POLITECHNIKA GDAŃSKA

Aplikacje i usługi internetowe

Podstawy Programowania

Iwona Kochańska

Statyczne strony

- **Statyczny HTML**
  - Serwowanie tej samej treści wszystkim użytkownikom przy każdym żądaniu danego zasobu
  - Zmiana treści wymaga modyfikacji zawartości pliku po stronie serwera
- **Style CSS**
  - Statyczne reguły stylów dla elementów dokumentów HTML
  - Style uzależnione od akcji użytkownika na stronie
  - Brak możliwości modyfikowania struktury dokumentu HTML

Dynamiczne strony

- **Dynamika po stronie klienta** – przeglądarki
  - Modyfikowanie treści, struktury, wyglądu dokumentu HTML już po jego załadowaniu w przeglądarce - bez komunikacji z serwerem,
  - JavaScript.
- **Dynamika po stronie serwera**
  - Generowanie dokumentu HTML przez aplikację działającą po stronie serwera,
  - Serwowanie różnych treści, np. w zależności od użytkownika, dnia tygodnia, etc.,
  - PHP, Ruby on Rails, Python, ASP.NET MVC, Java, ...

Język JavaScript

- **JavaScript** to język programowania, który umożliwia wzbogacenie kodu HTML o:
  - interaktywność,
  - animacje,
  - dynamiczne efekty wizualne.
- Pierwsza wersja – rok 1995, przeglądarka Netscape Navigator 2.0
- Pierwotna nazwa: **LiveScript**
- **JavaScript** – nazwa marketingowa (przyciągnięcie uwagi deweloperów na fali szybko rosnącej popularności języka Java)
- **JavaScript i Java to dwa odrębne, niezależne języki programowania!**

Język JavaScript

- **JScript** – Internet Explorer 3.0, 1996 rok
  - bazował na języku JavaScript i oferował własne rozszerzenia,
  - był niekompatybilny z językiem JavaScript (różnice w API)
- **ECMAScript** – specyfikacja (standard) języka skryptowego (Ecma International, http://www.ecmascript.org/)
  - prace rozpoczęte po koniec 1996 roku
  - definiuje jednolity standard unifikujący języki JavaScript i JScript
  - kompatybilność między przeglądarkami

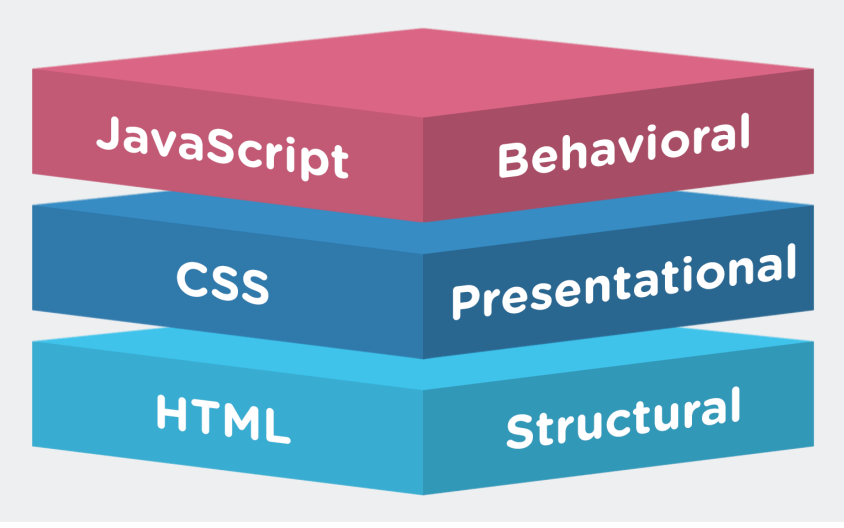
Język JavaScript

- Wszystkie popularne przeglądarki interpretują JavaScript
- Wszystkie implementacje bazują na standardzie ECMAScript
- Różne implementacje dostarczają własne rozszerzenia, np. dodatkowe API, metody rozszerzające istniejące API
- Problemy z kompatybilnością w różnych przeglądarkach!

Język JavaScript - typowe zastosowania

- Modyfikowanie struktury i treści dokumentu po załadowaniu strony (API DOM),
- Reagowanie na działania użytkownika w przeglądarce np. kliknięcie przycisku, wskazanie elementu kursorem,
- Walidacja danych w formularzach bez wysyłania żądań do serwera,
- Wyświetlanie reklam np. Google AdWords, Amazon Advertising,
- Statystyki odwiedzin np. Google Analytics.

Podstawowa struktura strony HTML



zrodło: http://blog.teamtreehouse.com

- **HTML** - organizacja treści
- **CSS** - prezentacja
- **JavaScript** - interakcja z użytkwonikiem

Gdzie umieścić kod JavaScript?

HTML 4

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script type="text/javascript">
    ...
    </script>
  </head>
<body>
  Content
</body>
</html>
```

HTML 5

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
    ...
    </script>
  </head>
<body>
  Content
</body>
</html>
```

Gdzie umieścić kod JavaScript?

Kod JavaScript w kodzie HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
      document.write("<p> Hello
          World !</p>");
    </script>
  </head>
<body>
  Content
</body>
</html>
```

Kod JavaScript w osobnym pliku (helloWorld.js)

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src = "helloWorld.js">
    </script>
  </head>
<body>
  Content
</body>
</html>
```

Literatura I

- Specyfikacja języka ECMAScript http://www.ecma-international.org/publications/standards/Ecma-262.htm
- JavaScript Tutorial na http://www.w3schools.com/js/
- David Sawyer McFarland, JavaScript i jQuery. Nieoficjalny podręcznik. Wydanie III, Helion, 2013

Instrukcje

- **Instrukcja** to podstawowa jednostka języka programowania
- Instrukcje mogą być:
  - uniwersalne (np. *isNaN()*)
  - specyficzne dla przeglądarek (np. *alert(), document.write()*)
- Każda instrukcja JavaScript kończy się **średnikiem**
- Instrukcja może zajmować kilka linii kodu, np:

```
var text = 'Adam ma ' +
      5 + ' psów ' +
      'i ' + 2 + ' koty';
```

Zmienne

- Zmienne są to konstrukcje programistyczne pozwalające na przechowywanie danych.
- Każda zmienna posiada:
  - **nazwę**, dzięki której można się do niej odwoływać w kodzie skryptu,
  - **typ**, który określa, jakiego rodzaju dane może przechowywać zmienna.
- Nazwy zmiennych powinny powinny być jasne i znaczące!
- Przykłady zmiennych:

```
var indeks;
var name = "Thomas Edison";
var liczba = 42, licznik, liczba_stron = 10;
var tablica = ["Ala", "Zosia", "Ania"];
```

Nazwy zmiennych

- **Nazwa** jest nadawana zmiennej przez programistę
- Może być dowolna, jednak musi spełniać następujące zasady:
  - musi zaczynać się od litery, znaku **$** lub _
  - może zawierać jedynie **litery**, **cyfry**, znak **$** i _
  - może zawierać znaki narodowe (czyli np. polskich liter ą, ź czy ć)
- W nazwach wolno stosować zarówno **duże**, jak i **małe** litery, ale są one **rozróżniane**
- Znak **$** - mimo że jest dopuszczalny, lepiej go nie używać (korzystają z niego często narzędzia do automatycznego generowania i optymalizacji kodu)
- Należy unikać **słów kluczowych** jako nazw zmiennych

Słowa kluczowe języka JavaScript

break	else	instanceof	true
case	false	new	try
catch	finally	null	typeof
continue	for	return	var
default	function	switch	void
delete	if	this	while
do	in	throw	with

Słowa zarezerwowane dla przyszłych wersji ECMAScript

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

Słowa zarezerwowane na potrzeby przeglądarek

alert	history	open	screenY
blur	innerHeight	outerHeight	statusbar
closed	innerWidth	outerWidth	window
document	length	parent	
focus	location	screen	
frames	navigator	screenX	

Tworzenie zmiennych

- Deklaracja zmiennej - za pomocą słowa kluczowego **var**

```
var typ_zmiennej nazwa_zmiennej = wartosc;
```

Przykład:

```
var zmienna1;
var zmienna2 = 100;
```

- Deklaracja zmiennej bez słowa kluczowego **var** - zmienna jest globalna**:**

```
zmienna = 100;
```

Takiej deklaracji unikamy!

Tworzenie zmiennych

- Deklaracja kilku zmiennych w jednej instrukcji:

```
var waga = 40, wiek , imie = 'Alicja';
```

lub:

```
var waga = 40,
  wiek,
  imie = 'Alicja';
```

- Po utworzeniu zmiennej można zapisać w niej dane dowolnego typu

```
var wiek;
wiek = 10;
```

- Wartość zapisaną w zmiennej można pobrać za pomocą nazwy tej zmiennej

```
console.log(wiek);
```

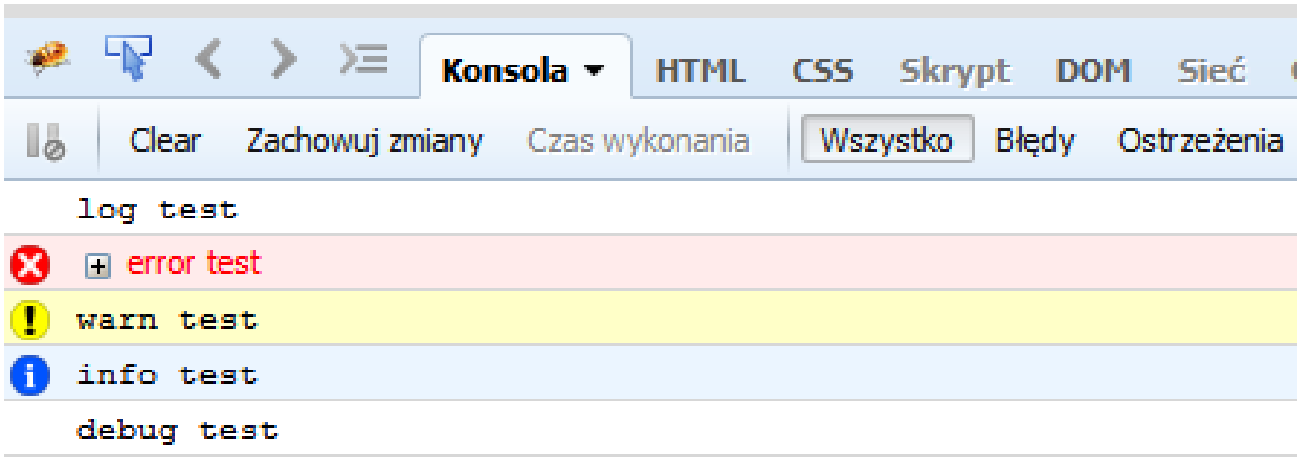
Funkcja console.log()

- Obiekt console nie jest standardowym obiektem języka JavaScript, jest jednak obsługiwany przez konsole w różnych przeglądarkach
- console.log() powoduje wyświetlenie wiadomości w konsoli
- Umożliwia logowanie wiadomości na różnych poziomach:

```
console.log("log test");
console.error("error test");
console.warn("warn test");
console.info("info test");
console.debug("debug test");
```

Konsola debuggera

Przeglądarka Mozilla Firefox: klawisz **F12**



Wypisanie wartości zmiennych

- Na stronie HTML:

```
var text = "Ala ma kota";
document.write(text);
```

- Wraz ze znacznikami HTML:

```
var text = "Ala ma kota";
document.write("<p style='font-weight: bold'>");
document.write(text + "</p>");
```

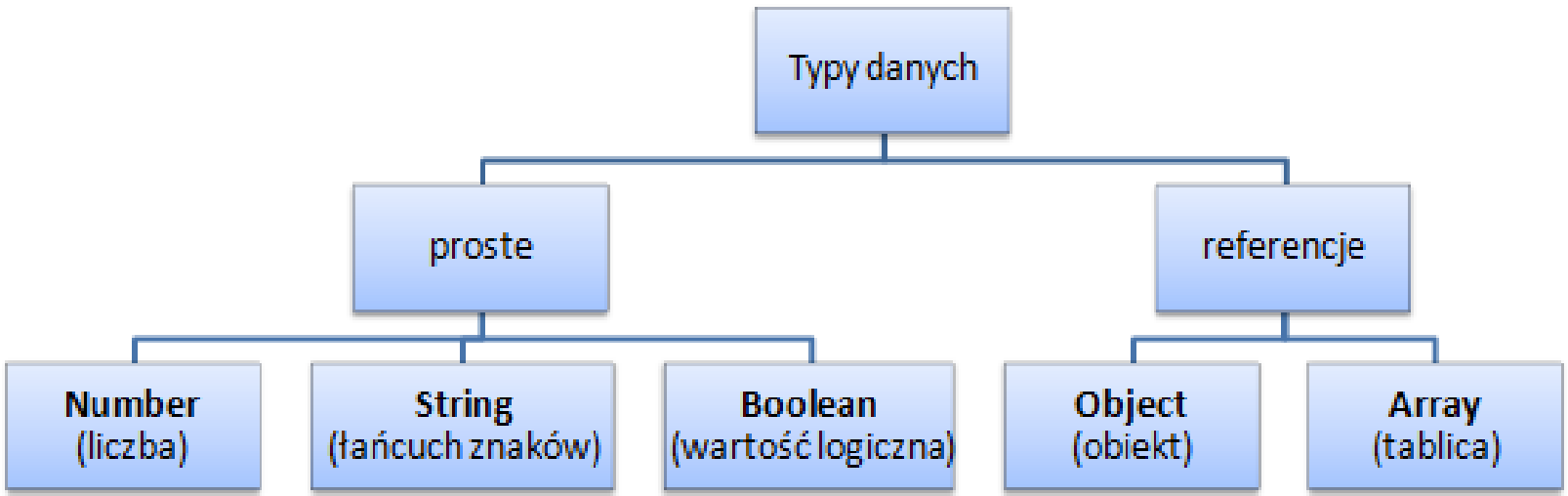
Wypisanie wartości zmiennych

- Wewnątrz elementu strony HTML:

```
<!DOCTYPE html>
<html>
  <body>
    <p id="komunikat"></p>
    <script>
      var text = "Ala ma kota";
      document.getElementById("komunikat").innerHTML =
          text;
    </script>
  </body>
</html>
```

Typy danych

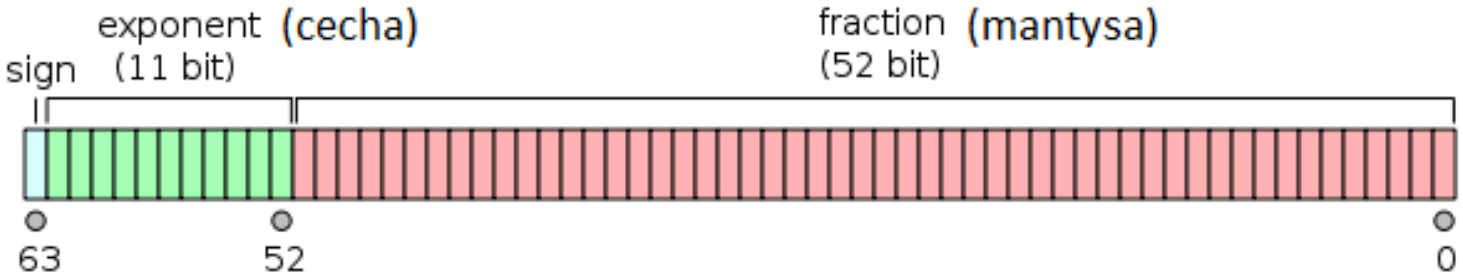
- W językach programowania przetwarzanie danych przebiega w odmienny sposób w zależności od ich **typu**
- Typy danych w języku JavaScript:



+ dwa typy/wartości specjalne: **Null** (jedna wartość **null**) i **Undefined** (jedna wartość **undefined**).

Typ Number

- Liczby w języku JavaScript to zawsze **64-bitowe liczby zmiennoprzecinkowe** w formacie **binary64** (IEEE 754 double-precision binary floating-point format)



źródło: https://en.wikipedia.org/wiki/Double-precision_floating-point_format

- Wartość liczby zmiennoprzecinkowej jest obliczana według wzoru:

$$x = SMB^E$$

gdzie:
S - znak liczby (1 lub -1),
M - mantysa (znormalizowana liczba ułamkowa)
B - podstawa systemu liczbowego (2 dla systemów komputerowych)
E - cecha (wykładnik)

Typ Number

- Liczby w wyrażeniach mogą być zapisane w postaci **całkowitej** lub **rzeczywistej**. Możliwy jest także zapis tzw. **naukowy**.

```
var a = 10;
var b = 3.14;
var c = 1.234e33;
```

- **Precyzja** liczb **całkowitych** to 15 cyfr:

```
var a = 1e15-1;          // a = 999999999999999
var b = 1e16-1;          // b = 10000000000000000
```

- Maksymalna **precyzja** liczb **rzeczywistych** to 17 miejsc dziesiętnych

```
var a = 0.3 + 0.6;          // a = 0.8999999999999999
var b = (0.3 * 10 + 0.6 * 10)/10;// b = 0.9
```

Typ Number

- Liczba interpretowana jest jako zapisana w systemie **szesnastkowym**, jeśli rozpoczyna się ona od znaków **0x**

```
var x = 0xAB;       // x = 171
```

- Reprezentacja w różnych systemach pozycyjnych za pomocą metody **toString():**

```
var a = 171;
a.toString(16);     // a = AB
a.toString(8);      // a = 253
a.toString(2);      // a = 10101011
```

Typ Number

- **Infinity** - nieskończoność. Wartość, jaką zwróci JavaScript w przypadku przekroczenia zakresu liczby

```
var a =  1 / 0;          //  Infinity
var b = -2 / 0;          //  -Infinity
```

- **NaN** (Not a Number)

```
var a = 100 / "liczba";  // NaN (Not a Number)
var a = 100 / "10";     // 10
```

Typ Number - metody

- **toString()**

```
var x = 326;
x.toString();              //  "326"
(326).toString();          //  "326"
(300 + 26).toString();     //  "326"
```

- **toExponential()**

```
var x = 5.656;
x.toExponential(2);        // 5.66e+0
x.toExponential(4);        // 5.6560e+0
```

Typ Number - metody

- **toFixed()**

```
var x = 5.656;
x.toFixed(0);          // 6
x.toFixed(2);          // 5.66
```

- **toPrecision()**

```
var x = 3.2656;
x.toPrecision(2);      // 3.3
x.toPrecision(4);      // 3.266
```

Konwersja zmiennych do liczb

- **Number()**

```
Number(true);                 //   1
Number("10");                 //   10
Number("10 20");              //  NaN
```

- **ParseInt()**

```
parseInt("10");               // 10
parseInt("10.33");            // 10
parseInt("10 20 30");         // 10
```

- **ParseFloat()**

```
parseFloat("10");             // 10
parseFloat("10.33");          // 10.33
```

Własności typu Number

MAX_VALUE	największa możliwa wartość liczby w JS
MIN_VALUE	najmniejsza możliwa wartość liczby w JS
NEGATIVE_INFINITY	minus nieskończoność
POSITIVE_INFINITY	plus nieskończoność
NaN	Not-a-Number

```
var x1 = Number.MAX_VALUE;          //1.7976931348623157e+308
var x2 = Number.MIN_VALUE;          //5e-324
var x3 = Number.NEGATIVE_INFINITY; //-Infinity
var x4 = Number.NaN;                //NaN
```

Typ Boolean (logiczny)

- Typ **Boolean** reprezentuje dwie wartości: **true** oraz **false**
- Funkcja **Boolean()** zwraca wartość logiczną wyrażenia

```
Boolean(5>2);    //true
//lub bez jawnie wywolanej nazwy funkcji:
5 > 2;           //true
```

- Liczby mają wartość **true**, z wyjątkiem 0, null oraz NaN:

```
var a = 100;
Boolean(a);    //true
```

```
var b = 0;
Boolean(b);    //false
```

```
var c = null;
Boolean(c);    //false
```

```
var d = 10/"kot";
Boolean(d);    //false
```

Typ Boolean (logiczny)

- Łańcuch znaków ma wartość **true**, chyba że jest to łańcuch pusty:

```
var str = "Alicja";
Boolean(str);   // true
```

```
var str = "";
Boolean(str);   // false
```

- Zmienna niezdefiniowana ma wartość logiczną **false**:

```
var x;
Boolean(x);   // false
```

Typ String (łańcuch znaków)

- Zmienna typu **String** przyjmuje wartości będące łańcuchami znaków.
- Łańcuchy znaków definiowane są za pomocą cudzysłowia lub apostrofów:

```
var imie = "Alicja";
var nazwisko = 'Nowak';
```

- Cudzysłów w cudzysłowiu:

```
var zdanie1 = "Tytul lektury to 'Zbrodnia i kara'";
var zdanie2 = 'Tytul lektury to "Zbrodnia i kara"';
var zdanie3 = "Tytul lektury to \"Zbrodnia i kara\"";
```

Typ String (łańcuch znaków)

- Znaki specjalne w łańcuchach znaków:

\’	apostrof
\”	cudzysłów
\\	backslash
\b	backspace (usuwa poprzedzający znak)
\n	nowy wiersz
\r	powrót do początku wiersza (powrót karetki)
\f	wysunięcie strony
\t	tabulator poziomy
\v	tabulator pionowy
\0ooo	znak (ooo) napisany za pomocą liczby w systemie ósemkowym
\xhh	znak (hh) napisany za pomocą liczby w systemie szesnastkowym

Typ String (łańcuch znaków)

- Łamanie długich łańcuchów znaków zgodnie ze standardem ECMAScript

```
var zdanie =
"Tytul lektury to \"Zbrodnia i kara\", jej autorem jest
    Fiodor Dostojewski";
```

```
var zdanie = "Tytul lektury to \"Zbrodnia i kara\"," +
" jej autorem jest Fiodor Dostojewski";
```

- Niezgodnie ze standardem ECMAScrpit

```
var zdanie = "Tytul lektury to \"Zbrodnia i kara\", \
jej autorem jest Fiodor Dostojewski";
```

Metody klasy String

- Długość łańcucha znaków: metoda **length()**

```
var str = "Tytul lektury to \"Zbrodnia i kara\",";
document.write(str.length);
```

Wynik: 35

- Szukanie łańcucha znaków w łańcuchu znaków (Indeks pierwszego znaku to 0!)

```
var str = "Ala ma kota i ten kot jest czarny.";
//indeks pierwszego wystapienia:
var i = str.indexOf("kot");          // 7
var i = str.indexOf("pies");        // -1
var i = str.search("kot");          // 7
//indeks ostatniego wystapienia:
var i = str.lastIndexOf("kot");     // 18
var i = str.lastIndexOf("pies");    // -1
```

Metody klasy String

Wycinanie fragmentów łańcuchów znaków

- **slice**(start, end) - wycina fragment łańcucha znaków i zwraca ten fragment jako nowy string.

```
var str = "Ala ma kota i ten kot jest czarny.";
var str2 = str.slice(18, 21);   //"kot"
var str3 = str.slice(-15, -12); //"kot"
var str4 = str.slice(18);       //"kot jest czarny."
```

  - **start** - indeks początku wycinanego fragmentu,
  - **end** - indeks końca wycinanego fragmentu,
  - jeśli indeks jest ujemny, pozycja liczona jest od końca łańcucha znaków,
  - przy braku drugiego parametru końcem fragmentu jest koniec oryginalnego łańcucha,
  - ujemne indeksy nie działają w przeglądarce Internet Explorer 8 i wcześniejszych!

- **substring**(start, length) - wycina fragment łańcucha znaków i zwraca ten fragment jako nowy string. Działa tak samo jak **slice()**, jednak parametry wejściowe nie mogą być ujemne.

Metody klasy String

Wycinanie fragmentów łańcuchów znaków

- **substr**(start, length) - wycina fragment łańcucha znaków i zwraca ten fragment jako nowy string.

```
var str = "Ala ma kota i ten kot jest czarny.";
var str2 = str.substr(18, 3);     //"kot"
var str3 = str.substr(-15, 3);    //"kot"
var str4 = str.substr(18);        //"kot jest czarny."
```

  - **start** - indeks początku wycinanego fragmentu,
  - **length** - liczba znaków wycinanego fragmentu
  - jeśli indeks początku jest ujemny, pozycja liczona jest od końca łańcucha znaków.
  - przy braku drugiego parametru końcem fragmentu jest koniec oryginalnego łańcucha

Metody klasy String

- **replace**(str1, str2) - zastępuje fragment łańcucha znaków innym łańcuchem

```
var str = "Ala ma kota i ten kot jest czarny.";
var str2 = str.replace("kot", "smok");
//Wynik: Ala ma smoka i ten kot jest czarny.
var str3 = str.replace(/kot/g,"smok");
//Wynik: Ala ma smoka i ten smok jest czarny.
```

- **toUpperCase()** i **toLowerCase()** - zamiana małych liter na wielkie:

```
var str = "Ala ma kota i ten kot jest czarny.";
var str2 = str.toUpperCase();
//Wynik: ALA MA KOTA I TEN KOT JEST CZARNY.
var str3 = str.toLowerCase();
//Wynik: ala ma kota i ten kot jest czarny.
```

Metody klasy String

- **concat(str)** - łączenie łańcuchow znaków

```
var str1 = "Ala ma kota";
var str2 = "i ten kot jest czarny.";
var str3 = str1.concat(" ",str2);
//Wynik: Ala ma kota i ten kot jest czarny.
var str4 = str1 + " " + str2;
//Wynik: Ala ma kota i ten kot jest czarny.
```

- **charAt(pos)** i **charCodeAt(pos)** - pobranie znaku z łańcucha znaków

```
var str = "Ala ma kota i ten kot jest czarny.";
str.charAt(0);          //znak: A
str.charCodeAt(0);      //kod ASCII: 65
```

Operatory i przypisania

- Operatory i przypisania należą do najważniejszych elementów JavaScript.
  - przypisania nadaja wartości zmiennym,
  - operatory wykonują rozmaite typy operacji
- Rodzaje operatorów:
  - przypisania
  - przypisania bitowe
  - porównania
  - inkrementacji i dekrementacji
  - logiczne
  - arytmetyczne

Operatory przypisania

operator	przykład	tożsamy z działaniem
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

```
var x1 = 5;
var x2 = 2;
var x3 = "5";
var x4 = "Ala";
```

```
x1 += 2;   //7
x3 += 2;   //52
x3 /= x2;  //26
x4 += x1;  //  Ala7
x2 %= x3;  //  2
```

Operatory przypisania bitowe

operator	przykład	tożsamy z działaniem	znaczenie
<<=	x <<= y	x = x << y	przesunięcie w lewo
>>=	x >>= y	x = x >> y	przesunięcie w prawo
>>>=	x >>>= y	x = x >>> y	przesunięcie w prawo ze znakiem

```
var n = 48;          // 48 = 110000
n >>= 2;             // 12 = 001100
n <<= 1;             // 24 = 011000
```

```
var n1 = -13, n2 = -13;
//-13=11111111111111111111111111110011
n1 >>= 2;
//-4=11111111111111111111111111111100
n2 >>>= 2;
//1073741820=00111111111111111111111111111100
```

Operatory bitowe

operator	przykład	tożsamy z działaniem	znaczenie
~	x=~y	x=~y	NOT
&=	x &= y	x = x & y	AND
^=	x ^= y	x = x ^ y	XOR
|=	x |= y	x = x | y	OR

NOT	
0	1
1	0

AND	0	1
0	0	0
1	0	1

XOR	0	1
0	0	1
1	1	0

OR	0	1
0	0	1
1	1	1

Operatory bitowe

Jak wyświetlić liczby w postaci binarnej?

```
(parseInt('11001010', 2) & parseInt('1111', 2)).toString(2)
//'1010'
```

```
(parseInt('11001010', 2) | parseInt('1111', 2)).toString(2)
//'11001111'
```

```
(parseInt('11001010', 2) ^ parseInt('1111', 2)).toString(2)
//'11000101'
```

Operatory porównania

==	Zwraca **true** jeżeli operandy są równe
!=	Zwraca **true** jeżeli operandy są różne
===	Zwraca **true** jeżeli operandy są równe i tego samego typu
!==	Zwraca **true** jeżeli operandy nie są równe i/lub nie są tego samego typu
>	Zwraca **true** jeżeli lewy operand jest większy od prawego
>=	Zwraca **true** jeżeli lewy operand jest większy lub równy prawemu
<	Zwraca **true**, jeśli lewy operand jest mniejszy, niż prawy
<=	Zwraca **true** jeżeli lewy operand jest mniejszy lub równy prawemu

```
var x1 = 5;
var x2 = 5;
var x3 = "5";
x1 == x2; //true
x1 != x2; // false
x1 == x3; //true
x1 === x3; //false
```

```
Number(x1) === 5;
//true
x1 < x2;  //false
x1 <= x2; // true
```

```
var  x1 = "ala";
var x2 = "atest";
var x3 = "5";
var x4 = "Ala";
x1 > x2;  //false
x1 > x3;  //true
x1 > x4;  //true
```

Operatory arytmetyczne

+	dodawanie	/	dzielenie	++	inkrementacja
-	odejmowanie	%	modulo	–	dekrementacja
*	mnożenie				

```
var x1 = 5;
var x2 = "2";
var x3 = "Ala";
x1 + x2 // 52
x2 + x1 // 25
x1 * x2 // 10
x2 * x2 // 4
x2 + x3 // 2Ala
x2 - x1 // -3
x2 - x3 // NaN
```

```
var x1 = 5;
var x2 = 2;
var x3 = "5";
var x4 = "Ala";
x1 / x2;  // 2.5
x2 / x3;  //0.4
x3 / x2   //2.5
x1 / x4;  //NaN
x1 % x2;  //1
x3 % x2;  //1
```

```
var x1 = 5;
var x2 = "2";
var x3 = "Ala";
x1++;
//zwraca 5 -
    dodaje do x1
    wartosc 1 -
    zwraca 6
++x2;
//zwraca 3
x3++;
//NaN
```

Operatory logiczne

operator	tożsamy z działaniem
&&	logiczne **and**
||	logiczne **or**
!	logiczne **not**

Przykład:

```
var x = 2, y = "5";
var flaga = (x && !y) || (!x && y);
document.write(flaga);
```

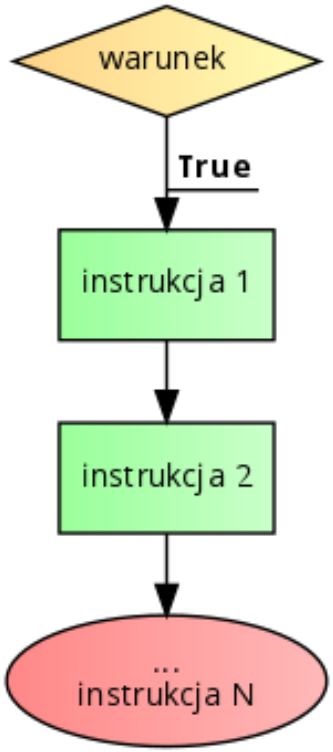
x	y	!x	!y	x && !y	!x && y	flaga
2	“5”	false	false	false	false	false

Wynik: **false**

Instrukcja warunkowa if

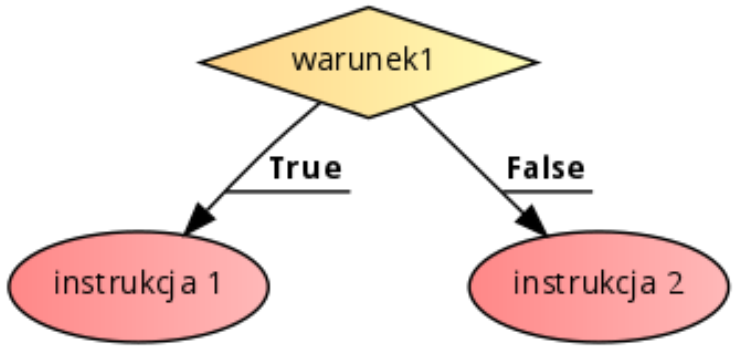
```
if ( warunek ){
   instrukcja 1;
   instrukcja 2;
   ...
   instrukcja N;
}
```

- słowo kluczowe **if** rozpoczyna instrukcję warunkową
- jeśli **warunek** ma wartość logiczną **true,** wykonywane są instrukcję w nawiasach **{ }**



Instrukcja warunkowa if - else

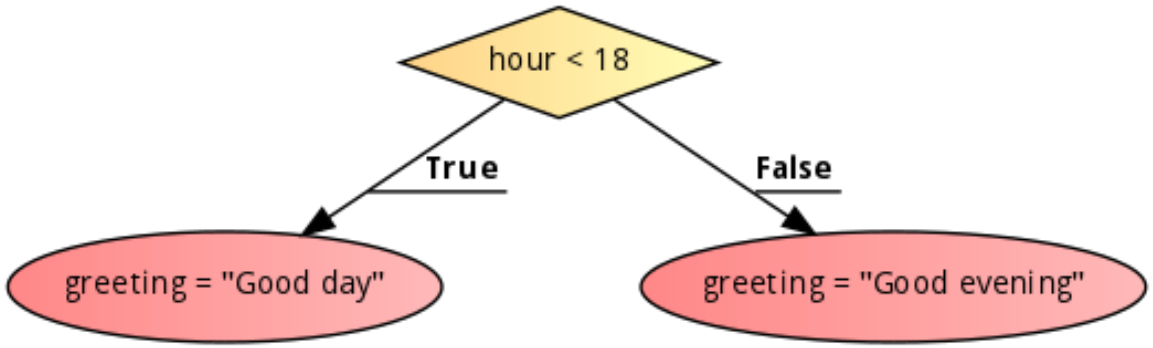
```
if ( warunek )
{
  instrukcja 1;
}
else
{
  instrukcja 2;
}
```



- słowo kluczowe **if** rozpoczyna instrukcję warunkową
- jeśli **warunek** ma wartość logiczną **true,** wykonywane są instrukcję w nawiasach **{ }**
- jeśli **warunek** ma wartość logiczną **false,** wykonywane są instrukcję w nawiasach **{ }** po słowie kluczowym **else**
- dla pojedynczych instrukcji nawiasy **{ }** są opcjonalne!

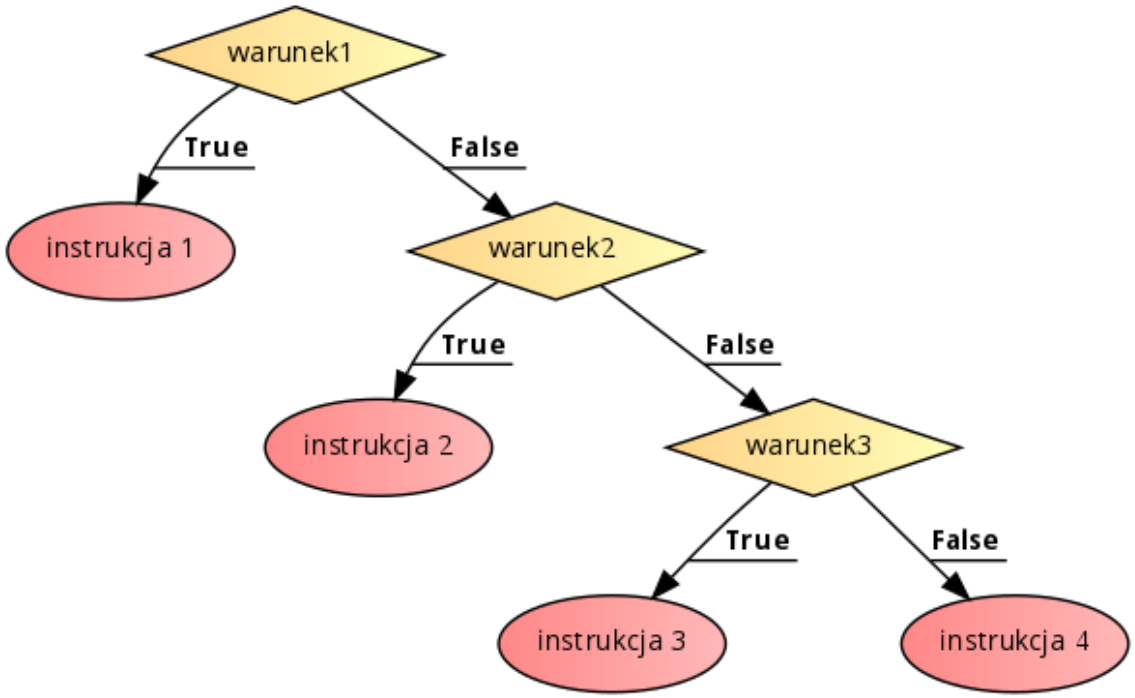
Instrukcja warunkowa if - else

```
if (hour < 18)
  { greeting = "Good day"; }
else
  { greeting = "Good evening"; }
```



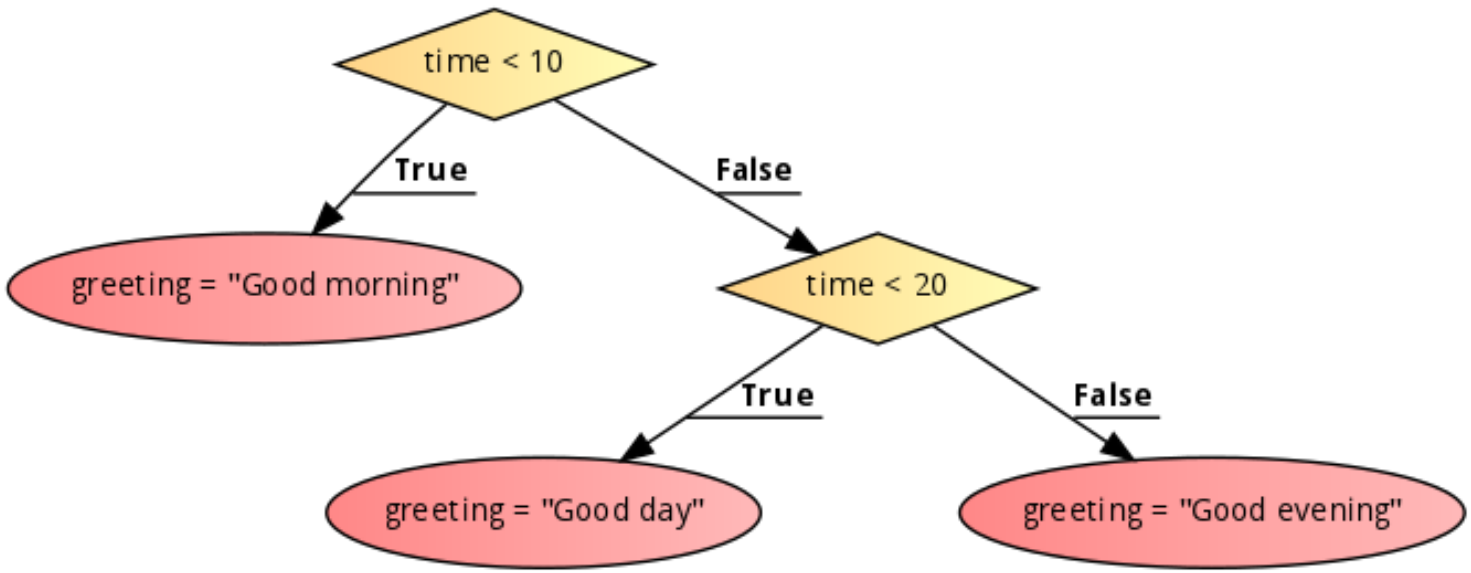
Instrukcja warunkowa if - else

```
if (warunek1)
{
  instrukcja 1;
}
else if (warunek2)
{
  instrukcja 2;
}
else if (warunek3)
{
  instrukcja 3;
}
else
{
  instrukcja 4;
}
```



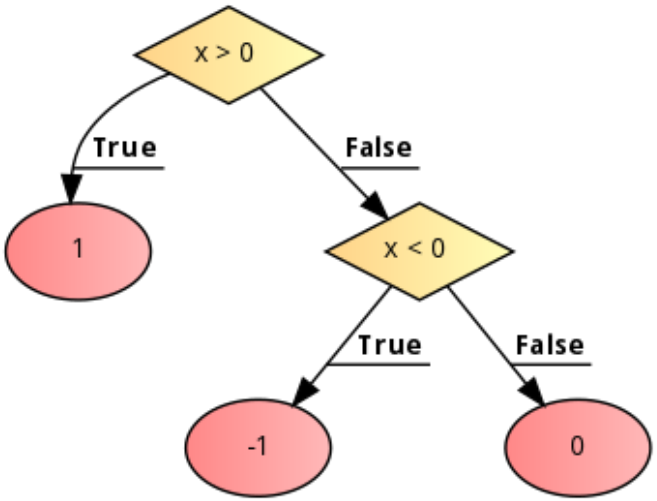
Instrukcja warunkowa if - else

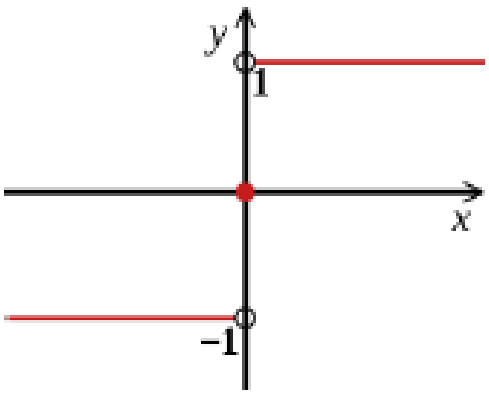
```
if (time < 10) {greeting = "Good morning"; }
else if (time < 20){ greeting = "Good day"; }
else { greeting = "Good evening"; }
```



Instrukcja warunkowa if-else

```
var x;
if (x > 0)
  x = 1;
else if (x < 0)
    x = -1;
else
  x = 0;
```





$$sgn(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

Instrukcja warunkowa if-else

Instrukcje warunkowe zagnieżdżone - przykład

```
var Op_prog = 5, C_prog = 800;
var Op = 10; var C = 1000;
var text = 'Oceniamy telefon'+ '<br>';

if (!(C > 0 && Op > 0))
    text+= 'Niepoprawne dane'+ '<br>';
else {
  text+= 'Model 1: ' + 'opinie = ' +
  Op + ', cena = ' + C + '<br>';

  if (C < C_prog) text+= 'model niedrogi ';
  else text+= 'model drogi ';

  if (Op < Op_prog) text+= 'i zle oceniany';
  else text+= 'i dobrze oceniany';
}
document.write(text);
```

Instrukcja warunkowa if-else

Czy punkt o współrzędnych (x,y) znajduje się **wewnątrz** prostokąta o bokach leżących na prostych: x = 5, x = 10, y = -4, y = 3 ?

```
var xp = 5, xk = 10;
var yp = -4, yp = 3;
var x = 2, y = 2;
var text = '';

// czy punkt lezy wewnatrz kwadratu?
if( (x >= xp) && (x <= xk) && (y >= yp) && (y <= yk))
  text+= 'Punkt lezy wewnatrz kwadratu';
else text+= 'Punkt nie lezy wewnatrz kwadratu';
```

Instrukcja warunkowa if-else

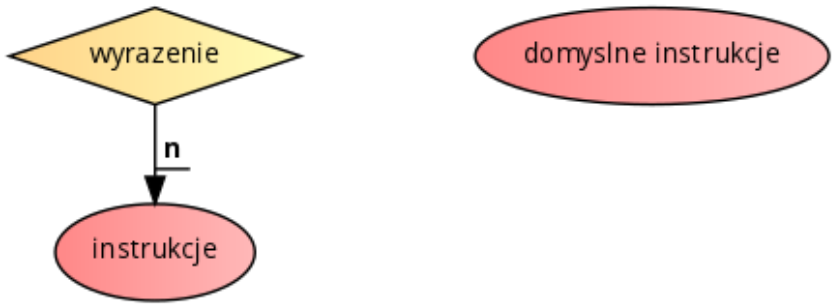
Czy punkt o współrzędnych (x,y) znajduje się **na zewnątrz** prostokąta o bokach leżących na prostych: x = 5, x = 10, y = -4, y = 3 ?

```
var xp = 5, xk = 10;
var yp = -4, yp = 3;
var x = 2, y = 2;
var text = '';

//czy punkt nie lezy wewnatrz kwadratu?
if( !((x < xp) || (x > xk) || (y < yp) || (y > yk))
  text+= 'Punkt lezy wewnatrz kwadratu';
else text+= 'Punkt nie lezy wewnatrz kwadratu';
```

Instrukcja warunkowa switch

```
switch(wyrazenie) {
    case n: instrukcje;
        break;
     case n: instrukcje;
        break;
    default: domyslne
        instrukcje }
```

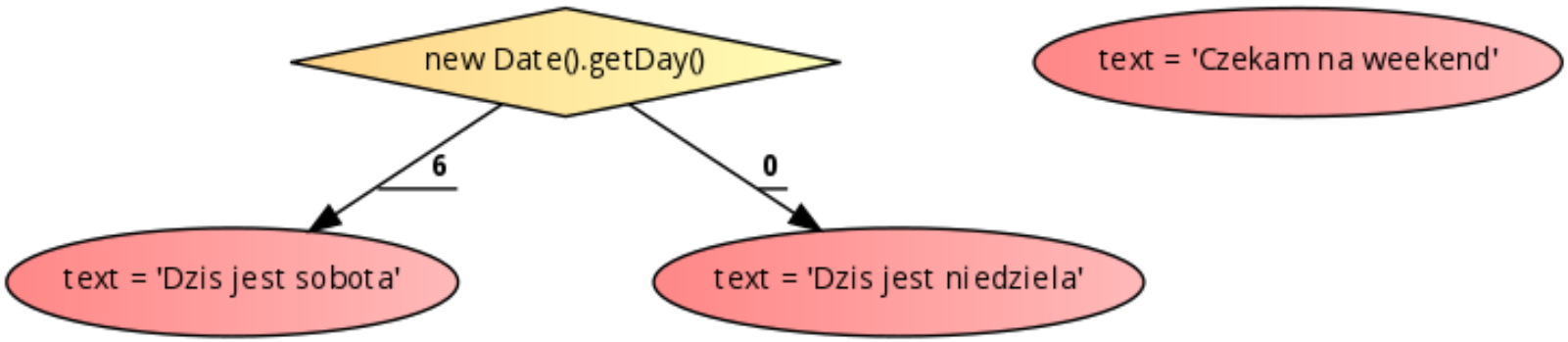


Wartość wyrażenia porównywana jest z wartościami wymienionymi po ’case’. Jeśli są one równe, wykonywany jest kod po dwukropku, aż do instrukcji **break**

Instrukcja warunkowa switch

```
switch (new Date().getDay()) {
    case 0: dzien = "niedziela";
        break;
    case 1: dzien = "poniedzialek";
         break;
    case 2: dzien = "wtorek";
         break;
    case 3: dzien = "sroda";
         break;
    case 4: dzien= "czwartek";
         break;
    case 5: dzien= "piatek";
         break;
    case 6: dzien= "sobota"; }
document.write(dzien);
```

Instrukcja warunkowa switch




```
switch (new Date().getDay()) {
  case 6: text = 'Dzis jest sobota';
      break;
  case 0: text = 'Dzis jest niedziela';
      break;
  default: text = 'Czekam na weekend';
}
document.write(text);
```

Instrukcje warunkowe - częste błędy

- Operator **przypisania** (’=’) zamiast operatora **porównania** (’==’).

```
if (ocena = 5) {
  text+= 'dobry wynik!';
}
```

Wyrażenie-warunek **’ocena = 5’** ma wartość **true**, ponieważ przypisanie wartości **5** do zmiennej **ocena** powiodło się. Instrukcja wewnątrz **if** zostanie wykonana zawsze!

Prawidłowa instrukcja warunkowa:

```
if (ocena == 5) {
  text+= 'dobry wynik!';
}
```

Warto warunek konstruować tak:

```
if (5 == ocena) {
  text+= 'dobry wynik!';
}
```

Instrukcje warunkowe - częste błędy

- Brak nawiasów **{ }** dla bloku instrukcji wykonywanych warunkowo

```
if (ocena == 5)
  liczba_piatek++;
  text+= 'dobry wynik!';
```

Tylko **liczba_piatek++** wykonywana jest warunkowo. Druga instrukcja zostanie wykonana zawsze, ponieważ nie znajduje się wewnątrz **if.**
Prawidłowa instrukcja warunkowa:

```
if (ocena == 5) {
  liczba_piatek++;
  text+= 'dobry wynik!';
}
```

Instrukcje iteracyjne

- Instrukcje iteracyjne (tzw. “pętle”) to bloki instrukcji wykonywane wielokrotnie.
- Przykłady:
  - sprawdzenie, czy użytkownik wypełnił wszystkie 53 pola formularza
  - obsługa tablic
- Trzy instrukcje iteracyjne:
  - instrukcja **for**
  - instrukcja **while**
  - instrukcja **do-while**

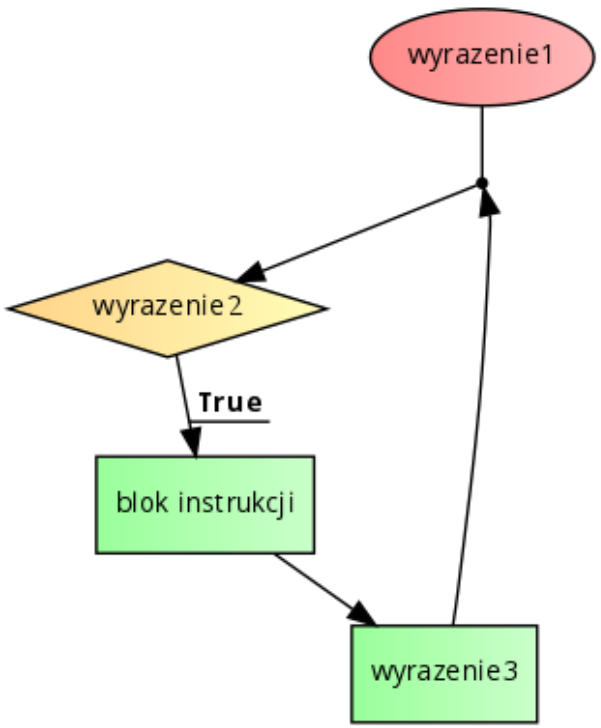
Instrukcja iteracyjna **for**

```
for (wyrazenie1; wyrazenie2; wyrazenie3)
{
  blok instrukcji;
}
```

wyrażenie1 - instrukcja wykonywana przed blokiem instrukcji w pętli

wyrażenie2 - warunek, który musi być spełniony, by blok instrukcji został wykonany

wyrażenie3 - instrukcja wykonywana po bloku instrukcji w pętli



Instrukcja iteracyjna **for**

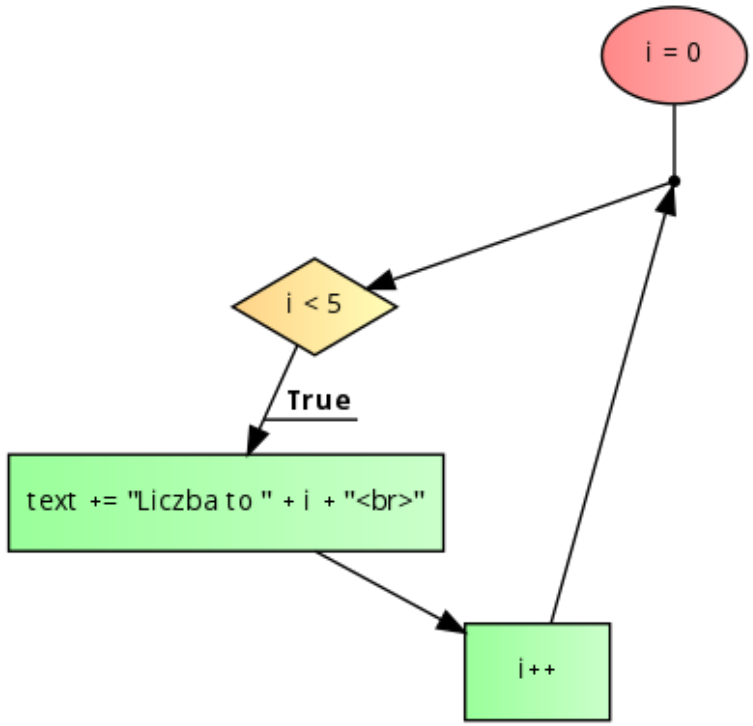
Przykład:

```
for (i = 0; i < 5; i++)
{
  text += "Liczba to " + i + "<br>";
}
```

i = 0 - wykonane przed blokiem instrukcji w pętli

i < 5 - instrukcja iteracyjna jest wykonywana, jeśli to wyrażenie ma wartość **true**

i++ - wykonane na koniec każdej iteracji



Instrukcja iteracyjna **for**

- W miejscu **wyrażenie1** może się znaleźć kilka instrukcji

```
for (i = 0, text=""; i < 5; i++)
{
  text += "Liczba to " + i + "<br>";
}
```

- **wyrażenie1** może się też w ogóle nie pojawić

```
var i = 0;
for ( ; i < 5; i++)
{
  text += "Liczba to " + i + "<br>";
}
```

Instrukcja iteracyjna **for**

- **wyrażenie2** również można pominąć. Konieczne jest wówczas użycie instrukcji **break** wewnątrz pętli, która przerwie jej działanie

```
var i = 0;
for (;; i++)
{
  text += "Liczba to " + i + "<br>";
  if (i >= 5)
    break;
}
```

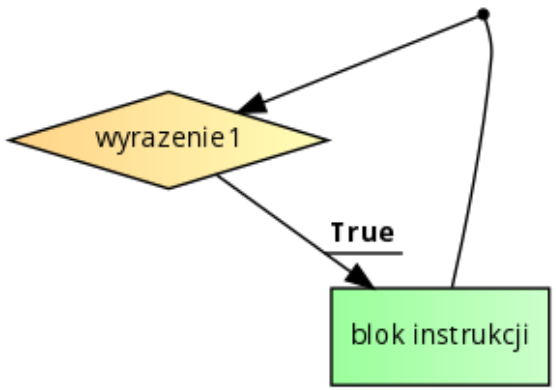
- ... pominąć można też **wyrażenie3** :)

```
var i = 0;
for (;;)
{
  text += "Liczba to " + i + "<br>";
  if (i >= 5)
    break;
  i++;
}
```

Instrukcja iteracyjna **while**

```
while (wyrazenie1)
{
  blok instrukcji;
}
```

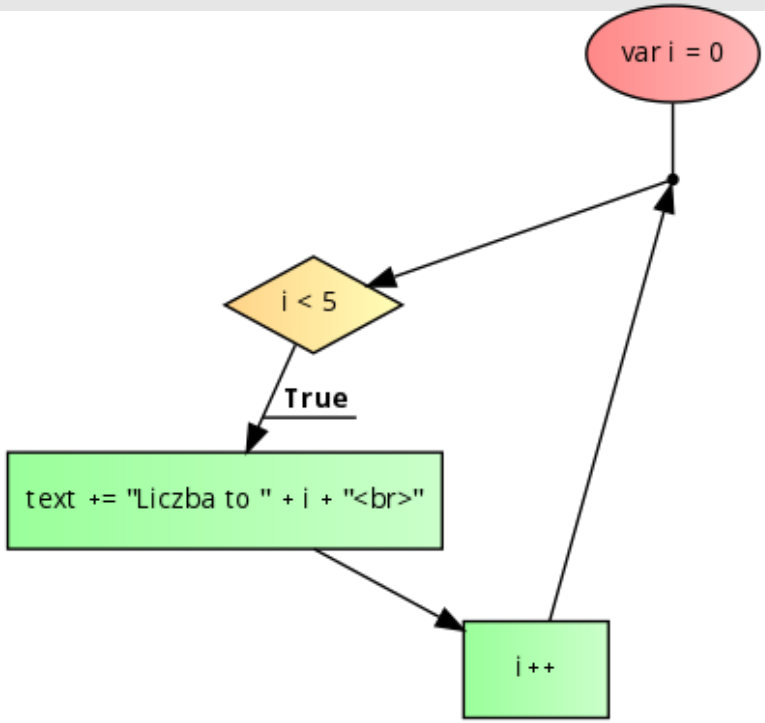
wyrażenie1 - warunek, który musi być spełniony, by blok instrukcji został wykonany



Instrukcja iteracyjna **while**

```
var i = 0;
while( i < 5 )
{
  text += "Liczba to " + i + "<br>";
  i++;
}
```

i < 5 - instrukcja iteracyjna jest wykonywana, jeśli to wyrażenie ma wartość **true**

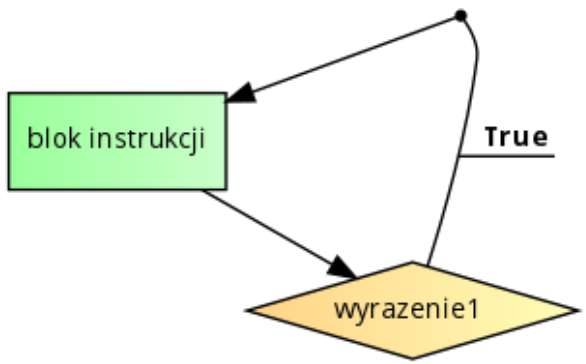


Instrukcja iteracyjna **do-while**

```
do
{
  blok instrukcji;
} while (wyrazenie1);
```

wyrażenie1 - warunek, który musi być spełniony, by blok instrukcji został wykonany po raz kolejny

blok instrukcji zostanie wykonany raz, zanim zostanie sprawdzona wartość logiczna **wyrażenie1**



Instrukcja iteracyjna **do-while**

```
var i = 5, text = '';
while( i < 5 )
{
  text += "Liczba to " + i +
      "<br>";
  i++;
}
document.write(text);
```

Wynik: pusty tekst

```
var i = 5, text = '';
do
{
  text += "Liczba to " + i +
      "<br>";
  i++;
}while( i < 5 );
document.write(text);
```

Wynik: ***Liczba to 5***

Instrukcje iteracyjne - częste błędy

- Brak nawiasów **{ }** dla bloku instrukcji wykonywanych iteracyjnie.

```
var i = 0;
while ( i < 5 )
  text += "Liczba to " + i + "<br>";
  i++;
```

Liczba **i** nie jest zwiększana wewnątrz instrukcji iteracyjnej. W konsekwencji instrukcja **while** wykonywana jest nieskończenie wiele razy.

- **Średnik** kończący instrukcję iteracyjną

```
for ( i = 0; i < 5; i++ );
  text += "Liczba to " + i + "<br>";
```

Instrukcja iteracyjna kończy się wraz ze średnikiem. Druga instrukcja w kodzie zostanie wykonana tylko raz.

Tablice

Tablica - umożliwia przechowywanie wielu wartości w jednej zmiennej

```
var nazwa_tablicy = [element1, element2, ...];
```

Deklaracja może zajmować kilka linii kodu:

```
var osoby = [
      "Piotr",
      "Jan",
      "Adam"
];
```

Inna forma deklaracji tablicy (rzadziej używana):

```
var osoby = new Array(element1, element2, ...);
```

Dostęp do elementów tablicy

- Do elementów tablicy odwołujemy się za pomocą **indeksu**
- **Pierwszy** element tablicy ma **indeks** równy **0**
- Odczyt pierwszego elementu tablicy:

```
var imie = osoby[0];
```

- Zapis drugiego elementu tablicy:

```
osoby[1] = "Andrzej";
```

- Odczyt całej tablicy:

```
document.write(osoby);
```

wynik: Piotr, Andrzej, Adam

Dostęp do elementów tablicy

- **Liczba elementów** tablicy**:**

```
document.write(osoby.length);
```

- Odczyt ostatniego elementu tablicy:

```
var imie = osoby[osoby.length-1];
```

- W tablicy mogą znaleźć się elementy różnego typu:

```
var tablica = ["Andrzej", "Damian", 25] ;
```

Dostęp do elementów tablicy

- Dodawanie elementów na koniec tablicy:

```
osoby[osoby.length] = "Andrzej";
```

lub za pomocą metody **push()**:

```
osoby.push("Andrzej");
```

- Metoda **push()** umożliwia dodanie wielu elementów jednocześnie:

```
osoby.push("Andrzej", "Damian", "Patryk");
```

```
var tablica = ["Andrzej", "Damian", 25] ;
tablica.push("Jan", 23.2, 11.1, "Anna");
document.write(tablica);
```

Wynik: Andrzej,Damian,25,Jan,23.2,11.1,Anna

Dostęp do elementów tablicy

- Dodawanie elementów na początek tablicy

```
var osoby = ["Piotr", "Jan", "Adam"];
osoby.unshift(45, "Andrzej");
document.write(osoby);
```

Wynik: 45,Andrzej,Piotr,Jan,Adam

- Usuwanie elementu z końca tablicy

```
var tablica = ["Andrzej", "Damian", 25] ;
var usuniety = tablica.pop(); //25
```

- Usuwanie elementu z początku tablicy

```
var tablica = ["Andrzej", "Damian", 25] ;
var usuniety = tablica.shift(); //Andrzej
```

- Usuwanie elementu o dowolnym indeksie

```
var tablica = ["Andrzej", "Damian", 25] ;
delete tablica[1]; //usuwa drugi element tablicy
```

Przetwarzanie tablicy

- Metoda **splice(p1, p2, p3, ..)** - dodaje/usuwa elementy do tablicy
  p1 - indeks, począwszy od którego należy zmienić tablicę,
  p2 - ile elementów trzeba usunąć,
  p3,... - nowe elementy tablicy.

```
var osoby = ["Andrzej", "Damian", 25] ;
osoby.splice(1, 0, "Leon", "Konrad");
document.write(osoby);
```

Wynik: Andrzej,Leon,Konrad,Damian,25

```
var osoby = ["Andrzej", "Damian", 25] ;
osoby.splice(1, 1); //usuwa drugi element
document.write(osoby);
```

Wynik: Andrzej,25

Przetwarzanie tablicy

- Metoda **join()** - łączenie elementów tablicy w łańcuch znaków

```
var osoby = ["Andrzej", "Damian", 25] ;
document.write(osoby.join(" - "));  // Andrzej-Damian-25
```

- Metoda **concat()** - łączy tablice ze sobą (i zwraca nową tablicę)

```
var osoby1 = ["Andrzej", "Damian", 25];
var osoby2 = ["Jan"];
var osoby3 = ["Aneta", "Daria"];
var osoby4 = osoby1.concat(osoby2, osoby3);
document.write(osoby4);
```

Wynik: Andrzej,Damian,25,Jan,Aneta,Daria

Przetwarzanie tablicy

- Metoda **slice(p1,p2)** - wycina wskazany fragment tablicy (i zwraca nową tablicę)
  p1 - indeks, począwszy od którego należy wyciąć fragment tablicy,
  p2 - jak długi fragment (brak tego parametru oznacza wycięcie wszystkich elementów począwszy od p1 do końca tablicy),

```
var osoby1 = ["Andrzej", "Damian", 25];
var osoby2 = osoby1.slice(1);
document.write(osoby2);
```

Wynik: Damian,25

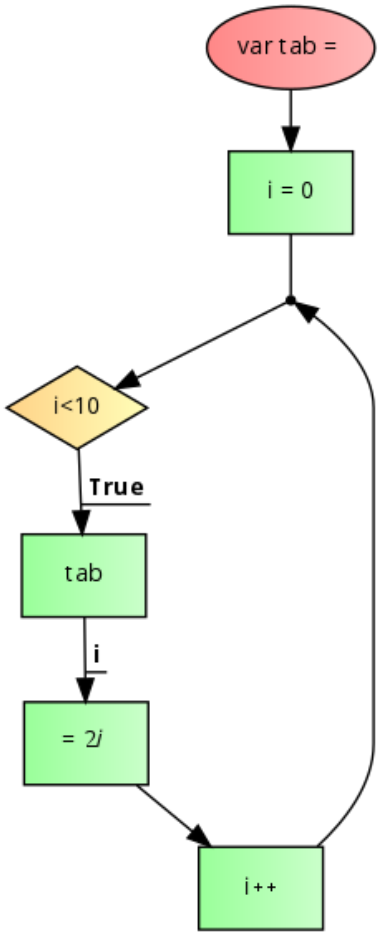
```
var osoby1 = ["Andrzej", "Damian", 25];
var osoby2 = osoby1.slice(0,2);
document.write(osoby2);
```

Wynik: Andrzej, Damian

Iteracyjne przetwarzanie tablicy

```
var tab = [];
for (i = 0; i<10; i++)
{
  tab[i] = i*2;
}
document.write(tab);
```

Wynik: 0,2,4,6,8,10,12,14,16,18



Iteracyjne przetwarzanie tablicy

Kolejność wykonywania działań

```
var tab = [];
var i = 0;
while (i<10)
{
  tab[i] = (i++)*3;
}
document.write(tab);
```

Wynik: 0,3,6,9,12,15,18,21,24,27

```
var tab = [];
var i = 0;
while (i<10)
{
  tab[i++] = i*3;
}
document.write(tab);
```

Wynik: 3,6,9,12,15,18,21,24,27,30

Iteracyjne przetwarzanie tablicy

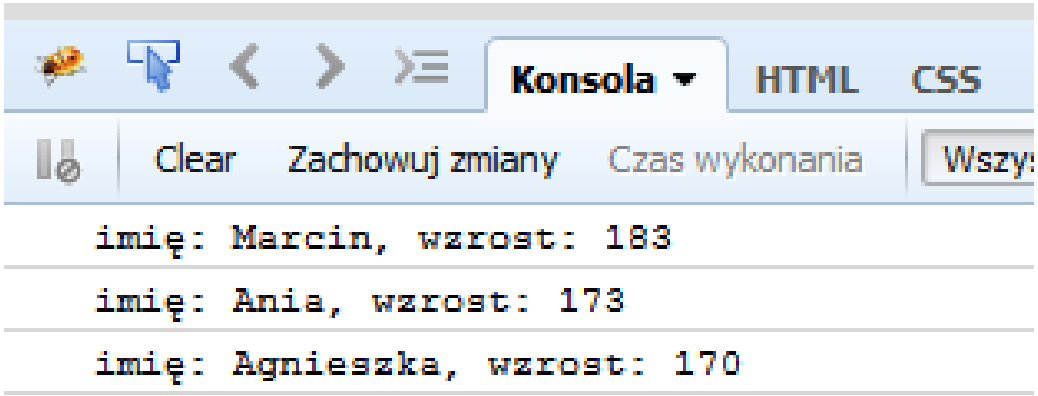
```
var osoby, text, o_len, i;
osoby = ["Jan", "Adam", "Piotr", "Andrzej", "Damian"];
o_len = osoby.length;
text = "Osoby:" + "<br>" + "<ul>";
for (i = 0; i < o_len; i++)
{
  text += "<li>" + osoby[i] + "</li>";
}
text += "</ul>";
document.write(text);
```

Osoby:

- Jan
- Adam
- Piotr
- Andrzej
- Damian

Tablice wielowymiarowe

```
var tablica = [];
tablica[0] = ['Marcin' , '183'];
tablica[1] = ['Ania' , '173'];
tablica[2] = ['Agnieszka' , '170'];
for( i=0; i<3 ; i++)
  console.log('imie: ' + tablica[i][0] + ', wzrost: ' +
      tablica[i][1]);
```



Jak sprawdzić, czy zmienna jest tablicą?

1. Operator **typeof** - zwraca typ zmiennej, jednak tablica jest dla niego obiektem

```
osoby = ["Jan", "Adam", "Piotr"];
typeof osoby;    // zwraca: object
```

2. Metod **Array.isArray()** - tylko w ECMAScript 5

```
osoby = ["Jan", "Adam", "Piotr"];
Array.isArray(osoby);    // zwraca: true
```

3. Operator **instanceof** - sprawdza, czy obiekt został utworzony przez dany konstruktor

```
osoby = ["Jan", "Adam", "Piotr"];
osoby instanceof Array;    // zwraca: true
```