Wstęp - JavaScript

v 1.13

Coders Lab
SZKOŁA PROGRAMOWANIA

PLAN

- Trochę teorii
- O języku JavaScript
- Pierwszy skrypt
- Komentarze i oddzielanie instrukcji
- Zmienne
- Typy danych
- Operatory
- Kontrola przepływu programu
- Tablice
- Dobre praktyki
- Najczęstsze błędy początkujących
- Przydatne linki

Trochę teorii

Czym jest komputer?

Komputer to maszyna przeznaczona do przetwarzania informacji.

Współczesne komputery od wszystkich innych maszyn odróżnia **możliwość ich programowania**, czyli wprowadzenia do pamięci komputera listy instrukcji, **które mogą być wykonane w innym czasie**.

Co to jest algorytm?

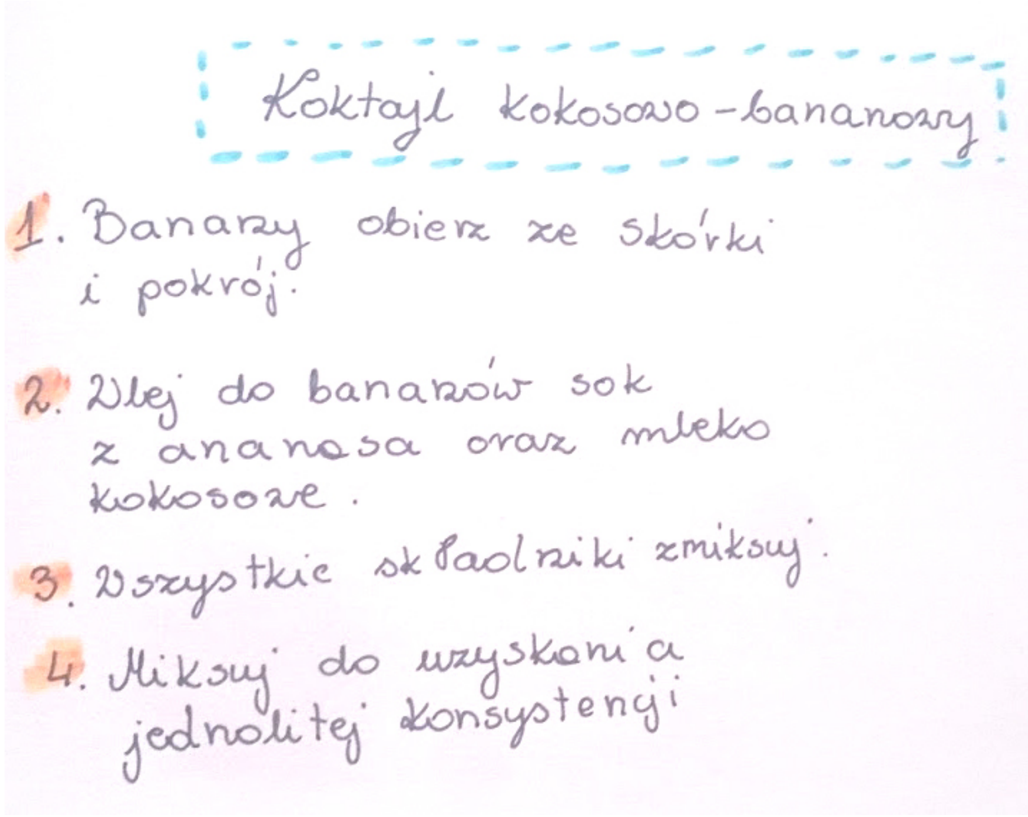
Algorytm – skończony ciąg jasno zdefiniowanych czynności koniecznych do wykonania pewnego rodzaju zadań.

Gdzie na co dzień spotykamy się z algorytmami?

➢ W kuchni – wszystkie przepisy.

➢ Na drogach – zasady ruchu drogowego.

➢ Praktycznie w każdej dziedzinie naszego życia.



Pseudokod

Definicja

- Pseudokod to popularna notacja, w której zapisywane są algorytmy.
- Nie jest językiem programowania.
- Składa się z instrukcji sterujących, zmiennych przypisanych. Często też z opisu tekstowego w języku naturalnym.
- Nie ma bardzo sztywnych reguł.

Przykład

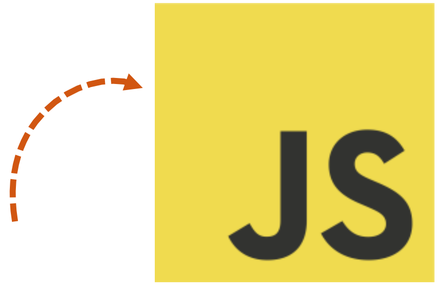
```
funkcja foo(a, b)
  dopóki a ≠ b
    jeżeli a > b
      a = a − b
    inaczej
      b = b − a
  zwróć a
```

Język programowania

Język programowania

- Jest to **zbiór zasad**, które opisują jak należy budować instrukcje, tak aby komputer je rozumiał.

- Od języka naturalnego odróżnia go m.in. **jednoznaczność** oraz **precyzja**.
  Przykład: Gdy mówimy, popełniamy różnorodne błędy, a mimo to druga osoba jest w stanie nas zrozumieć. W przypadku porozumiewania się z komputerem nie jest to możliwe.



Ten język programowania będziemy poznawać w trakcie kursu

Kod źródłowy

Kod źródłowy

- Jest to algorytm (lub wiele współpracujących algorytmów) rozwiązujący dany problem.
- Kod jest zapisany w konkretnym języku programowania.
- Zapis ten jest czytelny dla człowieka.
- Obok na zdjęciu: **Margaret Hamilton**, lead software engineer w Projekcie Apollo, a obok niej wydrukowany **kod źródłowy dla AGC** (Apollo Guidance Computer), czyli dla cyfrowego komputera zainstalowanego w modułach na pokładzie Apollo.

Źródło: http://threefingeredfox.net/?p=143

Kompilator i interpreter

Kompilator

- Program, który służy do tego, by przetłumaczyć kod źródłowy na kod maszynowy – czyli taki, który rozumie komputer.
- Proces tłumaczenia to inaczej **kompilacja**
- Komputer nie rozumie żadnego języka programowania tylko **kod maszynowy**, dlatego trzeba mu odpowiednio go przetłumaczyć.
- To nie jest pełna definicja kompilatora, ale na potrzeby kursu i zrozumienia JS wystarczy

Interpreter

- Program, który analizuje kod źródłowy programu i wykonuje go linia po linii.
- Jaka jest różnica pomiędzy kompilatorem, a interpreterem?
  Podczas kompilacji nie jest wykonywany kod źródłowy, jest on tylko tłumaczony na język maszynowy, który jest zapisywany z kolei do pliku i wykonywany później.
  Interpreter natomiast analizuje kod i od razu go wykonuje

Program

Program to wynik **kodu źródłowego**, który jest tłumaczony (przez inny program np. kompilator lub interpreter) na kod maszynowy zrozumiały przez komputer i gotowy do uruchomienia.

Program
gotowy do działania

01011100
010001010
01011100
010001010
01011100
01000101

Coders Lab
SZKOŁA PROGRAMOWANIA

Język programowania a język znaczników

Nie każdy język zrozumiały dla komputera jest językiem programowania.
Wyróżniamy jeszcze **języki znaczników**.

Są to języki:
- ➢ opisujące wygląd przechowywanej treści,
- ➢ nie da się w nich realizować obliczeń i kontrolować przepływu treści,
- ➢ najpopularniejszym językiem znaczników stosowanym obecnie jest **HTML** – który już trochę znasz ☺

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Podstawy programowania</title>
    <script src="js/app.js"></script>
  </head>
  <body>


  </body>
</html>
```

O języku JavaScript

JavaScript

- Powstał w **1995 roku**

  (Z ciekawostek: w tym roku była też denominacja złotego, oraz powstał album Gangsta's Paradise, amerykańskiego rapera Coolia.) ☺

- **JavaScript jest głównie używany do interakcji z użytkownikiem.**

- Działa zazwyczaj w przeglądarce internetowej razem z HTML i CSS.

- **Pamiętaj, że JavaScript to nie Java!**

- Początkowo JS miał służyć **tylko i wyłącznie** do interakcji z użytkownikiem.

- Od kilku lat, dzięki **Node.js**, JavaScript może być użyty wszędzie, nie tylko w przeglądarce. Pierwsza wersja Node.js powstała w 2009 roku. Od tej chwili JS w zasadzie może zostać uruchomiony na każdym komputerze lub urządzeniu (serwery, roboty, skafandry kosmiczne).

- Obowiązującą wersją jest wersja **ES6/2015** (trwają prace nad ES7), chociaż jeszcze nie wszystkie nowości zostały zaimplementowane w przeglądarkach.

- Istnieją języki przetwarzane na **JavaScript** (**CoffeeScript**, **TypeScript**).

Pierwszy skrypt

Pierwszy skrypt

- Do uruchomienia kodu JavaScript, będziesz potrzebować dwóch plików HTML i JS. Niech struktura plików wygląda tak:

- index.html

- script.js

Plik HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>Coders Lab</title>
        <script src='script.js'></script>
    </head>
    <body>

    </body>
</html>
```

Plik JavaScript

```
console.log('Hello World!');
```

Plik **index.html** wczytuje się jako pierwszy (podstawowy) i to on wywołuje uruchomienie skryptu JS.

* **console.log();** to polecenie wypisujące dane do konsoli przeglądarki

Konsola JavaScript

Ten kod, który napisaliśmy na poprzednim slajdzie ma za zadanie wypisać tekst w tzw. konsoli.

Jeśli uruchomisz plik index.html **nic się nie powinno pojawić.**
Musisz uruchomić specjalne narzędzie, które ma dostęp do tzw. **konsoli JavaScript**. Jest ono dostępne w prawie każdej przeglądarce.

Najpopularniejsze to jednak **Chrome Developer Tool** (jak sama nazwa wskazuje, dostępne w przeglądarce Chrome)

Chrome developer tools

Jak włączyć?

➢ **Na Windows:** *Crtl* + *Shift* + *I* lub *F12*

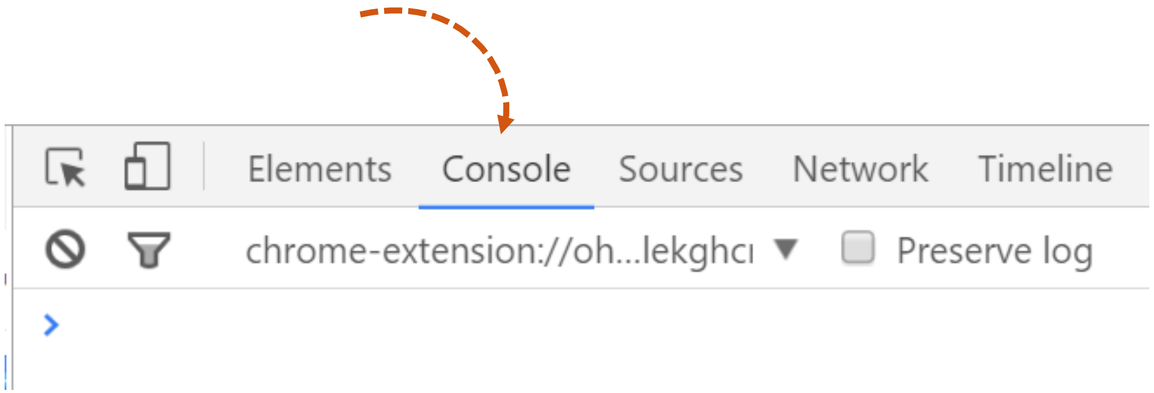
➢ **Na OS X:** *Cmd* + *Opt* + *I*

Pamiętaj, że aby zobaczyć tekst, który wpisałeś w pliku script.js, musisz:

a) dodać script.js do pliku index.html (tak jak w przykładzie na slajdzie „Pierwszy skrypt”) ,
b) otworzyć index.html (w Chrome)
c) włączyć narzędzie Chrome Developer Tool
d) Przejść do zakładki Console

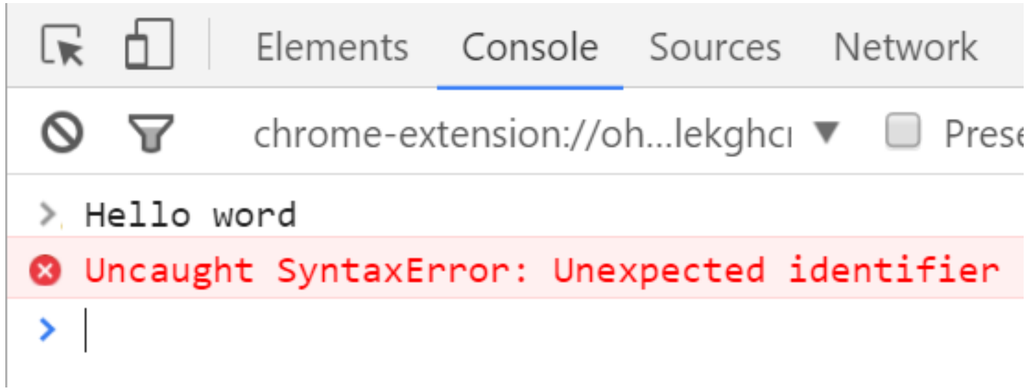
Więcej informacji o tym, jak używać tego narzędzia:
http://developer.chrome.com/devtools

Interesuję Cię ta zakładka: Console



Chrome developer tools

Do Konsoli możesz wpisywać również kod JavaScript. Jeśli wpiszesz coś nie tak, zostaniesz o tym szybko poinformowany za pomocą wypisania odpowiedniego błędu, przykład:



Nie martw się jeśli na razie ten błąd nie jest dla Ciebie jasny, w dalszej części wszystko się wyjaśni.

Komentarze i oddzielanie instrukcji

Komentarze

- W kodzie JavaScript można dodać tekst, który będzie ignorowany przez kompilator. Są to tak zwane **komentarze**.

- Komentarzy używamy zarówno do opisywania, co robi kod (dla innych programistów), jak i chwilowego wyłączania niepotrzebnych części skryptu.

```
// Komentarz

// Komentarz


/*

Między tymi znacznikami wszystko jest

zakomentowane.

*/
```

Oddzielanie instrukcji

- Instrukcja to nakazanie spełnienia jakiegoś zadania. W języku naturalnym każde zdanie (przekazanie informacji) kończymy kropką.
- Dlatego każda instrukcja powinna kończyć się **znakiem średnika** i nowej linii.
- Instrukcja może mieć **więcej niż jedną linię.**

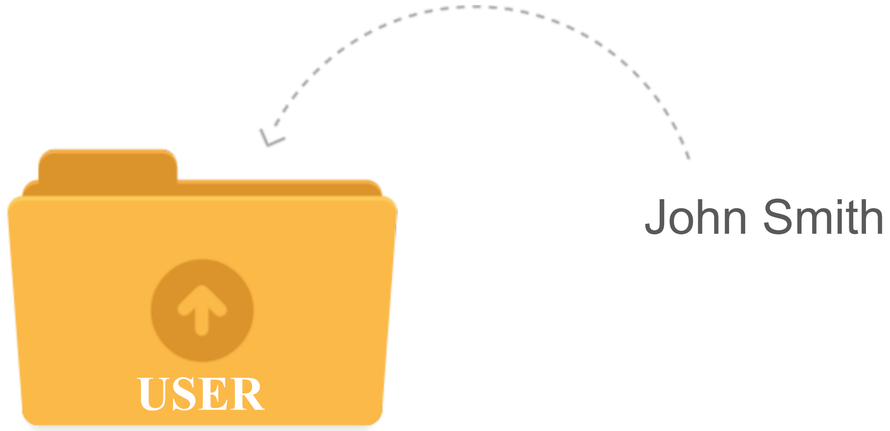

```
console.log('Hello')  //OK, ale nie jest najlepiej.

console.log('Hello'); // Jest najlepiej!

console.log(
'Hello');      // Również poprawnie!

console.log('Hello') console.log('Hello') // Źle
```

Wykonaj zadania z działu
Pierwszy skrypt.

Zmienne

Zmienne

- Zmienna w pewnym uproszczeniu, to etykieta odnosząca się do jakiegoś obszaru pamięci.
- Jest to tak naprawdę pudełko, które coś trzyma. Takie pudełko ma na sobie nalepkę ze swoją nazwą.
- Chcąc uzyskać dostęp do zawartości pudełka musimy wywołać je za pomocą jego nazwy.



var

- Jeśli chcemy korzystać ze zmiennych, należy dodać takie właśnie pudełka na zmienne. W JS robi się to przy pomocy słowa kluczowego **var**.
- Słowo **var** to skrót od **variable** czyli zmienna.
- Jeżeli definiujemy zmienną, powinniśmy zawsze pamiętać o słowie kluczowym **var.**
- Dzięki temu nasz kod jest czytelny oraz unikamy nieprzewidywalnych zachowań silnika javascriptowego (zmiennych globalnych).

Zmienne

Nazwa zmiennej

- Każda nazwa zmiennej musi być poprzedzona słowem kluczowym **var**.
- Składa się z dowolnej liczby liter, cyfr i niektórych znaków specjalnych (np. $ i _).
- Nazwa zmiennej musi zaczynać się od litery.
- Zmienne powinno się nazywać takimi słowami, które opisują, co ta zmienna będzie przechowywać (najlepiej po angielsku).
- Brak słowa kluczowego **var** sprawi, że utworzymy zmienną globalną, która może nadpisywać inne zmienne. O zasięgu zmiennych nauczysz się więcej na kursie.

Dwie różne zmienne

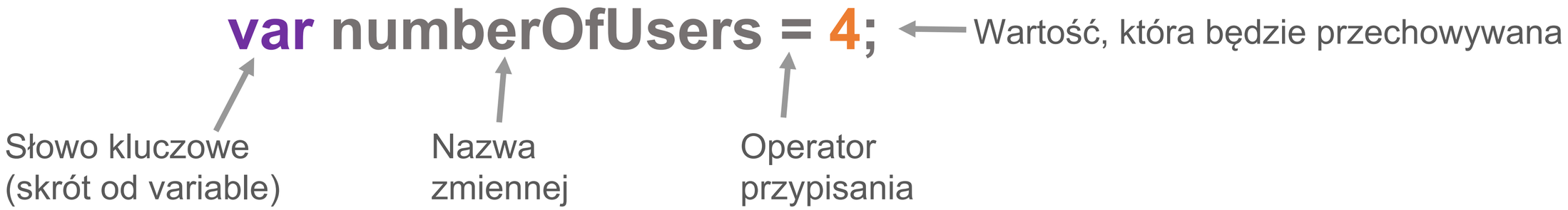
To są dwie różne zmienne.

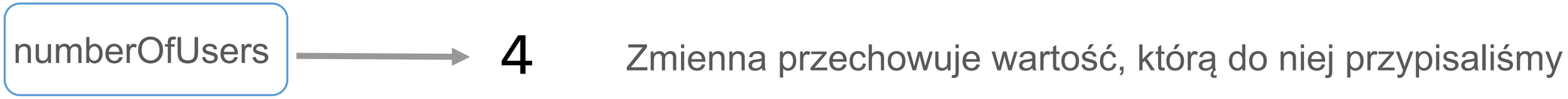
```
var test = 1;
var TEST = 10;
```

Uwaga! Zmienne są czułe na wielkość znaków!

Zmienne

Zmiennych używamy do przechowywania danych i zarządzania nimi.





Nadawanie nazw zmiennym

var number_Of_Users		OK
var numberOfUsers		OK – camelCase

Nadawanie nazw zmiennym

var number Of Users		Żadnych spacji
var 4Users		Żadnych liczb na początku
var użytkownik		Żadnych polskich liter
var xxx		Brak sensu

Typy danych

Typy danych

Liczby(Number)

```
var liczba = 10;
var liczba2 = 2.2;
```

Ciągi znaków (String)

```
var tekst = "Ala ma kota";
var tekst2 = "2.2";
```


Wartości logiczne (Boolean)

```
var prawda = true;
var falsz = false;
```

Specjalne

```
var foo = null;
var bar = undefined;
```

Prymitywne typy danych

Obiekty

```
var kot = {
    imie: "Mruczek",
    wiek: 3
}
```

Tablice

```
var tab1 = [1, 2, "Ala"];
var tab2 = [1, 2, 45 ];
```

Informacje o **tablicach** znajdziesz w dalszej części preworku.
Czym są **obiekty** nauczysz się w trakcie kursu.

Liczby

Liczby

Wartości liczbowe mają tylko jeden typ: 64-bitowy, zmiennoprzecinkowy.

```
var foo = 100;
var bar = 100.0;
var baz = 1e2;
```

} 100

```
100 === 100.0; // true
100 === 1e2; // true
100.0 === 1e2; // true
```

```
//false
0.1 + 0.2 == 0.3

//true
(0.1 + 0.2).toFixed(2) === (0.3).toFixed(2)
```

```
0.1 + 0.2 = 0.30000000000000004
// wynik nie  jest równy 0.3
```

* Dzięki metodzie **toFixed()** możemy poprawnie zaokrąglać ułamki dziesiętne. Przykładowo **toFixed(2)** zaokrągla do drugiego miejsca po przecinku.

Liczby

- **var result = 9;**
- **var length = 102.52;**
- **var numberOfboxes = 20;**

Na liczbach możemy wykonywać działania matematyczne.

+ - / * %

Modulo
(reszta z dzielenia)

Przykład:

```
var liczba = 3;
var liczba2 = 4;

var suma = liczba + liczba2; // 7

var modulo = suma % liczba; // 1
```

Stringi

Stringi

Stringi to ciągi znaków.

```
var text = "Ala ma kota";
var name = "John";
var text2 = "20";
```

Stringi możemy dodawać jest to tzw. **konkatenacja**, czyli łączenie łańcuchów znaków.

```
name + " Travolta";
/* otrzymamy wynik */
"John Travolta"
```

Stringi

$$2 \neq \text{"2"}$$

Zapamiętaj!

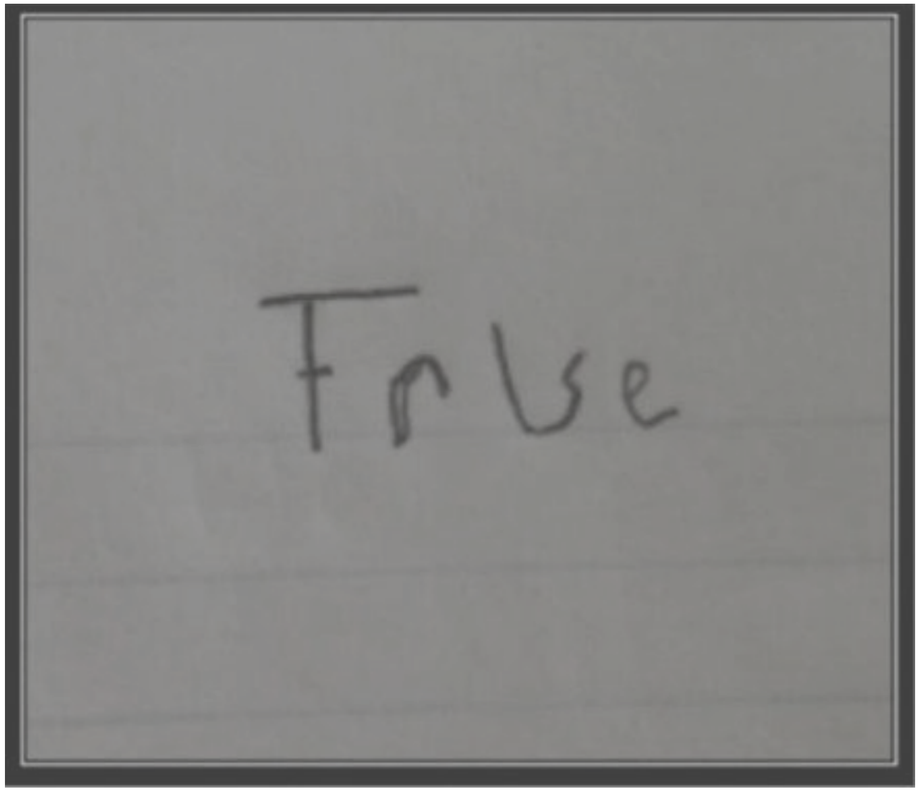
Stringi to nie to samo co liczby

Wartości logiczne

Wartości logiczne

Wartości logiczne (z ang. boolean) to wartości reprezentujące po prostu prawdę lub fałsz.

```
var isChecked = false;

var isRendered = true;
```

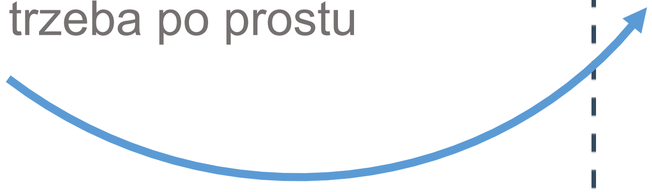


Wartości logiczne – falsy values

W JavaScripcie często występuje zjawisko koercji (konwertowania jednego typu danych na inny – dowiesz się o tym więcej na kursie).

W związku z tym w JS istnieją tzw. ***falsy values*** czyli wartości, które przy konwersji na wartość logiczną dadzą nam **fałsz**.

Wartości **falsy values** trzeba po prostu zapamiętać!



Oprócz logicznego fałszu (*false*), fałszem będą również:

- **0 (zero)**
- **"" (pusty string)**
- **null**
- **undefined**
- **NaN (Not a Number)**

Wartości specjalne

Wartości specjalne

null

null reprezentuje pustą wartość.

Uwaga! null nie oznacza 0 (zero).

```
var foo = null;
var bar = undefined;
```

undefined

undefined reprezentuje wartość niezdefiniowaną,

Jeżeli stworzymy zmienną i nic do niej nie przypiszemy/wsadzimy, wywołanie jej zwróci **undefined**.

```
var maxValue;
console.log(maxValue);
```

Konsola wypisze **undefined**

null i **undefined** praktycznie oznaczają to samo, różnica jest tylko semantyczna. **undefined** mówi nam, że jakiejś zmiennej nie przypisaliśmy żadnej wartości, natomiast **null** używamy, gdy chcemy przypisać wartość pustą.

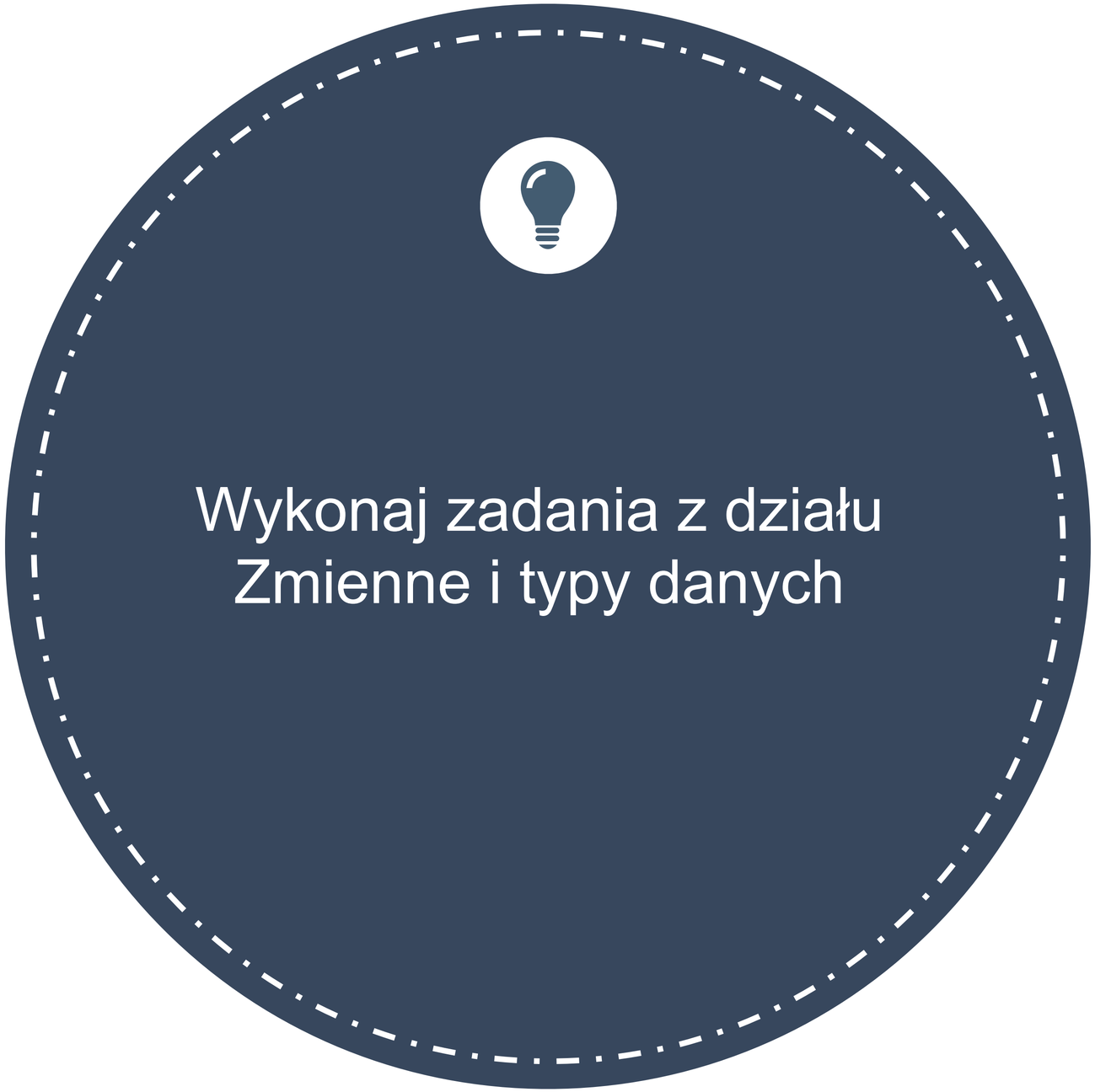
typeof()

Sprawdzanie typów danych

W JavaScripcie istnieje tzw. **operator typu**, dzięki któremu możesz **sprawdzić typ danej zmiennej**.

Uwaga!
Na etapie preworku zajmujemy się tylko prymitywnymi typami danych. Więcej o typach danych i koercji nauczysz się w trakcie kursu.

```
var liczba =  2;
var tekst = "u mnie działa";
var prawda = true;


typeof(liczba); // number
typeof(tekst); // string
typeof(prawda); // boolean
```



Wykonaj zadania z działu
Zmienne i typy danych

Operatory

Podstawowe operatory w JavaScripcie

W każdym języku programowania istnieją operatory (specjalne znaki służące do operowania na zmiennych).

Postaraj się w trakcie przerabiania tej prezentacji przyswoić operatory, które są pogrubione obok czyli arytmetyczne, przypisania oraz porównania.

Resztę omówimy w trakcie kursu ☺

Są one podzielone na grupy:

- **operatory arytmetyczne**,
- operatory logiczne (omówimy w trakcie kursu),
- **operatory przypisywania**,
- **operatory porównywania**,
- operatory działań na napisach (omówimy w trakcie kursu),

Operatory arytmetyczne

Operatory arytmetyczne służą do przeprowadzania operacji matematycznych na zmiennych.

Operator	Opis
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	reszta z dzielenia (modulo)

Operatory porównywania

Grupa operatorów służąca do porównywania zmiennych ze sobą. Zwracają zawsze **true** albo **false** (wartości **boolean**).

Operator	Opis	Przykład
==	równe sobie (ta sama wartość, ale mogą mieć różne typy)	2 == "2" // true
===	równe sobie i mające ten sam typ	4 === "4" // false
!=	nierówne	6 != 12 // true
!==	nierówne i/lub różnego typu	4 !== "4" // true
> >=	większe niż (i większe równe niż)	12 > 10 // true 12 >= 12 // true
< <=	mniejsze niż (i mniejsze równe niż)	10 < 12 // true 10 <= 10 // true

Operatory porównywania

Wiemy już, że w zmiennych możemy przechowywać różne typy danych, np. liczby albo stringi.
Jeśli chcemy porównać dwie wartości możemy użyć **" == "** , na przykład:

```
var numberOfUsers = 2;
var loggedUsers = "2";

numberOfUsers == loggedUsers;
```

Wyświetli nam true

Jeżeli natomiast chcielibyśmy sprawdzić również typ przechowywanej wartości w zmiennej to musimy użyć **" === "** , na przykład:

```
var numberOfUsers = 2;
var loggedUsers = "2";

numberOfUsers === loggedUsers;
```

Wyświetli nam false

Operatory przypisania

Czyli **operator =** , przypisuje wartość do zmiennej. Zobacz przykład obok - do zmiennej o nazwie ***liczba*** przypisaliśmy liczbę 1, a do zmiennej o nazwie ***tekst*** przypisaliśmy tekst „Ala ma kota”.

```
var liczba = 1;
var tekst = "Ala ma kota";
```

Możemy również zmieniać wartości w zmiennej, poprzez ponowne przypisanie.
Teraz już nie tworzymy zmiennej więc nie potrzebujemy słowa **var**. Zmienne są już stworzone, wstawiamy do nich tylko inne dane.

```
liczba = 298;
tekst = "Kot ma Alę";
```

Operatory przypisania

Inkrementacja i dekrementacja

W programowaniu bardzo często spotykamy się ze zwiększaniem lub zmniejszaniem wartości zmiennych o jeden.
Jest to tak bardzo często wykorzystywane, że przyjęły się specjalne nazwy, które określają te operacje:

Inkrementacja – zwiększanie zmiennej o jeden

```
var zmienna = 56;
zmienna = zmienna + 1;
```

Dekrementacja – zmniejszanie zmiennej o jeden

```
var zmienna2 = 56;
zmienna = zmienna - 1;
```

Inkrementacja i dekrementacja są tak popularne, że mają również swoje specjalne zapisy:

Inkrementacja:

```
var zmienna = 56;
zmienna++;  // to inny zapis: zmienna = zmienna + 1
console.log(zmienna); // wypisze wartość 57
```

Dekrementacja:

```
var zmienna = 56;
zmienna--;  //to inny zapis: zmienna = zmienna -1
console.log(zmienna); // wypisze wartość 55
```

Wykonaj zadania z działu
Operatory

Kontrola przepływu programu

O co chodzi z tą kontrolą przepływu?

Instrukcje warunkowe

Wyobraź sobie sytuację, w której piszesz program na podstawie danych, które wpisze użytkownik.

Na przykład **panel logowania.**

Użytkownik musi podać **login i hasło**, żeby móc się zalogować. Aplikacja, do której się loguje, musi **sprawdzić czy login pasuje do hasła**, jeśli nie to wypisze użytkownikowi informację, „Zły login lub hasło”, jeśli pasuje to użytkownik zostanie zalogowany.

Widzisz w powyższym przykładzie sytuacje, w której programista musi sprawdzić dane i na ich podstawie **zdecydować co zrobić dalej**.

Do tego typu sytuacji będą nam potrzebne instrukcje warunkowe.

Pętle

A teraz wyobraź sobie sytuację, w której musisz wybrać z całej bazy użytkowników, tego jednego, do którego pasują login i hasło. Żeby móc przejrzeć całą bazę będziesz potrzebować **pętli, czyli specjalnej konstrukcji, która wykonuje te same czynności dla podanej grupy danych.**

To tylko przykładowe sytuacja, która mają pomóc Ci zrozumieć sens używania instrukcji warunkowych i pętli.

Inne przykłady

Instrukcje warunkowe

- **Po zmianie szerokości okna:** Jeśli szerokość ekranu jest mniejsza niż 560px schowaj menu, w przeciwnym przypadku pokaż menu
- **Po kliknięciu:** Jeżeli element na stronie jest widoczny, schowaj go, w przeciwnym przypadku pokaż go
- **Po wejściu na stronę:** Jeżeli jest godzina pomiędzy 6:00 – 19:00 to wypisz użytkownikowi „dzień dobry” , w przeciwnym wypadku wypisz użytkownikowi „dobry wieczór”.

Pętle

- Dla każdego paragrafu na stronie, dodaj klasę o nazwie „red-border”.
- Dla wszystkich obrazków w galerii dodaj ramkę
- Dopóki licznik galerii jest mniejszy od liczby zdjęć, zwiększaj licznik o 1 i pokazuj kolejne zdjęcie
- Wypisz liczby od 1 do 10
- Wypisz sumę liczb od 1 do 100

Na kolejnych slajdach poznamy szczegóły
Jak tworzyć instrukcje warunkowe i pętli

Instrukcje warunkowe **if** oraz **switch**

Instrukcja warunkowa – if …else... else if

- Instrukcja warunkowa **if** pozwala na wykonanie kawałka kodu w zależności od warunku postawionego po słowie kluczowym **if**.

```
if (instrukcja warunkowa 1) {
    kod, który się wykona jeśli instrukcja warunkowa 1
    jest prawdziwa;
} else if (instrukcja warunkowa 2) {
    kod, który się wykona jeśli instrukcja warunkowa 2
    jest prawdziwa;
} else {
    kod, który się wykona jeśli instrukcja warunkowa 1
    i instrukcja warunkowa 2 są fałszywe;
}
```

Przykład w języku naturalnym

Jeśli pada deszcz, weź parasol, **w przeciwnym** przypadku **jeśli** pada śnieg weź czapkę, **jeśli żadne z powyższych nie zostało spełnione** weź okulary przeciwsłoneczne.

Przykład w języku JavaScript

```
var pogoda = "deszcz"; // tą wartość można zmienić

if (pogoda === "deszcz") {
    console.log("Weź parasol");
} else if (pogoda === "śnieg") {
    console.log("Weź czapkę");
} else {
    console.log("Weź okulary przeciwsłoneczne");
}
```

Instrukcja warunkowa – if …else… else if

Przy tworzeniu instrukcji warunkowej nie musisz tworzyć warunków innych niż **if**.

To znaczy, że **else if** oraz **else** są opcjonalne.

Możesz użyć instrukcji **if** w następujący sposób:

```
var isRaining = "deszcz"; // tą wartość można zmienić

if (isRaining === "deszcz") {
        console.log("Weź parasol");
}
```

Przykład w języku JavaScript

Możesz użyć instrukcji **if tylko z else** w następujący sposób:

```
var isRaining = "deszcz"; // tą wartość można zmienić

if (isRaining === "deszcz") {
        console.log("Weź parasol");
} else {
        console.log("Weź okulary przeciwsłoneczne");
}
```

Instrukcja warunkowa – switch

Oprócz instrukcji **if** mamy również do dyspozycji instrukcję **switch**. Używamy jej najczęściej wtedy kiedy mamy do wyboru więcej opcji niż dwie lub trzy. Zobacz konstrukcję switcha obok:

Po każdym warunku **case** pamiętaj o słowie kluczowym **break**, za pomocą którego możemy wyjść z instrukcji **switch**. Nie jest obowiązkowe, ale w przypadku jego braku każda instrukcja zostanie wykonana. Czyli w przypadku usunięcia słowa **break** z przykładu obok, jeśli padałby deszcz musielibyśmy wziąć parasol, czapkę i nie wychodzić z domu… bez sensu ☹

Słówko kluczowe **default** zostaje wykonane wtedy, gdy żaden warunek nie pasuje do **case.**

Przykład w języku JavaScript

```
var weather = "deszcz"; // tą wartość można zmienić

switch(weather) {
    case  "deszcz": {
        console.log("Weź parasol");
        break;
    }
    case  "śnieg": {
        console.log("Weź czapkę");
        break;
    }
    default : {
        console.log("Weź okulary przeciwsłoneczne");
    }
}
```

Instrukcja warunkowa – switch

Inny przykład instrukcji switch

```
var expression = "John";

switch (expression) {
      case "Ala":
            console.log("Jestem, wpadaj na kawę");
            break;
      case "John":
            console.log("Jestem, ale zaraz idę");
            break;
      default:
            console.log("Nie ma mnie w domu");
}
```

Program szuka etykiety **case** pasującej do **expression**. Jeżeli napotka **break**, to kończy działanie instrukcji **switch**.

Jeżeli żadna etykieta **case** nie pasuje do **expression**, zostanie wykonana domyślna akcja.

if… else if kontra switch

if… else if

Tworząc instrukcję warunkową możesz potrzebować dużo więcej opcji niż tylko trzy.

Pamiętaj, że opcji **else if** możesz użyć dowolną ilość razy np.

```
var weather = "deszcz"; // tą wartość można zmienić

if (weather === "deszcz") {
        console.log("Weź parasol");
} else if (weather === "śnieg") {
        console.log("Weź czapkę");
} else if (weather === "grad") {
        console.log("nie wychodź z domu");
} else if (weather === "mróz") {
        console.log("ubierz się grubo");
}
//…. I tak dalej ….
```

Lub użyć switch

```
var weather = "deszcz"; // tą wartość można zmienić

switch(weather) {
        case  "deszcz": {
                console.log("Weź parasol");
                break;
        }
        case  "śnieg": {
                console.log(" Weź czapkę");
                break;
        }
        case  "grad": {
                console.log("nie wychodź z domu ");
                break;
        }
}
```

Pętle:
for,
for zagnieżdżony
oraz while

Pętla - for

- Pętle pozwalają na wielokrotne wykonywanie danego kodu. Pętle z założenia działają w nieskończoność, chyba, że istnieje warunek kończący działanie pętli.

- Pętla, która nie ma warunku kończącego, nazywa się pętlą nieskończoną (**infinite loop**). Taka pętla potrafi zablokować silnik przeglądarki.

```
for (start, warunek kończący, skok) {
      kod wykonywany w każdej iteracji pętli
}
```

Pętla

```
for(var i=0; i<=10; i=i+1) {
    console.log(i);
}
```

Wynik: 0 1 2 3 4 5 6 7 8 9 10

Pętla

```
for(var i=10; i>=0; i=i-1) {
    console.log(i);
}
```

Wynik: 10 9 8 7 6 5 4 3 2 1 0

* wyniki (poszczególne liczby w konsoli) będą wyświetlane pod sobą, każdy w nowej linii

Pętla for

```
for (start od i = 0; dopóki i jest mniejsze niż 3; zwiększ i o 1) {
    console.log("Pętle są fajne");
}
```

W pętli for definiujemy zmienną, która będzie **licznikiem** pętli. W naszym przypadku jest to zmienna **i,** ale to może być dowolna nazwa np. counter, licznik, j. Przyjęło się jednak zwykle stosować **i**.

```
for (var i=0; i<3; i++) {
    console.log("Pętle są fajne");
}
```

Pętle są fajne
Pętle są fajne
Pętle są fajne

Pętla for

```
var result = 120;
for (var i=0; i<3; i++) {
    result = result+1;
}
```

Jeśli masz problem ze zrozumieniem pętli, stwórz sobie podobną tabelkę jak obok. W każdej kolumnie wypisz zmienne, które będą się zmieniać w kolejnych krokach. Możesz również umieścić **warunek stopu** – czyli wyjścia z pętli.

Krok	i=0	result=120	stop (i<3)
1	0	121	i<3
2	1	122	
3	2	123	

i jest równe **2**, a warunek stopu mówi, że musi być mniejsze od **3**, dlatego kolejna iteracja pętli jest już niemożliwa, a więc pętla kończy się.

Pętla while

Dopóki spełniony jest warunek, wykonuj pętlę.

```
var i = 0;
while (i != 5) {
    console.log("Pętle są fajne");
    i = Math.floor(Math.random() * 10);
}
```

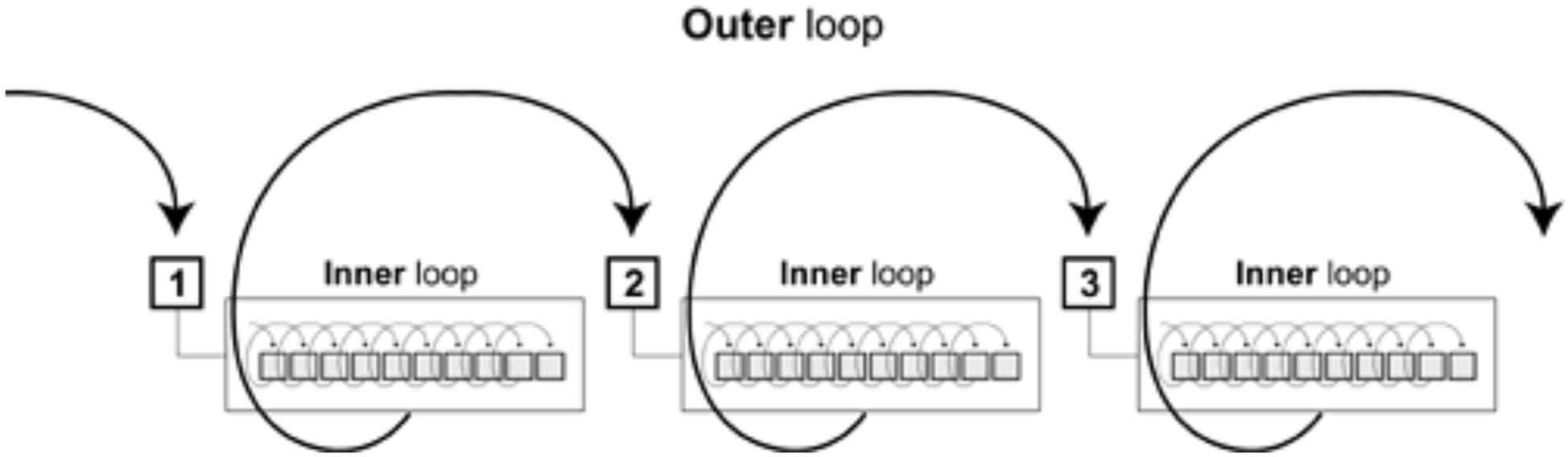
Pętlę **while** wykorzystujemy, jeżeli nie wiemy, ile razy będziemy wykonywać jakieś instrukcje.

Obiekt **Math** jest wbudowany w JavaScript, dzięki jego metodom możemy korzystać z funkcji matematycznych, tutaj **floor** to zaokrąglenie w dół a **random** losowa liczba z przedziału 0-1

Np.:
Math.random() zwróci 0.2315634
0.2315634 * 10 = 2.315634
Math.floor(2.315634) = 2

Pętla for (podwójna)

➢ Pętle możemy w sobie zagnieżdżać.
➢ Dzięki temu w każdej iteracji pętli zewnętrznej będzie wykonywane wiele iteracji pętli wewnętrznej.



Pętla for (podwójna)

Dobrze opanuj pętlę for pojedynczą

Aby zrozumieć pętle podwójną for musisz dobrze rozumieć pętle pojedynczą.

Przećwicz dobrze pojedynczą pętlę for.
Wykonaj zadania.

Jeśli potrzebujesz pomocy, to zajrzyj do sieci, poszukaj więcej materiałów, filmików, które w inny sposób tłumaczą pętle i pętle zagnieżdżone, np.:

- http://labs.codecademy.com/BJwV#:workspace
- https://www.khanacademy.org/computing/computer-programming/programming/looping/p/nested-for-loops

Przykład pętli podwójnej for

```
for(var i=0; i<10; i++) {
    for(var j=0; j<10; j++) {
        console.log("i=" + i + ", j=" + j);
    }
}
```

Dla każdej liczby od 0 do 9 wykonaj 10 razy pętlę wewnętrzną.

Pętla for (podwójna – niezależna)

W pętlach tworzymy dwie zmienne i oraz j. Są one niezależne od siebie. Zmienne te sterują pętlami. Mówimy wtedy o **pętlach niezależnych.**

```
for(var i=0; i<3; i++) {
    for(var j=0; j<4; j++) {
        console.log("i=" + i + ", j=" + j);
    }
}
```

Wynik w konsoli:

```
i= 0, j= 0
i= 0, j= 1
i= 0, j= 2
i= 0, j= 3
i= 1, j= 0
i= 1, j= 1
i= 1, j= 2
i= 1, j= 3
i= 2, j= 0
i= 2, j= 1
i= 2, j= 2
i= 2, j= 3
```

Pętla for (podwójna – zależna)

Możemy uzależnić zmienne wewnątrz pętli od siebie (np. przypisując numer iteracji pętli zewnętrznej jako start pętli wewnętrznej).

```
for(var i=0; i<4; i++) {
    for(var j=i; j<4; j++) {
        console.log("i=" + i + ", j=" + j);
    }
}
```

Wynik w konsoli:

```
i= 0, j= 0
i= 0, j= 1
i= 0, j= 2
i= 0, j= 3
i= 1, j= 1
i= 1, j= 2
i= 1, j= 3
i= 2, j= 2
i= 2, j= 3
i= 3, j= 3
```

Break

Wszystkie pętle możemy zakończyć przed warunkiem stopu za pomocą polecenia **break**.

W przykładzie po prawej pętla z założenia ma trzy iteracje (kiedy **i** jest równo 0, potem kiedy **i** jest równe 1 i ostatni raz kiedy **i** jest równe 2). W każdej iteracji do zmiennej ***result*** przypisujemy losową liczbę od 1 do 10, po czym sprawdzamy, czy wylosowano liczbę 5. Jeśli tak, to **przerywamy pętlę** – nie wykona się już ona ani razu, niezależnie od tego, w której iteracji wylosowaliśmy 5.

```
for (var i=0; i<3; i++) {
        var result = Math.floor(Math.random() * 10);
        if (result === 5) {
                break;
        }
}
```

Wykonaj zadania z działu
Kontrola przepływu programu

Tablice

Tablice

Czym są tablicę?

Jeśli chcielibyśmy mieć na przykład 10 liczb w jednej zmiennej, to jak to zrobić?

Możemy użyć tablicy.

Tablica to taki zbiór różnych wartości. Tablicę deklarujemy jak zwykłą zmienną, ale wstawiamy do niej specjalną konstrukcję, np.:

```
var  myNumbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

Możemy również stworzyć **pustą tablicę** np.

```
var  emptyArray = [ ];
```

Poniżej konstrukcja tablicy. Zauważ kilka rzeczy:

- wartości umieszczamy w nawiasach kwadratowych

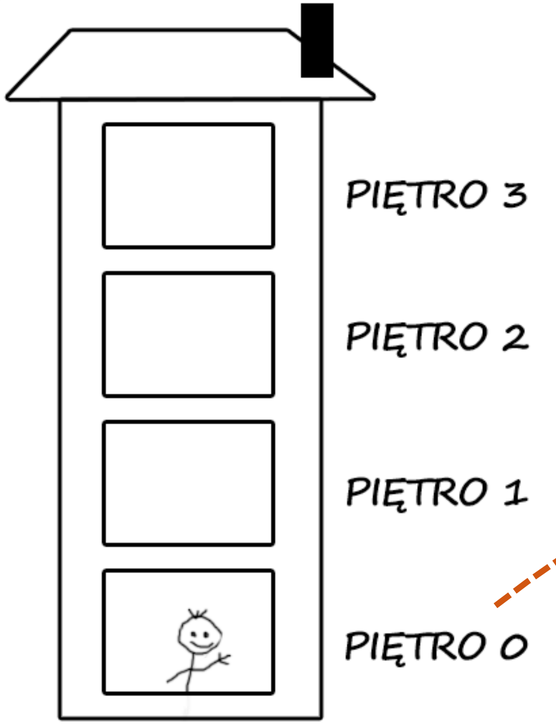
- każdą wartość oddzielamy przecinkiem

- w tablicy możemy umieszczać różne typy danych, nawet inne tablice ☺

[1, 2, 3, "ala"]

Tablice

Tablice można porównać do 1-klatokowego bloku, gdzie na każdym piętrze mieszka jeden lokator. Jak dowiedzieć się, kto mieszka w tym bloku na drugim piętrze?




```
var users =  ["Ala", "John", "Naomi", "Adam"];

var prices =  [23, 12, 9.40];
```

Tablice numerujemy od 0! To bardzo ważne! To znaczy, że pierwszy element tablicy ma indeks 0.

Aby pobrać element tablicy, musimy podać jej nazwę oraz **indeks**.

```
console.log(users[0]);  // wypisze: "Ala"
console.log(users[1]);  // wypisze: "John”
console.log(users[3]);  // wypisze: ”Adam"
console.log(users[4]);  // wypisze:  undefined
//ponieważ nie ma takiego elementu w tablic users
```

Tablice

Typy danych

Tablice mogą przechowywać różne typy danych:

- liczby,
- stringi,
- typy specjalne,
- wartości logiczne - boolean,
- inne tablice,
- itd

```
var mixTypes =  ["Ala",  23, true ];
```

Indeksy (klucze) tablic rozpoczynają się od 0.

```
console.log(mixTypes[0]);  // wypisze "Ala"
console.log(mixTypes[1]);  // wypisze 23
console.log(mixTypes[2]);  // wypisze true
```

Aby pobrać wielkość tablicy, korzystamy z atrybutu **length**:

```
console.log(mixTypes.length);   // 3
```

Tablice i pętle

Tablice i pętle

Jeżeli chcielibyśmy w prosty sposób wypisać wszystkie elementy tablicy w konsoli, możemy użyć pętli. Zobacz przykład obok.

Zauważ, że zmienną **i** ustawiliśmy na 0 – ponieważ pierwszy element tablicy ma indeks 0.

Wykonujemy pętlę do momentu kiedy **i < 3,** to znaczy, że nie może być równe 3, wtedy pobralibyśmy wartość **undefined**

```
var numbers = [120, 3, 45];
for (var i=0; i < numbers.length; i++) {
   console.log( numbers[i] );
}
```

```
120
3
45
```

Wykonaj zadania z działu
Tablice

Dobre praktyki

Używaj średnika

Kończcie każdą linię znakiem średnika

- Bardzo ważna zasada – każda pojedyncza instrukcja (komenda) powinna być zakończona średnikiem.
- Na końcach bloków { } nie robimy średników. Wyjątkiem są obiekty (o nich porozmawiamy na kursie).
- Czyli na końcach pętli i instrukcji warunkowych nie dajemy średników.
- Dzięki temu kod jest czytelniejszy i pomaga nam to unikać różnych błędów.

Gdzie średnik, a gdzie nie?

```
var  i;       // deklaracja zmiennej
i = 5;        // przypisanie do zmiennej wartości
i = i + 1;    // przypisanie do zmiennej wartości
i++;          // przypisanie do zmiennej wartości
var x = 9; //  deklaracja i przypisanie


// Nie potrzebujemy średników po }:
if  (...) {...} else {...}
for (...) {...}
while (...) {...}
switch() {}
```

Używaj wcięć

- Bardzo ważna zasada – każdy blok kodu powinien dodawać dwie lub cztery spacje.
- Dzięki temu kod jest czytelniejszy i od razu widać, gdzie dany blok się kończy.
- Zauważ, że zawsze to co jest w bloku kodu np. **if** lub **for** jest wcięte, dzięki temu widzimy, gdzie kończy się nasz blok i co się w nim znajduje.

```
var numbers = [120, 3, 45];
for (var i=0; i < numbers.length; i++) {
   console.log(numbers[i] );
}
```

Używaj camelCase

- Nazwy zmiennych zaczynamy od małej litery i używamy **camelCase**.
- Nazwy funkcji zaczynamy od małej litery i używamy **camelCase**. O funkcjach porozmawiamy na kursie ☺

```
var myFavoriteNumber = 76;


function registerNewUser(x) {
    //ciało funkcji
};
```

Konwencje i dobre praktyki w JS:
http://javascript.crockford.com/code.html

Logicznie nazywajmy funkcje i zmienne

Wystrzegaj się zmiennych o nazwie test, bla, costam i innych tego typu.

Pamiętaj też, aby raczej używać angielskich nazw.

Błędne nazwy:

~~var test = 76~~

~~var bla = "nic nic nic"~~

~~var costam = true~~

Przykłady poprawnych nazw:

```
var numberOfUser = 102;

var successText = "Poprawne zalogowanie!"

var isActive = true;


function getNumberOfUsers(x) {
    // ciało funkcji
}
```

Praktyki ogólne

DRY – don't repeat yourself (nie powtarzaj się)!

Praktyka polegająca na nietworzeniu dwa razu tej samej funkcjonalności.

Unikamy więc kopiowania tych samych (lub bardzo podobnych) fragmentów kodu w wielu miejscach.

KISS – keep it simple, stupid!

Dobra zasada, która zamyka się w dwóch słowach: nie komplikuj!

Reguła ta jest często wspominana przy dyskusji architektury lub szczegółów budowy projektów. Jej istotą jest dążenie do utrzymania eleganckiej i przejrzystej struktury, bez dodawania niepotrzebnych elementów.

Oryginalny przekaz nie ma oddźwięku negatywnego i raczej przekazuje, aby utrzymywać rzeczy/sprawy w sposób prosty i zrozumiały dla każdego, nawet osoby średnio zorientowanej.

Nie kopiuj kodu,
którego nie rozumiesz!

Najczęstsze błędy początkujących

Najczęstsze błędy

Łatwo widoczne (błędy interpretatora)

- literówki,
- brak klamer, nawiasów, cudzysłowów zamykających blok.

Mniej widoczne (błędy logiczne)

- użycie złej zmiennej,
- niezainicjowanie zmiennej,
- porównanie różnych typów danych.

Linki przydatne każdemu programiście

W3Schools

Znajdziecie tu podstawowe tutoriale dotyczące wszystkich języków związanych z programowaniem aplikacji internetowych:

- HTML + CSS,
- JavaScript i jQuery,
- PHP.

http://www.w3schools.com

Dokumentacja JavaScript

Podstawowe miejsce, w którym dowiesz się, do czego służy funkcja.

Autorzy dokumentacji wyjaśniają podstawy używania języka.

Zanim użyjemy nowej funkcji, warto tu zajrzeć!

http://developer.mozilla.org/pl/docs/Web/JavaScript

Stackoverflow

Zawiera odpowiedzi na ogromną liczbę pytań.
Bardzo przydatne zarówno początkującym,
jak i zaawansowanym.

Zachęcamy do zaglądania – rozstrzał zagadnień jest tak duży, że każdy znajdzie coś dla siebie do poczytania!

http://stackoverflow.com