POLITECHNIKA GDAŃSKA

Aplikacje i usługi internetowe

Podstawy Programowania

Iwona Kochańska

Funkcje

Funkcje

- **Funkcje (podprogramy)** pozwalają wielokrotnie wykonywać powtarzalne operacje.
- Podstawowa struktura funkcji

```
function nazwaFunkcji(parametr1, parametr2, ...){
  //blok instrukcji
}
```

nazwaFunkcji - obowiązują te same zasady, co przy wyborze nazw zmiennych
parametr1, parametr2, ... - parametry wejściowe

Funkcje

- **Definicja** funkcji: a i b to **parametry** wejściowe

```
function wypiszParametry(a, b){
  document.write("argument1: " + a + "</br>");
  document.write("argument2: " + b + "</br>");
}
```

- **Wywołanie** funkcji: c i d to **argumenty** wejściowe

```
var c = 3;
var d = "Ala ma kota";
wypiszParametry(a, b);
```

Funkcje

- Funkcja może zwracać wartość za pomocą instrukcji **return**

```
function dodajDwieLiczby(a, b){
  return a+b;
}
```

- Jeśli za instrukcją **return** nie ma żadnego wyrażenia, funkcja zwraca wartość **undefined**.
- W treści funkcji może znajdować się dowolna liczba instrukcji.
- Nazwy argumentów funkcji w jej wnętrzu są dostępne jako zmienne. Odnoszą się do wartości, które zostały w ich miejsce przekazane w wywołaniu funkcji.

Funkcje

- Definicja funkcji:

```
function dodajDwieLiczby(a, b){
  return a+b;
}
```

- Wywołanie funkcji:

```
var x = dodajDwieLiczby(3,4);
document.write(x);
```

Wynik: 7

- Wywołanie nazwy funkcji bez nawiasów () zwraca jej definicję

```
var x = dodajDwieLiczby;
document.write(x);
```

Wynik: function dodajDwieLiczby(a, b) { return a+b; }

Funkcje

Wartości zwracanej przez funkcję można użyć tak, jak zmiennej

```
var text = 'Suma liczb wynosi ' + dodajDwieLiczby(3,4);
document.write(text);
```

wynik: Suma liczb wynosi 7

```
document.write(dodajDwieLiczby(5,6));
```

wynik: 11

Funkcje

- Nie jest konieczne przekazywanie do funkcji ścisłej liczby argumentów.
- Możliwe jest przekazanie argumentów przez tablicę **arguments**

```
//definicja funkcji:
function calculate(){
  var argLength = arguments.length;
    if (argLength == 0){
    console.warn('Blad: Nie podales zadnych liczb');
  }
  else if (argLength == 1) {
        console.warn('Podales tylko jedna liczbe:' +
            arguments[0]);
    }
  else {
      var result = 0;
      for (i = 0; i < arguments.length; i++) {
            result += arguments[i];
      }
      console.log(result);
    }
}
```

Funkcje

- Przykład przekazania argumentów przez tablicę **arguments**

```
//wywoanie funkcji
x = suma(1, 123, 500, 115, 44, 88);
//definicja funkcji
function suma() {
  var i, sum = 0;
  for (i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```

Funkcje

Argumenty przekazywane są do funkcji przez **wartość.** Po wyjściu z funkcji wartości zmiennych a, b i c pozostają niezmienione!

```
var a = 5; var b = "Ala";
document.write("Przed fun. a = "+a+"  b = "+b+"</br>");
przyklad(a, b);
document.write("Po fun. a = "+a+"  b = "+b+"</br>");
function przyklad(a, b, c){
  a = 7;
  b = "ma kota";
  document.write("W fun. a = "+a+"  b = "+b+"</br>");
  return a+b;
}
```

Przed funkcja: a = 5 b = Ala
W funkcji: a = 7 b = ma kota
Po funkcji: a = 5 b = Ala

Funkcje

Jeśli definicja funkcji zawiera parametry wejściowe, ale w wywołaniu nie podamy takiej samej liczby argumentów, to brakującym argumentom przypisywana jest wartość **undefined**

```
function suma(a, b, c){
  document.write("Argument c w funkcji = " + c + "  ");
  return a+b+c;
}
document.write("Wynik dzialania funkcji = " + suma(2,3));
```

Argument c w funkcji = undefined, wynik dzialania funkcji = NaN

```
function suma(a, b, c){
  if (c === undefined){c = 0;}
  document.write("Argument c w funkcji = " + c + "  ");
  return a+b+c;
}
document.write("Wynik dzialania funkcji = " + suma(2,3));
```

Argument c w funkcji = 0, wynik dzialania funkcji = 5

Funkcje

Podczas szukania zmiennej wewnątrz funkcji najpierw przeszukiwane jest środowisko lokalne i dopiero, gdy nie uda się jej tam znaleźć przeszukiwane jest główne środowisko. Dzięki temu zmienne znajdujące się wewnątrz funkcji mogą „zasłaniać” zmienne z głównego środowiska o takich samych nazwach.

```
var a = 3, b = 7;
var text = '';

var c = dodajDwieLiczby(8,9);

text+= 'W programie glownym: ' +
    'a = ' + a + ' b = ' + b + '<br>';
document.write(text);

function dodajDwieLiczby(a, b) {
  text+= 'W funkcji: '+'a = '
    + a + ' b = ' + b + '<br>';
  return a+b;
}
```

wynik:
W funkcji: a = 8 b = 9

Funkcje

Zmienna utworzona w funkcji istnieje tylko w tej funkcji.

```
var a = 3, b = 7;
var c = pomnozDwieLiczby(a,dodajDwieLiczby(a,b));
document.write('Wynik: '+d);
//Wynik: Unresolved variable or type d

function dodajDwieLiczby(a, b){
     return a+b;
}
function pomnozDwieLiczby(a, b){
    var d = a*b;
    return d;
}
```

Funkcje

- Parametrem wejściowym funkcji może być wartość zwaracana przez inną funkcję

```
var a = 3, b = 7;
var c = pomnozDwieLiczby(a,dodajDwieLiczby(a,b));
document.write('Wynik: '+c);

function dodajDwieLiczby(a, b){
     return a+b;
}
function pomnozDwieLiczby(a, b){
    return a*b;
}
```

- wynik: 30

Funkcje

- Funkcja może być zdefiniowana jako wyrażenie:

```
var c = function (a, b) {return a * b};
```

- jest to **funkcja anonimowa** - nie posiada nazwy,
- można ją wywołać poprzez odwołanie do zmiennej, do której została przypisana:

```
var c = function (a, b) {return a * b};
var d = c(4, 3);
```

Funkcje

- co zwróci metoda **typeof** zawołana dla funkcji?

```
var c = function (a, b) {return a * b};
document.write(typeof(c));
```

wynik: function

- Obiekt reprezentowany przez funkcję można zamienić na łańcuch znaków:

```
var c = function (a, b) {return a * b};
var txt = c.toString();
document.write(txt);
```

wynik: function (a, b) {return a * b}

Funkcje

- co zwróci metoda **typeof** zawołana dla funkcji?

```
function iloczyn(a, b) {return a * b};
document.write(typeof(iloczyn));
```

wynik: function

- Obiekt reprezentowany przez funkcję można zamienić na łańcuch znaków:

```
function iloczyn(a, b) {return a * b};
var txt = iloczyn.toString();
document.write(txt);
```

wynik: function iloczyn(a, b) {return a * b}

Funkcje

- Funkcja może byc utworzona przez specjalną funkcję JS - konstruktor **Function()**

```
var iloczyn = new Function("a", "b", "return a * b");
var x = iloczyn(4, 3);
```

- Tą metodę stosuje się rzadko!

Typ Object

Typ Object

- **Obiekt**, w przeciwieństwie do zmiennej typu prostego, może zawierać w sobie wiele wartości (typu prostego lub złożonego)
- Przykład:

```
var osoba = {imie:"Jan", nazwisko:"Kowalski", wiek:40,
    wzrost:180};
```

lub:

```
var osoba = {
  imie:"Jan",
  nazwisko:"Kowalski",
  wiek:40,
  wzrost:180
};
```

```
document.write(osoba);
```

wynik: [object Object]

Typ Object

- Obiekt może być utworzony za pomocą słowa kluczowego **new**

```
var osoba = new Object();
osoba.imie = "Jan";
osoba.nazwisko = "Kowalski";
osoba.wiek = 40;
osoba.wrost = 180;
```

```
document.write(osoba);
```

wynik: [object Object]

Tworzenie obiektu za pomocą konstruktora

- Definiując specjalną funkcję zwaną **konstruktorem** możemy tworzyć wiele obiektów naszego własnego “typu”

```
function osoba(im, nazw, wiek, wzrost) {
    this.imie = im;
    this.nazwisko = nazw;
    this.wiek = wiek;
    this.wzrost = wzrost;
}
```

- Zadaniem konstruktora jest określenie właściwości (**properties**) obiektu
- Słowo kluczowe **this** oznacza “ten obiekt”

Tworzenie obiektu za pomocą konstruktora

- Przykład tworzenia obiektów za pomocą konstruktora:

```
var osoba1 = new osoba("Adam", "Kowalski", 40, 180);
var osoba2 = new osoba("Adam", "Kwiatkowski", 34, 170);
```

```
document.write(osoba1);
```

wynik: [object Object]

```
document.write(osoba1.imie + " " + osoba1.nazwisko);
```

wynik: Adam Kowalski

Wbudowane konstruktory JS

- W języku JavaScript istnieją konstruktory obiektów różnych typów, również prostych
- Jednak w większości przypadków nie ma potrzeby ich używania

zamiast używać konstruktora:

```
var x1 = new Object();
var x2 = new String();
var x3 = new Number();
var x4 = new Boolean();
var x5 = new Array();
var x6 = new Function();
```

lepiej zrobić tak:

```
var x1 = {};
var x2 = "";
var x3 = 0;
var x4 = false;
var x5 = [];
var x6 = function(){};
```

Odwołania do obiektów - przez referencję

- Do zmiennych **typu prostego** odwołujemy się przez **wartość**

```
var x = 15;
var y = x;
y *= 2;
document.write( " y = " + y + " x = " + x);
```

wynik: y = 30 x = 15

- Do **obiektów** odwołujemy się przez **referencję**, czyli **adres w pamięci**

```
var x = new osoba("Adam", "Kowalski", 40, 180);
var y = x;
// od tego momentu x i y sa referencjami do tego samego
    obiektu!
y.imie = "Jan";
document.write( " y.imie = " + y.imie + " x.imie = " + x
    .imie);
```

wynik: y.imie = Jan x.imie = Jan

Dostęp do właściwości obiektu

- Do właściwości (properties) obiektu można się odwołać na trzy sposoby:

```
objectName.property              // osoba.imie
objectName["property"]           // osoba["imie"]
objectName[expression]           // x = "imie"; osoba[x]
```

- Przykład:

```
var x = new osoba("Adam", "Kowalski", 40, 180);
document.write(x["imie"] + " " + x.nazwisko + " ma " +
    x['wiek'] + " lat.");
```

wynik: Adam Kowalski ma 40 lat.

Dostęp do właściwości obiektu - pętla for .. in

- Do kolejnych właściwości (properties) obiektu można odwołać się poprzez pętlę **for .. in**

```
var adas = new osoba("Adam", "Kowalski", 40, 180);
var text = "";
for (z in adas) {
    text += adas[z] + " ";
}
document.write(text);
```

```
function osoba(im, nazw, wiek, wzrost) {
    this.imie = im;
    this.nazwisko = nazw;
    this.wiek = wiek;
    this.wzrost = wzrost;
}
```

wynik: Adam Kowalski 40 180

Dodawanie właściwości obiektu

- Do istniejącego obiektu można dodać właściwość:

```
adas.kolor_oczu = "niebieski";
var text = "";
for (z in adas) {
    text += adas[z] + " ";
}
document.write(text);
```

```
function osoba(im, nazw, wiek, wzrost) {
    this.imie = im;
    this.nazwisko = nazw;
    this.wiek = wiek;
    this.wzrost = wzrost;
}
```

wynik: Adam Kowalski 40 180 niebieski

Usuwanie właściwości obiektu

- Właściwość można również usunąć:

```
delete adas.wiek;
var text = "";
for (z in adas) {
    text += adas[z] + " ";
}
document.write(text);
```

wynik: Adam Kowalski 180 niebieski

```
document.write(adas.wiek);
```

wynik: undefined

Metody

- **Metoda (method)** to **właściwość** obiektu, która jest **funkcją**

```
var osoba = {
  imie:"Jan",
  nazwisko:"Kowalski",
  wiek:40,
  wzrost:180,
  tozsamosc: function(){
      return this.imie + " " + this.nazwisko;
    }
};
```

```
document.write(osoba.tozsamosc());//wypisanie tego, co
    zwraca metoda tozsamosc()
```

wynik: Jan Kowalski

```
document.write(osoba.tozsamosc);//wypisanie definicji
    metody tozsamosc()
```

wynik: function (){ return this.imie + " " + this.nazwisko; }

Metody

- **Metoda (method)** definiowana w konstruktorze:

```
function osoba(im, nazw, wiek, wzrost) {
    this.imie = im;
    this.nazwisko = nazw;
    this.wiek = wiek;
    this.wzrost = wzrost;
    this.tozsamosc = function(){
        return this.imie + " " + this.nazwisko;
   };
}
```

```
var janek = new osoba("Jan", "Kowalski", 34, 178);
document.write(osoba.tozsamosc());
```

wynik: Jan Kowalski

Data i czas

Obiekt Date

- Obiekt **Date** pozwala określić rok, miesiąc, dzień tygodnia, godzinę i inne informacje związane z datą i czasem
- Konstruktor **Date()** zwraca bieżącą datę i godzinę:

```
document.write(Date());
```

wynik: **Wed Dec 14 2016 00:01:46 GMT+0100 (Europa Zachodnia (czas stand.))**

- Utworzenie nowego obiektu typu Date:

```
var teraz = new Date();
```

Obiekt Date

- Wypisanie daty z obiektu teraz:

```
document.write(teraz);
```

wynik: Wed Dec 14 2016 00:01:46 GMT+0100 (Europa Zachodnia (czas stand.))

```
document.write(teraz.toString());
```

wynik: Wed Dec 14 2016 00:01:46 GMT+0100 (Europa Zachodnia (czas stand.))

Metody pobierające właściwości obiektu Date

metoda	zwraca
getDate()	dzień jako liczba z zakresu od 1 do 31
getDay()	dzień tygodnia jako liczba z zakresu od 0 (niedziela) do 6 (sobota)
getFullYear()	rok
getHours()	godzina jako liczba z zakresu od 0 do 23
getMiliseconds()	milisekundy jako liczba z zakresu od 0 do 999
getMinutes()	minuty jako liczba z zakresu od 0 do 59
getMonth()	miesiąc jako liczba z zakresu od 0 do 11
getSeconds()	sekundy jako liczba z zakresu od 0 do 59
getTime()	czas jako liczba milisekund od 1 stycznia 1970 roku

Obiekt Date - nazwa miesiąca

- Metoda **getMonth()** zwraca liczbę z zakresu od 0 do 11. Można ją wykorzystać jako indeks tablicy zawierającej nazwy miesięcy:

```
var miesiace = ['stycze', 'luty', 'marzec',
                'kwiecie', 'maj', 'czerwiec',
                'lipiec', 'sierpie', 'wrzesie',
                'padzernik', 'listopad', 'grudzie'];
var teraz = new Date();
document.write(miesiace[teraz.getMonth()]);
```

wynik: grudzień

Obiekt Date - godzina w formacie 12-godzinnym

- Metoda **getHours()** zwraca liczbę z zakresu od 0 do 23. Jak ją wyświetlić w formacie 12-godzinnym?

```
var teraz = new Date();
var godzina = teraz.getHours();
if ( godzina > 12){
    postfix = ' pm';
    godzina = godzina - 12;
}
else{
    postfix = ' am';
}
godzina = godzina + postfix;
document.write(godzina);
```

wynik: 11 pm

Obiekt Date - metoda toUTCString()

- Metoda **toUTCString()** konwertuje czas do standardu UTC:

```
var teraz = new Date();
document.write('</br>' + teraz.toString() + '</br>');
document.write('</br>' + teraz.toUTCString() + '</br>');
```

wynik:

Wed Dec 14 2016 00:05:58 GMT+0100 (Europa Zachodnia (czas stand.))

Tue, 13 Dec 2016 23:05:58 GMT

Obiekt Date - metoda toDateString()

- Metoda **toDateString()** zwraca datę:

```
var teraz = new Date();
document.write('</br>' + teraz.toString() + '</br>');
document.write(teraz.toDateString() + '</br>');
```

wynik:

Wed Dec 14 2016 00:05:58 GMT+0100 (Europa Zachodnia (czas stand.))

Wed Dec 14 2016

Obiekt Date - data inna niż bieżąca

- Konstruktor **Date()** umożliwia utworzenie obiektu reprezentującego dowolną datę na 3 sposoby:

```
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds,
    milliseconds)
```

- Przykłady:

```
var d = new Date(86400000);
var d = new Date("October 13, 2014 11:13:00");
var d = new Date(99,5,24,11,33,30,0);
lub:
var d = new Date(99,5,24,11,33);
```

Obiekt Date - konstruktor z dateString

- Konstruktor Date() akceptuje łańcuchy znaków reprezentujących datę zgodnie z formatami:

typ	przykład
ISO Date	"2015-03-25"
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" lub "25 Mar 2015"
Full Date	"Wednesday March 25 2015"

Metody zmieniające właściwości obiektu Date

metoda	zwraca
setDate()	dzień jako liczba z zakresu od 1 do 31
setDay()	dzień tygodnia jako liczba z zakresu od 0 (niedziela) do 6 (sobota)
setFullYear()	rok
setHours()	godzina jako liczba z zakresu od 0 do 23
setMiliseconds()	milisekundy jako liczba z zakresu od 0 do 999
setMinutes()	minuty jako liczba z zakresu od 0 do 59
setMonth()	miesiąc jako liczba z zakresu od 0 do 11
setSeconds()	sekundy jako liczba z zakresu od 0 do 59
setTime()	czas jako liczba milisekund od 1 stycznia 1970 roku

Zmiana właściwości obiektu Date - przykłady

- ustawienie elementów daty:

```
var d = new Date();
d.setFullYear(2020, 0, 14); // ustawienie daty
    14.01.2020
d.setFullYear(2016);     // ustawienie roku 2016
d.setDate(20);          //ustawienie 20 dnia miesica
```

- dodanie określonej liczby dni do daty:

```
var d = new Date();
document.write(d.toDateString() + "</br>");
d.setDate(d.getDate() + 100);
document.write(d.toDateString());
```

wynik:
Wed Dec 14 2016
Fri Mar 24 2017

Wyrażenia regularne

Wyrażenia regularne

Wyrażenie regularne (regular expression, regex) to sekwencja znaków definiujących pewien wzorzec, który chcemy odszukać w innym łańcuchu znaków.

- wyrażenia regularne rozpoczynają się i kończą znakiem ’/’, np:

```
var regexp = /kot/;
```

- wyrażenia regularne to w języku JavaScript obiekty klasy RexExp. Można je tworzyć na dwa sposoby:

```
var regexp = /kot/;
```

lub

```
var regexp = new RegExp("kot");
```

- zwykle wybieramy pierwszy (prostszy) sposób!

Wyrażenia regularne - metoda search()

- do wyszukiwania wzorca w łańcuchu znaków stosuje się metodę **search**()

```
var regexp = /kot/;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var pos = zdanie.search(regexp);
document.write(pos);
```

wynik: 7

- Przypomnienie: metodę search() można również wołać dla zmiennych typu string

```
var wyraz = "kot";
var zdanie = "Ala ma kota  i ten kot jest czarny";
var pos = zdanie.search(wyraz);
document.write(pos);
```

wynik: 7

Wyrażenia regularne - metoda replace()

- do zamiany wzorca w łańcuchu znaków na inny łańcuch służy metoda **replace**()

```
var regexp = /Ala/;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var text = zdanie.replace(regexp, "Dorota");
document.write(text);
```

wynik: Dorota ma kota i ten kot jest czarny

- Przypomnienie: metodę replace() można również wołać dla zmiennych typu string

```
var wyraz = "Ala";
var zdanie = "Ala ma kota  i ten kot jest czarny";
var text = zdanie.replace(wyraz, "Dorota");
document.write(text);
```

wynik: Dorota ma kota i ten kot jest czarny

Wyrażenia regularne - metoda match()

- do sprawdzenia, czy w łańcuchu znaków występuje fragment pasujący do wzorca, służy metoda **match**()
- metoda zwraca znaleziony fragment pasujący do wzorca

```
var regexp = /Ala/;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var text = zdanie.match(regexp);
document.write(text);
```

wynik: Ala

Obiekt RexExp - metoda test()

- do sprawdzenia, czy wzorzec występuje w łańcuchu znaków, służy metoda **test**()
- jeśli wzorzec występuje w łańcuchu znaków, metoda zwraca wartość **true,** w przeciwnym razie zwraca wartość **false**
- w przeciwieństwie do metod search() i replace(), metoda test() wołana jest na obiekcie reprezentującym wyrażenie regularne!

```
var regexp = /Ala/;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var czy_jest = regexp.test(zdanie);
document.write(czy_jest);
```

wynik: true

Obiekt RexExp - metoda exec()

- do sprawdzenia, czy w łańcuchu znaków występuje fragment pasujący do wzorca, służy metoda **exec**()
- metoda zwraca znaleziony fragment pasujący do wzorca
- metoda exec(), podobnie jak metoda test(), wołana jest na obiekcie reprezentującym wyrażenie regularne

```
var regexp = /Ala/;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var czy_jest = regexp.exec(zdanie);
document.write(czy_jest);
```

wynik: Ala

```
var regexp = /kot/g;
var zdanie = "Ala ma kota  i ten kot jest czarny";
var czy_jest = regexp.exec(zdanie);
document.write(czy_jest);
```

wynik: kot, kot

Wyrażenie regularne

Symbole używane w wyrażeniach regularnych

znak	pasuje do
.	dowolnego znaku
\w	dowolnego znaku używanego w słowach (litery a-z, A-Z, cyfry 0-9, podkreślenie)
\W	dowolnego znaku nie będącego znakiem używanego w słowach (przeciwieństwo znaku \w)
\d	dowolnej cyfry 0-9
\D	dowolnego znaku nie będącego cyfrą (przeciwieństwo znaku \d)
\s	znaku odstępu, tabulacji, powrotu karetki, nowego wiersza
\S	przeciwieństwo znaku \s

Wyrażenie regularne

Przykład:

```
var regexp = /\w\s\d\s\D\w\w/;
var zdanie = "Ala ma 9 kotów i tekoty s czarne";
var fragment = zdanie.match(regexp);
document.write(fragment);
```

wynik: a 9 kot

Wyrażenie regularne

Symbole używane w wyrażeniach regularnych

znak	pasuje do
^	początku łańcucha znaków
$	końca łańcucha znaków
\b	odstępu, początku łańcucha, końca łańcucha oraz dowolnego znaku nie będącego cyfrą ani literą
[ ]	dowolnego znaku podanego między nawiasami
[^]	dowolnego znaku z wyjątkiem podanych w nawiasach
|	znaku przed kreską pionową **lub** za nią
\	znaku, który jednocześnie jest znakiem specjalnym wyrażeń regularnych (np. \^)
()	ograniczenie zakresu

Wyrażenie regularne

Przykłady:

```
var regexp = /\b\w\w\w\w[!#@]$/;
var zdanie = "Alama9kotów i te koty mówi miau!";
var fragment = zdanie.match(regexp);
document.write(fragment);
```

wynik: miau!

```
var regexp = /(kot)|(lama)/;
var zdanie = "Alama9kotów i te koty mówi miau!";
var fragment = zdanie.search(regexp);
document.write(fragment);
```

wynik: 1

Wyrażenie regularne

Modyfikatory używane w wyrażeniach regularnych

znak	**znaczenie**
i	wyszukiwanie bez względu na wielkość liter
g	wyszukuje wszystkie wystąpienia wzorca w łańcuchu znaków
m	wyszukuje w wielu liniach tekstu

Przykład:

```
var zdanie = "Ala ma kota  i ten kot jest czarny";
var pos = zdanie.search(/ala/i);
var pos2 = zdanie.search(/ala/);
document.write(pos + "  " + pos2);
```

wynik: 0 -1

Wyrażenie regularne

Kwantyfikatory używane w wyrażeniach regularnych

znak	znaczenie
n+	jeden lub więcej znaków lub elementów n
n*	zero lub więcej znaków lub elementów n
n?	zero lub jeden znak lub element n
n{X}	sekwencja X znaków lub elementów n
n{X,Y}	sekwencja od X do Y znaków lub elementów n
n{X,}	sekwencja co najmniej X znaków lub elementów n

Elementem może być wyrażenie umieszczone wewnątrz nawiasów.

Kod pocztowy

Wyrażenie regularne odpowiadające polskiemu kodowi pocztowemu

```
/[\d]{2}-[\d]{3}/
```

- \d{2} - grupa dwóch cyfr
- \d{3} - grupa trzech cyfr

Przykład:

```
var codeReg = /[\d]{2}-[\d]{3}/;
var text = "Ala ma kota 80-288Gdask";
var pos = text.search(codeReg);    //12
var czy_jest = codeReg.test(text);    //true
var kod = codeReg.exec(text);      //80-288

text = "Ala ma kota 80-28Gdask";
pos = text.search(codeReg);      //-1
czy_jest = codeReg.test(text);     //false
kod = codeReg.exec(text);        //null
```

Kod pocztowy

Jak sprawdzić, czy użytkownik wpisał poprawny kod i tylko ten kod w polu formularza?

```
/^[\d]{2}-[\d]{3}$/
```

- \d{2} - grupa dwóch cyfr
- \d{3} - grupa trzech cyfr

Przykład:

```
var codeReg = /^[\d]{2}-[\d]{3}$/;
var text = "Ala ma kota 80-288Gdask";
var pos = text.search(codeReg);    //-1
var czy_jest = codeReg.test(text);    //false
var kod = codeReg.exec(text);      //null

text = "80-288";
pos = text.search(codeReg);      //0
czy_jest = codeReg.test(text);     //true
kod = codeReg.exec(text);       //80-288
```

Atomy

- Obejmując części wyrażenia regularnego w nawiasy tworzymy tzw. **atomy**.
- Kolejne atomy zawierają kolejne części znalezionego tekstu, do których można się indywidualnie odwołać.
- Przykład:

```
var regexp = /([\d]{2})-([\d]{3})-([\d]{4})/;
var text = "58-347-2004";
var kod = text.match(regexp);
for(i=0; i<kod.length; i++){
    document.write(kod[i] + '</br>');
}
```

Referencja wsteczna - odwołanie się do atomów

- Wykorzystanie obiektu RegExp:

```
var regexp = /([\d]{2})-([\d]{3})-([\d]{4})/;
var text = "58-347-2004";
var kod = text.match(regexp);
document.write(RegExp.$1 + '</br>');
document.write(RegExp.$2 + '</br>');
document.write(RegExp.$3 + '</br>');
```

wynik:
58
347
2004

Referencja wsteczna - odwołanie się do atomów

- Atomy w metodzie replace():

```
var regexp = /([\d]{2})-([\d]{3})-([\d]{4})/;
var text = "58-347-2004";
var textReplaced = text.replace(regexp, "$3 $2 $1");
document.write(textReplaced + '</br>');
```

wynik:
2004
347
58

Numer telefonu

Jak sprawdzić, czy użytkownik wpisał poprawny nr telefonu i tylko ten numer?

```
/^\(?([\d]{2})\)?[ -.]([\d]{3})[ -.]([\d]{4})$/
```

Przykład:

```
var codeReg = /^\(?([\d]{2})\)?[ -.]([\d]{3})[ -.]([\d]{4})$
    /;
var text = "58 347 2004";
var pos = text.search(codeReg);    //0
var czy_jest = codeReg.test(text);  //true
var kod = codeReg.exec(text);    //58 347 2004,58,347,2004
var kod2 = text.match(codeReg); //58 347 2004,58,347,2004
```

Adres e-mail

Jak sprawdzić, czy użytkownik wpisał poprawny adres e-mail?

```
/[-\w.]+@([A-z0-9][-A-z0-9]+\.)+[A-z]{2,4}/
```

Przykład:

```
var codeReg = /[-\w.]+@([A-z0-9][-A-z0-9]+\.)+[A-z]{2,4}/g;
var text = "iwona@gmail.com58 347 2004ul.Narutowicza ab@g.
    gom abb@gmail.com";
var pos = text.search(codeReg);    //0
var czy_jest = codeReg.test(text);  //true
var kod = codeReg.exec(text);    //abb@gmail.com,gmail.
var kod2 = text.match(codeReg);    //iwona@gmail.com,
    abb@gmail.com
document.write(pos + '</br>');
document.write(czy_jest + '</br>');
document.write(kod + '</br>');
document.write(kod2 + '</br>');
```

Adres strony www

Jak sprawdzić, czy użytkownik wpisał poprawny adres www?

```
/((\bhttps?:\/\/)|(\bwww\.))\S*/
```

Przykład:

```
var codeReg = /((\bhttps?:\/\/)|(\bwww\.))\S*/g;
var text = "iwona@gmail.com58 www.wp.pl ab@g.gom abb@gmail.
    com http://pg.edu.pl ";
var pos = text.search(codeReg);   //18
var czy_jest = codeReg.test(text);  //true
var kod = codeReg.exec(text);   //http://pg.edu.pl,http://,
    http://,
var kod2 = text.match(codeReg);   //www.wp.pl,http://pg.edu.
    pl
document.write(pos + '</br>');
document.write(czy_jest + '</br>');
document.write(kod + '</br>');
document.write(kod2 + '</br>');
```

Adres strony www - wyniki metody match()

```
var codeReg = /((\bhttps?:\/\/)|(\bwww\.))\S*/g;
var text = "iwona@gmail.com58 www.wp.pl ab@g.gom abb@gmail.
    com http://pg.edu.pl ";
var kod2 = text.match(codeReg);
for(i=0; i<kod2.length; i++){
  document.write(kod2[i] + '</br>');
}
```

wynik:
www.wp.pl
http://pg.edu.pl

Podział tekstu na słowa przy pomocy metody split()

```
var text = "Ala ma kota i ten kot jest czarny";
var textSplit = text.split(/\W+/);
for ( x=0; x<textSplit.length; x++) {
        document.write(textSplit[x]+"</br>");
}
```

wynik:
Ala
ma
kota
i
ten
kot
jest
czarny