

PLAN DZIAŁANIA

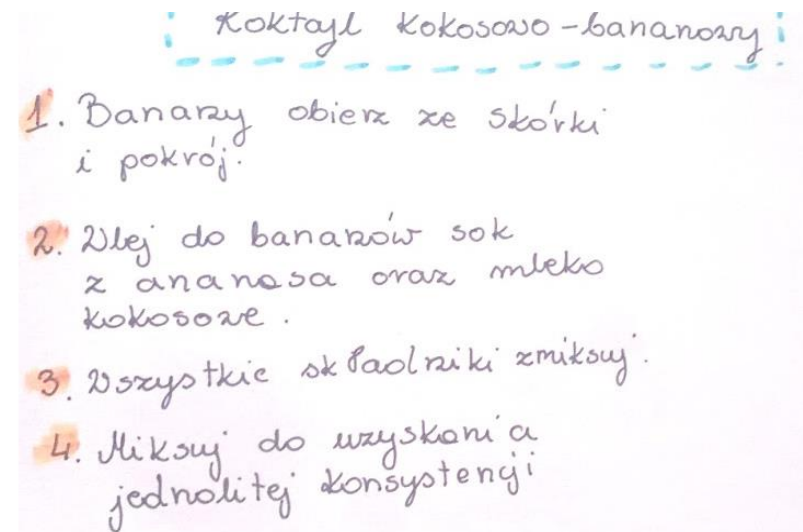
1. Chwila teorii
2. Wstęp do JS
3. Pierwszy skrypt
4. Komentarze i oddzielanie instrukcji
5. Zmienne
6. Typy danych
7. Operatory
8. Kontrola przepływu programu
9. Tablice
10. Dobre praktyki
11. Najczęstsze błędy początkujących

Chwila teorii – 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.



Chwila teorii – pseudokod

- 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.

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

Chwila teorii – 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.



Chwila teorii – 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.



Chwila teorii – 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

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Podstawy programowania</title>
    <link rel="stylesheet" href="js/app.js">
  </head>
  <body>
    ...
  </body>
</html>
```

Wstęp do JS

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

Pierwszy skrypt

Plik index.html

```
<!DOCTYPE html>
<html>

  <head>
    <title>Pierwszy skrypt - plik index.html</title>
    <script src='script.js'></script>
  </head>

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

Plik script.js

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


Komentarze i oddzielanie instrukcji

- 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 wyłączenia niepotrzebnych części skryptu.

```
/*  
Między tymi znacznikami wszystko jest zakomentowane.  
*/  
// Komentarz  
// Komentarz
```

- 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')  
console.log('Hello');  
console.log(  
  'Hello');  
console.log('Hello') console.log('Hello')
```



Czas na zadania

Zmienne

Zmiennych używamy do przechowywania danych i zarządzania nimi.

```
var numberOfUsers = 4;
```

var/let - Słowo kluczowe



numberOfUsers - nazwa zmiennej





= - operator przypisania

4 - Wartość, która będzie przechowywana

numberOfUsers -----> **4** - Zmienna przechowuje wartość, którą do niej przypisaliśmy

Zmienne – nazewnictwo!

var number_Of_Users		OK
var numberOfUsers		OK – camelCase

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

Typy danych

Liczby(Number)

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

Ciągi znaków (String)

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

31

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];
```

Typy danych – sprawdzanie typów

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

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



Czas na zadania

Operatory

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

Są one podzielone na grupy:

- **operatory arytmetyczne,**
- **operatory porównywania,**
- **operatory logiczne**
- **operatory przypisywania,**
- **operatory działań na napisach**

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
< <=	mniej niż (i mniej równe niż)	10 < 12 // true 10 <= 10 // true

Operatory – przypisania

```
var liczba = 1;  
var tekst = "Ala ma kota";  
liczba = 298;  
tekst = "Kot ma Alę";
```

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 = zmienna2 - 1;
```



Czas na zadania

Kontrola przepływu programu – WTF?

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 sytuację, w której programista musi sprawdzić dane i na ich podstawie zdecydować co zrobić dalej. Do tego typu sytuacji będą nam potrzebne instrukcje warunkowe.

Kontrola przepływu programu – WTF?

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 przykładowa sytuacja, która ma pomóc Ci zrozumieć sens używania instrukcji warunkowych i pętli.

Instrukcje warunkowe – 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; */  
}
```

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

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

Instrukcje warunkowe – switch

Oprócz instrukcji if mamy również do dyspozycji instrukcję switch. Używamy jej najczęściej wtedy kiedy mamy do wyboru więcej opcji niż 5.

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

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 okulary.

Pętle – 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 */  
}
```

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

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

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

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

Pętla – while

Dopóki spełniony jest warunek, wykonuj pętlę.

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

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

Pętle – zagnieżdżanie – niezależne

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
```

Pętle – zagnieżdżanie – zależne

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
```

Pętle – break

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

W przykładzie poniżej pętla z założenia ma trzy iteracje (kiedy i jest równe 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;  
    }  
}
```



Czas na zadania

Tablice

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];  
  
var emptyArray = [];
```

- wartości umieszczamy w nawiasach kwadratowych,
- każdą wartość oddzielamy przecinkiem,
- w tablicy możemy umieszczać różne typy danych, nawet inne tablice.

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

Tablice

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

```
var users = ["Ala", "John", "Naomi", "Adam"];  
var prices = [23, 12, 9.40];  
  
console.log(users[0]);  
console.log(users[1]);  
console.log(users[3]);  
console.log(users[4]);
```

Aby pobrać wielkość tablicy, korzystamy z atrybutu length:

```
console.log(prices.length);
```


Tablice i pętle

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

Zobacz przykład poniżej.

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]);  
}
```



Czas na zadania

Dobre praktyki - średniki

1. 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
- Czyli na końcach pętli i instrukcji warunkowych nie dajemy średników.
- Dzięki temu kod jest czytelniejszy i pomaga nam to unikać różnych błędów.

```
// deklaracja zmiennej
var i;
// przypisanie do zmiennej wartości
i = 5;
// przypisanie do zmiennej wartości
i = i + 1;
// przypisanie do zmiennej wartości
i++;
// deklaracja i przypisanie
var x = 9;
// Nie potrzebujemy średników po "}":
if (...) {...} else {...}
for (...) {...}
while (...) {...}
switch () {}
```

Dobre praktyki - wcięcia

- Bardzo ważna zasada – każdy blok kodu powinien dodawać 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]);  
}
```

Dobre praktyki - camelCase

- Nazwy zmiennych zaczynamy od małej litery i używamy camelCase.
- Nazwy funkcji zaczynamy od małej litery i używamy camelCase.

```
var myFavoriteNumber = 76;  
  
function registerNewUser(x) {  
    //ciało funkcji  
};
```

Dobre praktyki - nazewnictwo

- 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;

// poprawne nazwy
var numberOfUser = 102;
var successText = "Poprawne zalogowanie!";
var isActive = true;

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

Dobre praktyki - ogólnie

1. DRY – don't repeat yourself (nie powtarzaj się)!
2. KISS – keep it simple, stupid! -> nie komplikuj!

Najczęstsze błędy początkujących

- literówki,
- brak klamer, nawiasów, cudzysłowów zamykających blok.
- użycie złej zmiennej,
- niezainicjowanie zmiennej,
- porównanie różnych typów danych.