśliSpełniony : jeśliNieSpełniony
```

```
const isGuest = true
isGuest ? '5$' : 'free' // => '5$'

1 === 2 ? 'wow' : 'nie wow' // => 'nie wow'

isGuest && 0 ? '5$' : 'free' // => 'free'
```

Podczas sprawdzania warunków też ma miejsce wymuszona konwersja typów (type coercion).

Wszystkie zmienne są ewaluowane do typu Boolean – JS 'sprawdza' czy są truthy czy falsy.


0 == "0"
true
JS
0 == []
true
JS
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false
JS

Wyjaśnienie

```
0 == '0' // => true – wymuszona konwersja '0' do liczby 0

0 == [] // => true – wymuszona konwersja [] do liczby 0

'0' == [] // => false – nie można przekonwertować '0' do []
                        ani [] do '0'
```

Inne przykłady

```
true + true + true == 3 // => true
undefined != null // => false
({} && 0) ? true : false // => false
'0' || 1 // => true
!![] // => true
!0 // => true
!'0' // => false
2 === '2' // => false
{ a: 1 } !== { a: 1 } // => true
{ a: 1 } != { a: 1 } // => true
```

Obiekty są porównywane po referencjach, więc nigdy nie są sobie równe. Żeby sprawdzić, czy są takie same, porównujemy ich kolejne elementy.

Tabela zasad wymuszonej konwersji typów – link

Const vs let vs var

Var

- Pozwala przypisywać nowe wartości zmiennym i redefiniować je

```
var name = 'Miki';
var age = 23;

var age = 30
name = 'Mouse'

console.log(name, age) // => 'Mouse', 30
```

Var

- Zasięg (scope) globalny lub funkcyjny

```
var someVar = 5;

function foo () {
  var anotherVar = 10;
  console.log(someVar);
};

foo(); // => 5
console.log(anotherVar); // => ERROR: anotherVar is
                         //            not defined
```

- Jest windowane (hoisting) z wartością undefined

```
console.log(myVariable); // => undefined
var myVariable = 5;
```

Let

- Pozwala przypisywać nowe wartości zmiennym, ale nie – redefiniować ich

```
let name = 'Miki';
let age = 23;

name = 'Mouse'
console.log(name) // => 'Mouse'

let age = 30; // ERROR: Identifier 'age'
              // has already been declared
```

Let

- Zasięg (scope) blokowy (blok: kodu w {})

```
let someVar = 5;

if (2 == 2) {
  let anotherVar = 10;
  console.log(someVar);
};

foo(); // => 5
console.log(anotherVar); // => ERROR: anotherVar is
                         //            not defined
```

- Jest windowane (hoisting) bez inicjalizacji

```
console.log(myVariable); // => Cannot access 'myVariable
                         // before initialization
let myVariable = 5;
```

Const

- Przypisujemy stałą, ale możemy ją mutować

```
const name = 'Miki';
name = 'Mouse'; // TypeError: Assignment to
                // constant variable
```

```
const fuits = ['apple', 'orange'];
fruits.push('lemon');
console.log(fruits); // => ['apple', 'orange', 'lemon']
```

- Zasięg (scope) blokowy (blok: kodu w {}) - analogicznie do let
- Jest windowane (hoisting) bez inicjalizacji - analogicznie do let

Reasumując:

- Zawsze używaj słów kluczowych do definiowania zmiennych. Inaczej zmienne przypiszą się do obiektu window;
- Nie definiuj zmiennych globalnie, można jest wtedy zmienić, zmutować z dowolnego miejsca aplikacji;
- Nie używaj var;
- Używaj const, kiedy tylko to możliwe. Możliwe jest baaaardzo często.

Przydatne biblioteki

Biblioteki JavaScriptu

- Ramda, Lodash – dużo przydatnych funkcji
- Wow.js – animacje po przeskrolowaniu do elementu
- Moment.js – dużo funkcji związanych z datą i czasem
- Three.js – grafika 3D
- Popper.js – tooltipy

Korzystanie z bibliotek zewnętrznych

- Instalacja przez NPM – bardziej zaawansowane i najlepsze; nie na poziom tego kursu, ale warto się zapoznać – link
- Link do CDN (Content Delivery Network)
- Skopiowanie skryptu do swojego projektu

Link do CDN (Content Delivery Network)

```html
<head>
    <title>6BM Band Official Website</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,600,700&display=swap&subset=latin-ext"
          rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="./style.css" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/ramda/0.25.0/ramda.min.js" defer></script>
    <script src="./script.js" defer></script>
</head>
```

```js
const arr = [1, 2, 4];
R.isEmpty(arr); // => false

const sentence = 'Ala ma kota';
R.replace('Ala', 'Ola', sentence); // => 'Ola ma kota'
```

Co dalej

Front-end Developer Roadmap