



Интернет програмирање Предавање

функции.

Вовед

- Иако скриптните јазици најчесто овозможуваат пишување на мали програмски делови, сепак речиси е невозможно да се напише „потполна,, JavaScript програма без користење на функции.
- Како и во другите програмски јазици, функциите овозможуваат групирање на наредби во единствена целина.
- Бидејќи функциите се повикуваат по потреба со променлив број на параметри истите овозможуваат заштеда на програмерско време.
- Во JavaScript се комбинираат постоечки функции со кориснички дефинирани функции.
 - Постоечките функции се дефинирани во рамки на објекти кои нудат голем број на методи што извршуваат вообичаени математички пресметки, манипулација со стрингови, манипулација на време и датуми, и манипулација со колекции од податочни полиња.
 - Секој програмер може да дефинира сопствени функции кои извршуваат специфични задачи и да ги користат во многу точки во една скрипта
- Наредбите во функциите се пишуваат еднаш и се кријат од останатиот дел од наредбите во „програмата,,



Синтакса (начин 1) / декларирање



```
function name(arg1, ..., argN) { statements }
```



Функциите не дефинираат дали и каков вид на вредности враќаат.



повикување на функција

```
name(arg1, ..., argN)
```

Примери

```
function printprops(o){  
  for(var p in o)    console.log(p + ": " + o[p] + "\n");  
}
```

// Compute the distance between Cartesian points (x1,y1) and (x2,y2).

```
function distance(x1, y1, x2, y2) {  
  let dx = x2 - x1;  
  let dy = y2 - y1;  
  return Math.sqrt(dx*dx + dy*dy);  
}
```

Return израз

- Секоја функција може да врати вредност во кое и да е време со користење на наредбата `return`.

`return value;`

- пример:

```
function calc_sales(units_a, units_b, units_c) {  
    return units_a*79 + units_b*129 + units_c*699  
}
```



Return израз

- `return` изразот може да се користи и без да се специфицира вредноста што се враќа.
- Кога се користи на овој начин, функцијата го прекинува веднаш извршувањето и враќа вредност `undefined`.
- Ова обично се користи во функции кои не враќаат вредност да завршат пред да се извршат комплетно, како во следниот пример:

```
function sayHi(name, message) {  
    return;  
    alert("Hello " + name + ", " + message); //never called  
}
```
- Се препорачува функцијата или секогаш да враќа вредност или воопшто да не враќа вредност. Пишувањето на функции кои некогаш враќаат вредност може да предизвика конфузија особено при дебагирање на програмата.



Пример функции

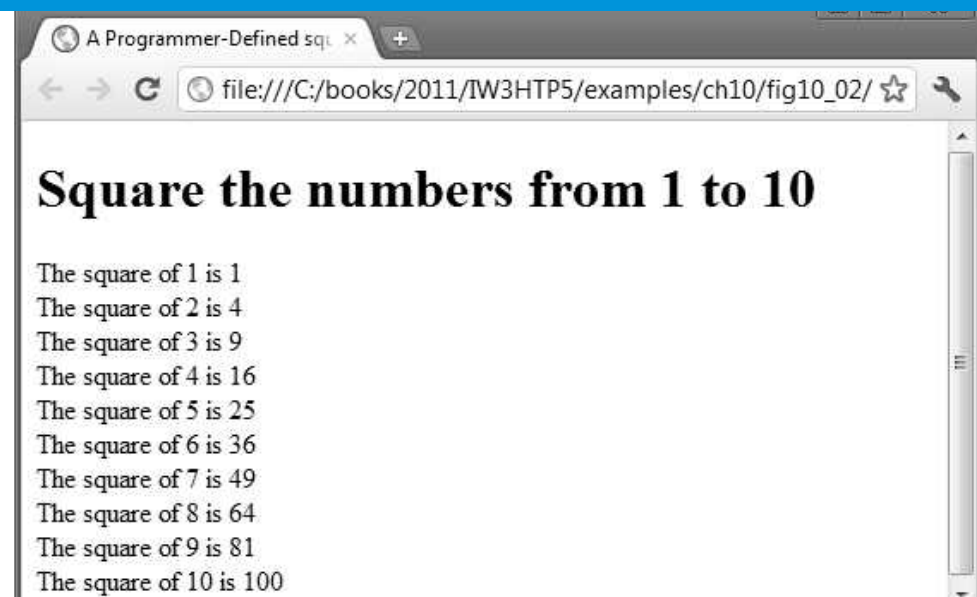
```
<html>
<head>
  <title>Function passing</title>
  <script language="Javascript" type="text/javascript">
    <!-- Hide script from older browsers
    function saySomething(message){
      alert(message)
    }
    // End hiding script from older browsers -->
  </script>
</head>
<body bgcolor="#FFFFFF">
<h2>Famous Presidential Quotes</h2>
<form action="#">
  <input type="button" value="Lincoln" onclick="saySomething('Four score and seven years ago...')"/>
  <input type="button" value="Kennedy" onclick="saySomething('Ask not what your country can do for you...')"/>
  <input type="button" value="Nixon" onclick="saySomething('I am not a crook!')"/>
</form>
</body>
</html>
```

Пример

```
<!DOCTYPE html>
<!-- Fig. 9.2: SquareInt.html -->
<!-- Programmer-defined function square. -->
<html> <head>
<meta charset = "utf-8">
<title>A Programmer-Defined square Function</title>
<style type = "text/css">
p { margin: 0; }
</style>
<script>

    document.writeln( "<h1>Square the numbers from 1 to 10</h1>" );
    // square the numbers from 1 to 10
    for( let x = 1; x <= 10; ++x )
        document.writeln( "<p>The square of " + x + " is " + square( x ) + "</p>" );
    // The following square function definition's body is executed
    // only when the function is called explicitly
    function square( y ){
        return y * y;
    } // end function

</script>
</head><body></body> <!-- empty body element -->
</html>
```



Пример

```
<!DOCTYPE html> <!-- Fig. 9.3: maximum.html -->
<!-- Programmer-Defined maximum function. --> <html>
<head>
<meta charset = "utf-8"> <title>Maximum of Three Values</title>
<style type = "text/css">
p { margin: 0; } </style>
<script>
```

```
function maximum( x, y, z ) { return Math.max( x, Math.max( y, z ) ); }
```

```
let input1 = window.prompt( "Enter first number", "0" );
```

```
let input2 = window.prompt( "Enter second number", "0" );
```

```
let input3 = window.prompt( "Enter third number", "0" );
```

```
let value1 = parseFloat( input1 );
```

```
let value2 = parseFloat( input2 );
```

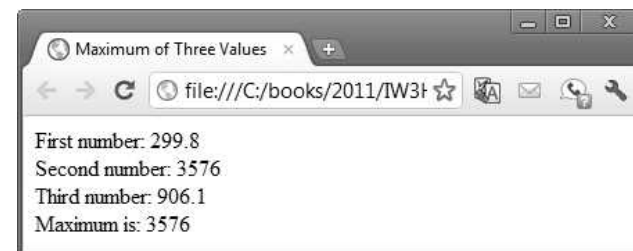
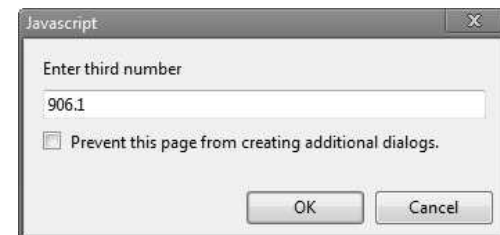
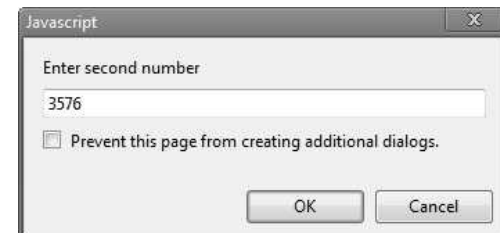
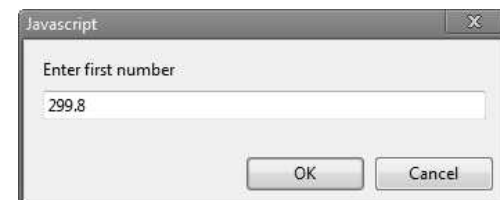
```
let value3 = parseFloat( input3 );
```

```
let maxValue = maximum( value1, value2, value3 );
```

```
document.writeln( "<p>First number: " + value1 + "</p>" + "<p>Second number: " + value2 + "</p>" + "<p>Third  
number: " + value3 + "</p>" + "<p>Maximum is: " + maxValue + "</p>" );
```

```
// maximum function definition (called from line 22)
```

```
</script> </head><body></body> </html>
```



Вгнездување на функции

```
function hypotenuse(a, b) {
    function square(x) { return x*x; }
    return Math.sqrt(square(a) + square(b));
}
```

- The interesting thing about nested functions is their variable scoping rules:
 - they can access the parameters and variables of the function (or functions) they are nested within.
 - In the code above, for example, the inner function `square()` can read and write the parameters `a` and `b` defined by the outer function `hypotenuse()`.

Поинаков начин на креирање на функции

- Втор начин на креирање на функција е со користење на функциски израз.
- Функцискиот израз има неколку форми.
- Највообичаената форма гласи:

```
let functionName = function(arg0, arg1, arg2){  
    //function body  
};
```

- Овој шаблон изгледа како доделување на вредност на променлива.
- Функцијата се креира и се доделува на променлива `functionName`.
- Креираната функција се смета дека е анонимна бидејќи нема идентификатор по клучниот збор за функцијата.
- Овие изрази се идентични на обичните изрази и мора да им биде доделена вредност пред да се користат.



примери

// This function expression defines a function that squares its argument.

// Note that we assign it to a variable

```
let square = function(x) { return x*x; }
```

// Function expressions can include names, which is useful for recursion.

```
let f = function fact(x) {
```

```
  if (x <= 1) return 1; else return x*fact(x-1);
```

```
};
```

// Function expressions can also be used as arguments to other functions:

```
data.sort(function(a,b) { return a-b; });
```

// Function expressions are sometimes defined and immediately invoked:

```
var tensquared = (function(x) { return x*x; })(10);
```

пример

```
// TypeError: functionOne is not a function
functionOne();

var functionOne = function() {
  console.log("Hello!");
};
```

⌚ Run code snippet

✖ Hide results

```
Error: {
  "message": "Uncaught TypeError: functionOne is not a function",
  "filename": "https://stacksnippets.net/js",
  "lineno": 14,
  "colno": 1
}
```

```
// Outputs: "Hello!"
functionTwo();

function functionTwo() {
  console.log("Hello!");
}
```

⌚ Run code snippet

✖ Hide results

🔗 [Full page](#)

Hello!

16:28:23.861

Arrow functions

- Трет начин на креирање на функција е со т.н. arrow functions.

- `let func = (arg1, arg2, ..., argN) => expression;`

- Пр.

`let sum = (a, b) => a + b;`

- Ова е исто со

```
let sum = function(a, b) {  
    return a + b;  
};
```

Arrow functions (2)

- Ако функцијата има само еден аргумент не мора да се стават загради на истиот.
Пр.
`let double = n => n * 2;`
- Ако функцијата нема аргументи, тогаш се ставаат празни загради.
Пр.
`let sayHi = () => alert("Hello!");`
- Доколку кодот на функцијата е комплексен и потребни се повеќе редови, тогаш кодот се пишува во големи загради и, доколку функцијата нешто враќа, мора да има `return` израз.
Пр.
`let sum = (a, b) => {
 let result = a + b;
 return result;
};`



new Function

- Функции може да се креираат и со new Function. Овој метод ретко се користи, но некогаш има ситуации каде динамички треба да се креира функција и нема друг начин.

```
let func = new Function ([arg1, arg2, ...argN],  
functionBody);
```

- Пр.

```
let sum = new Function('a', 'b', 'return a + b');
```

- Овој код креира функција sum која ги собира двата аргументи a и b.



Important

- As noted in function, function declaration statements are not true statements, and the ECMAScript specification only allows them as top-level statements.
- They can appear in global code, or within other functions, but they cannot appear inside of loops, conditionals, or try/catch/finally or with statements.
- Note that this restriction applies only to functions declared as statements.
- Function definition expressions may appear anywhere in your JavaScript code.



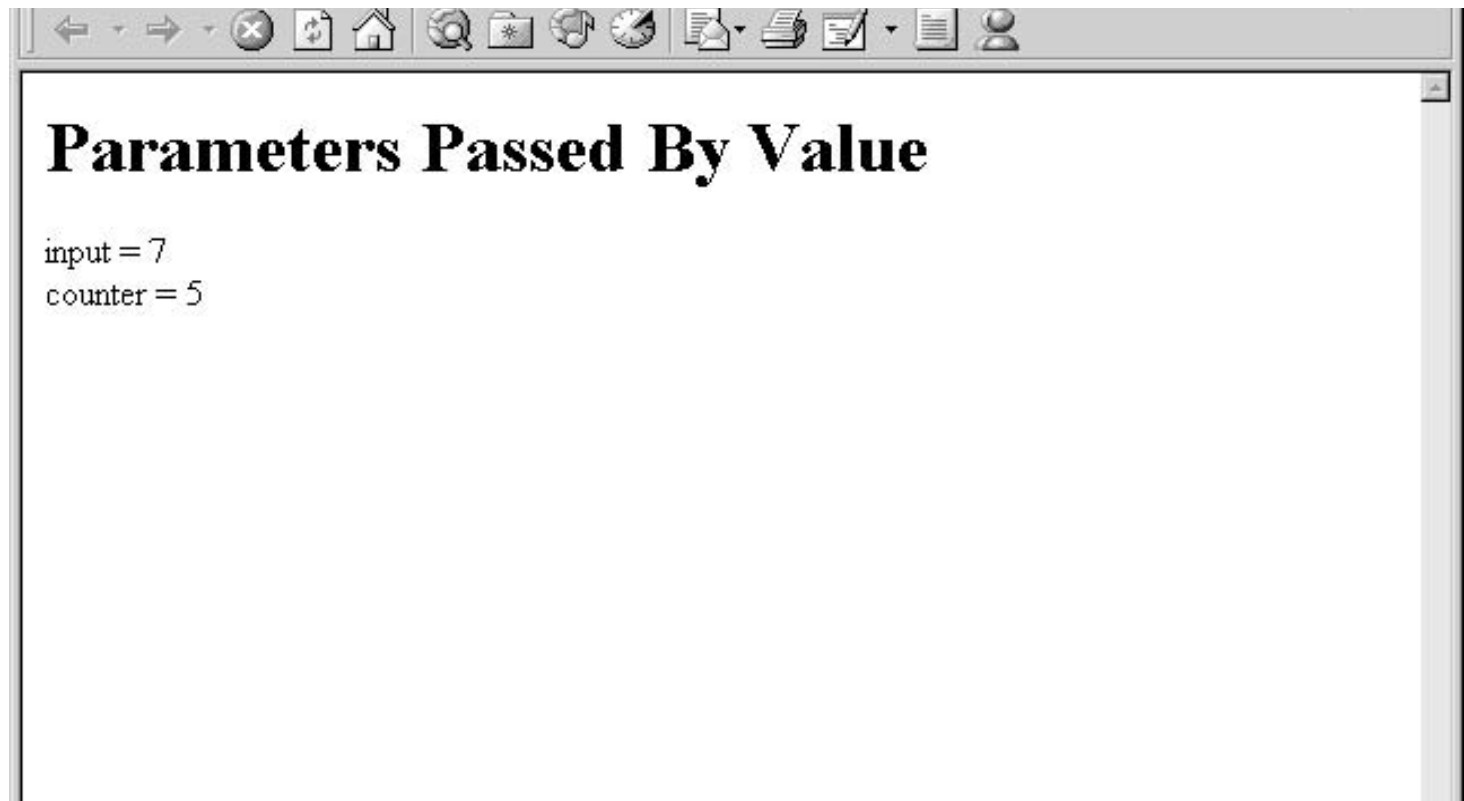
Влезни параметри

- Сите функциски аргументи се пренесуваат по вредност.
- Тоа значи дека вредноста надвор од функцијата се копира во аргументот од функцијата на ист начин како што се копира вредноста од една променлива во друга.
- Ова важи и за обичните променливи и за референците.

■ Пример:

```
var counter;  
counter = 5;  
function add2(input) {  
    input = input + 2;  
    document.write("input = " + input + "<br>");  
    return;  
}  
add2(counter);  
document.write("counter = " + counter + "<br>");
```

- Што ќе биде прикажано на компјутерскиот екран?
- Одговорот е дека променливата counter ќе има вредност 5 на крајот од програмата.
- JavaScript ги пренесува параметрите како вредност, што значи дека вредноста на аргументот не може да се смени перманентно во рамките на функцијата.



Број на влезни параметри

- Функциските аргументи не се однесуваат на ист начин како и кај другите програмски јазици.
- Во функциите е неважно колку аргументи се проследуваат ниту дали овие аргументи имаат соодветен вид.
- Само бидејќи е дефинирана функција која прима два аргумента не значи дека исто толку аргументи треба да бидат пренесени. Може да бидат обезбедени една, три или ни една вредност.
- Сите аргументи интерно се третираат како полиња, и затоа функцијата не води сметка што има во полето. До секој елемент во полето може да се пристапи со користење на стандардна нотација на полиња `arguments[0]`, `arguments[1]`, ИТН.

Пример:

```
function sayHi() {  
  alert("Hello " + arguments[0] + ", " + arguments[1]);  
}
```

- Во оваа верзија на функцијата нема именувани аргументи, но функцијата ќе се извршува нормално и без истите.
- Именуваните аргументи се погодност а не потреба.
- За функциите не се креира потпис што потоа мора да се согласува, и нема валидација за именуваните аргументи.

Област на важење (контекст)

- Атрибутите со кои се опишуваат променливите се име, вредност и податочен вид. Исто така имиња се дефинираат и за кориснички дефинираните функции. Секое име во една програма има област на важење.
- Областа на важење на идентификаторот за променлива или функција е дел од програмата во која истиот може да биде референциран. Областа на важење (контекст) на променливите во JavaScript е исто толку важна колку и кај другите програмски јазици, бидејќи дефинира кој има пристап до определена променлива или функција.
 - Глобалниот контекст е најнадворешниот (контекстот на прозорецот), и функциите и глобалните променливи се креираат како карактеристики на објектот прозорец (window).
 - Кога ќе се заврши извршување на даден контекст, тој се унишува и притоа се уништуваат сите променливи и функции дефинирани во истиот (глобалниот контекст важи се додека се извршува апликацијата, односно додека не се затвори веб страницата)
 - Секоја функција има свој контекст на извршување. При извршување на функцијата нејзиниот контекст се сместува со стекот со контексти. По завршување на функцијата, истиот се елиминира.

Пример

- Локално дефинирани променливи може наизменично да се користат со глобални променливи во иста област на важење (контекст).

- На пример:

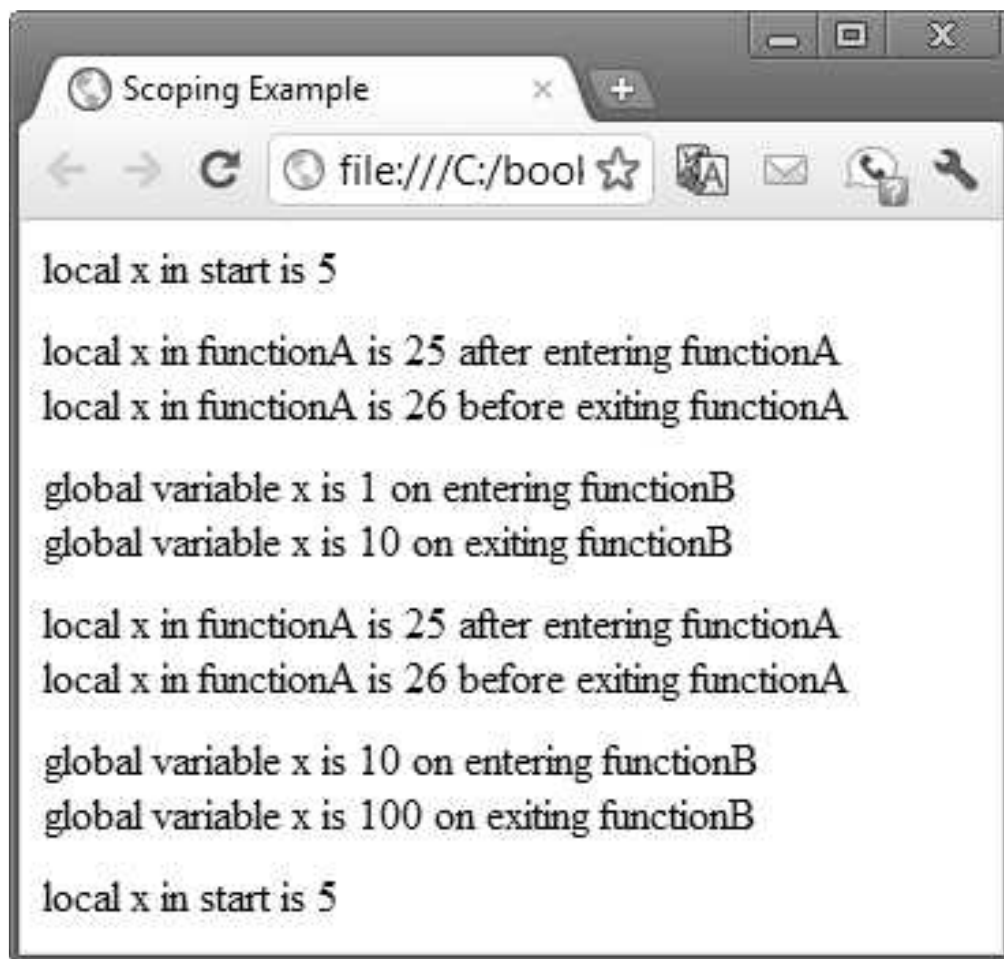
```
let color = "blue";
function changeColor(){
    let anotherColor = "red";
    function swapColors(){
        let tempColor = anotherColor; anotherColor = color;
        color = tempColor;
        //color, anotherColor, and tempColor are all accessible
here
    }
    //color and anotherColor are accessible here, but not tempColor
    swapColors();
}
```



```
<!DOCTYPE html> <!-- Fig. 9.9: scoping.html --> <!-- Scoping example. --> <html>
<head> <meta charset = "utf-8">
<title>Scoping Example</title>
  p { margin: 0px; }
<style type = "text/css"> p.space { margin-top: 10px; } </style>
<script>
let output; // stores the string to display
let x = 1; // global variable
function start() {
    let x = 5; // variable local to function start
    output = "<p>local x in start is " + x + "</p>";
    functionA(); // functionA has local x
    functionB(); // functionB uses global variable x
    functionA(); // functionA reinitializes local x
    functionB(); // global variable x retains its value
    output += "<p class='space'>local x in start is " + x + "</p>";
    document.getElementById( "results" ).innerHTML = output;
} // end function start
```



```
function functionA() {  
    let x = 25; // initialized each time functionA is called  
    output += "<p class='space'>local x in functionA is " + x + " after  
    entering functionA</p>";  
    ++x;  
    output += "<p>local x in functionA is "+x+" before exiting functionA</p>";  
} // end functionA  
  
function functionB(){  
    output += "<p class='space'>global variable x is " + x + " on entering  
    functionB";  
    x *= 10;  
    output += "<p>global variable x is " + x + " on exiting functionB</p>";  
} // end functionB  
  
window.addEventListener( "load", start, false );  
</script>  
</head> <body> <div id = "results"></div> </body>  
</html>
```

Scoping Example

file:///C:/bool

local x in start is 5

local x in functionA is 25 after entering functionA

local x in functionA is 26 before exiting functionA

global variable x is 1 on entering functionB

global variable x is 10 on exiting functionB

local x in functionA is 25 after entering functionA

local x in functionA is 26 before exiting functionA

global variable x is 10 on entering functionB

global variable x is 100 on exiting functionB

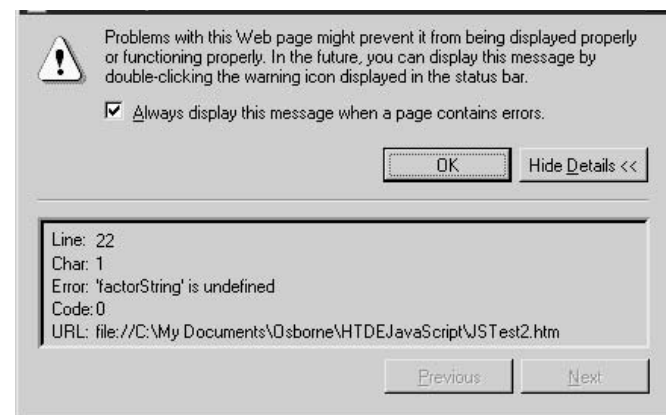
local x in start is 5

Област на важење

- Сите променливи декларирани со клучниот збор `let` во една функција се локални променливи – тие може да се користат и да им се пристапи само во рамките на функцијата во која се дефинирани.
- Функциските параметри се исто така локални променливи.
- Кога ќе се напушти функцијата, променливата не е повеќе дефинирана и нејзината вредност се губи.

```
// get_factors() returns all the numbers that divide into
// the parameter evenly
```

```
function get_factors(inputValue) {
  let counter; let factorString = "";
  for (counter = 1; counter <= inputValue; counter++)
    { if ((inputValue % counter) == 0)
      { factorString = factorString + counter + " "; } }
}
get_factors(6777214);
document.write (factorString);
```



Преоптоварување

- Функциите не може да бидат преоптоварени во традиционална смисла.
- Во другите програмски јазици може да се употреби исто име за две функции се додека функциите имаат различен прототип (видот и бројот на параметри е различен).
- Бидејќи овде функциите немаат прототипови, вистинско преоптоварување не е можно.
- Ако две функции имаат исто име тогаш последно дефинираната функција станува сопственик на името.
- На пример:

```
function addSomeNumber(num){ return num + 100; }  
function addSomeNumber(num){ return num + 200; }  
let result = addSomeNumber(100); //300
```
- Функцијата `addSomeNumber()` е дефинирана двапати. При извршување на последната наредба вратената вредност ќе биде 300 бидејќи втората функција ја препокрива првата.

Прифаќање на произволен број на параметри

- Функциите може да бидат дефинирани да примаат неограничен број на параметри. Ова е потребно кога сакаме функциите да примат голем број параметри од ист вид но не сакаме да користиме многу имиња за опционални параметри.
- Остатокот од параметрите треба секогаш да се појават на крајот на кои и да се неименувани параметри со користење на параметарот остаток (...). Именуваните параметри, дефинирани со именуван идентификатори, се уште може да се користат по овој параметар. Се разбира може да има само еден параметар остаток.

Функции со произволен број параметри

```
function printform (name, address, phoneNum , spouseName, ... childName ) {  
  // An unlimited number of children can be provided  
  with (document) {  
    write ("Name: " + name + "<br>");  
    write ("Spouse: " + spouseName + "<br>");  
    write ("Address: " + address + "<br>");  
    write ("Phone: " + phoneNum + "<br>");  
    for (child in childName) {  
      write ("Child " + childName[child] + "<br>");  
    }  
  }  
  return;  
}
```



Стандарни функции во JavaScript

Global function	Description
<code>isFinite</code>	Takes a numeric argument and returns true if the value of the argument is not NaN, <code>Number.POSITIVE_INFINITY</code> or <code>Number.NEGATIVE_INFINITY</code> (values that are not numbers or numbers outside the range that JavaScript supports)—otherwise, the function returns false.
<code>isNaN</code>	Takes a numeric argument and returns true if the value of the argument is not a number; otherwise, it returns false. The function is commonly used with the return value of <code>parseInt</code> or <code>parseFloat</code> to determine whether the result is a proper numeric value.
<code>parseFloat</code>	Takes a string argument and attempts to convert the <i>beginning</i> of the string into a floating-point value. If the conversion is unsuccessful, the function returns NaN; otherwise, it returns the converted value (e.g., <code>parseFloat("abc123.45")</code> returns NaN, and <code>parseFloat("123.45abc")</code> returns the value 123.45).
<code>parseInt</code>	Takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns NaN; otherwise, it returns the converted value (for example, <code>parseInt("abc123")</code> returns NaN, and <code>parseInt("123abc")</code> returns the integer value 123). This function takes an optional second argument, from 2 to 36, specifying the radix (or base) of the number. Base 2 indicates that the first argument string is in binary format, base 8 that it's in octal format and base 16 that it's in hexadecimal format. See Appendix E, for more information on binary, octal and hexadecimal numbers.

Функции со текстуални низи

<code>charAt (<i>index</i>)</code>	Returns one character from the string
<code>charCodeAt (<i>index</i>)</code>	Returns the ASCII code of one character from the string
<code>concat (<i>string</i>)</code>	Joins two strings together
<code>indexOf (<i>string</i>, <i>index</i>)</code>	Finds the first occurrence of one string inside another
<code>lastIndexOf (<i>string</i>, <i>index</i>)</code>	Finds the last occurrence of one string inside another
<code>match (<i>regexp</i>)</code>	Searches the string for the regular expression pattern, and returns any matches
<code>replace (<i>regexp</i>, <i>newString</i>)</code>	Searches the string for the regular expression pattern, and replaces any matches with the string provided
<code>search (<i>regexp</i>)</code>	Searches the string for the regular expression pattern, and returns the index of the first match
<code>split (<i>separator</i>, <i>limit</i>)</code>	Splits a single string into an array of substrings, based on a separator character or regular expression
<code>substring (<i>start</i>, <i>end</i>)</code>	Returns a part of a string
<code>toLowerCase ()</code>	Returns the string as lowercase characters
<code>toString ()</code>	Same as <code>valueOf ()</code> ; returns the string
<code>toUpperCase ()</code>	Returns the string as uppercase characters

Примери

```
var msg = "ERROR: Invalid User ID or Password<br>";
document.write ( "<b>Original message:</b> " + msg.valueOf() );
document.write ( "<b>Upper case:</b> " + msg.toUpperCase() );
document.write ( "<b>Lower case:</b> " + msg.toLowerCase() );
document.write ( "<b>Partial string:</b> " + msg.substring(20, 22) + "<br>" );
document.write ( "<b>Single character:</b> " + msg.charAt(17) + "<br>" );
document.write ( "<b>Search from beginning:</b> " + msg.toLowerCase().indexOf("sword") +
"<br>" );
document.write ( "<b>Search from end:</b> " + msg.lastIndexOf("User ID") + "<br>" );
```



Математички функции

Method	Purpose
abs (x)	The absolute value of x
acos (x), asin (x), atan (x), atan2 (y, x), cos (x), sin (x), tan (x)	Trigonometry functions
ceil (x)	Returns the next largest integer larger than x (or x, if it is an integer)
exp (x)	Returns the constant E to the power of x
floor (x)	Returns the next lower integer less than x (or x, if it is an integer)
log (x)	Returns the natural logarithm of x
max (x, y)	Returns x or y, whichever is highest
min (x, y)	Returns x or y, whichever is lowest
pow (x, y)	Returns x to the power of y

Method	Purpose
random ()	Returns a pseudorandom number between 0 and 1
round (x)	Rounds x to the nearest integer
sqrt (x)	Returns the square root of x



<!DOCTYPE html> 2

<!-- Fig. 9.4: RandomInt.html -->

<!-- Random integers, shifting and scaling. -->

<html>

<head>

<meta charset = "utf-8">

<title>Shifted and Scaled Random Integers</title> <style type = "text/css">

p, ol { margin: 0; }

li { display: inline; margin-right: 10px; } </style>

<script>

let value;

document.writeln("<p>Random Numbers</p>");

for (let i = 1; i <= 30; ++i) {

value = Math.floor(1 + Math.random() * 6); document.writeln("" + value + "");

} // end for

document.writeln("");

</script> </head><body></body> </html>

Функции за датуми и време

- Може да се одделуваат делови од датум
- Да се менуваат вредностите на делови од датум
- Да се претвори датумот во различен текстуален формат
- Да се претвори датумот од една во друга временска зона
- Раните верзии на JavaScript го ограничуваа објектот Date да не може да се справува со датуми пред January 1, 1970.
- Ова е поправено во верзијата 1.3 на јазикот.
- Датумите може да бидат во опсег -100 милион и +100 милион денови од January 1, 1970, што е апроксимативно 270,000 години во иднина и минато

Дел од функциите во објектот Date

Method	Purpose
<code>getDate ()</code>	Returns the day of the month (1–31) according to local time
<code>getDay ()</code>	Returns the day of the week (1–7) according to local time
<code>getFullYear ()</code>	Returns the four-digit year according to local time
<code>getHours ()</code>	Returns the hour component of the time according to local time
<code>getMilliseconds ()</code>	Returns the milliseconds component of the time according to local time
<code>getMinutes ()</code>	Returns the minutes component of the time according to local time
<code>getMonth ()</code>	Returns the month (0–11) according to local time; 0 is January, 1 is February, and so on
<code>getSeconds ()</code>	Returns the seconds component of the time according to local time



Introduction: callbacks

- Many actions in Javascript are *asynchronous*
- A callback is, essentially, a function that gets called when another function finishes
- A function that does something *asynchronously* should provide a *callback* argument where we put the function to run after it's complete.

- For instance, take a look at the function `loadScript(src)`:

```
function loadScript(src) {
  let script = document.createElement('script');
  script.src = src;
  document.head.append(script);
}
```

- The purpose of the function is to load a new script. When it adds the `<script src="...">` to the document, the browser loads and executes it.



- We can use it like this:

```
// loads and executes the script
loadScript('/my/script.js');
// the code below loadScript doesn't wait for the script loading to finish
// ...
```

- The function is called “asynchronously”, because the action (script loading) finishes not now, but later.
- The call initiates the script loading, then the execution continues.
- While the script is loading, the code below may finish executing, and if the loading takes time, other scripts may run meanwhile too.

lo



- Now let's say we want to use the new script when it loads.
- It probably declares new functions, so we'd like to run them.
- But if we do that immediately after the `loadScript(...)` call, that wouldn't work:

```
loadScript('/my/script.js'); // the script has "function newFunction(){...}"
```

```
newFunction(); // no such function!
```

- Naturally, the browser probably didn't have time to load the script.
- So the immediate call to the new function fails.
- As of now, `loadScript` function doesn't provide a way to track the load completion.
- The script loads and eventually runs, that's all.
- But we'd like to know when it happens, to use new functions and variables from that script.



- Let's add a callback function as a second argument to loadScript that should execute when the script loads:

```
function loadScript(src, callback){  
    let script = document.createElement('script');  
    script.src = src;  
    script.onload = () => callback(script);  
    document.head.append(script);  
}
```

- Now if we want to call new functions from the script, we should write that in the callback:

```
loadScript('/my/script.js', function() {  
    // the callback runs after the script is loaded  
    newFunction(); // so now it works  
    ...  
});
```

- That's the idea: the second argument is a function (usually anonymous) that runs when the action is completed.

Функции

пример

Convert temperature

Fahrenheit Celsius

```
<html>
<head>
<title>Temperature conversion</title>
<script type="text/javascript">
function convTempF2C(tempInF) {
    return 5 / 9 * (tempInF - 32);
}
function convTempC2F(tempInC) {
    return 9 * tempInC / 5 + 32;
}
</script>
</head>
<body>
<h2>Convert temperature</h2>
<form>
    <h3>Fahrenheit <input name="tempF" type="text"
        onkeyup="tempC.value = convTempF2C(this.value);">
        Celsius <input name="tempC" type="text"
            onkeyup="tempF.value = convTempC2F(this.value);">
    </h3>
</form>
</body>
</html>
```

функција convTempF2C

функција convTempC2F

Повикај ф-ја
convTempF2C

Повикај ф-ја
convTempC2F

Клучен збор
this

Малку повеќе за HTML – форми

About yourself

Name:

Age:

Gender: ☐ Male ☐ Female

Your favourite colour:

Your hobbies:

- ☐ Jogging
- ☐ Football
- ☐ Reading

Текстуални полиња (Textual fields) points to the Name and Age input fields.

Радио копчиња (Radio buttons) points to the Gender radio buttons.

Селекција од листа (List selection) points to the dropdown menu for favourite colour.

Check поле (Check field) points to the checkboxes for hobbies.

Копче (Button) points to the Submit button.

HTML – форма

```
<h1>About yourself</h1>
<form name="exampleform">
  <h3>Name: <input name="yourname" type="text"
    onchange='check(this);'></h3>
  <h3>Age: <input name="age" type="text"
    onchange='check(this);'>
</h3> <h3>Gender:
  <input name="gender" type="radio" value="male"> Male
  <input name="gender" type="radio" value="female"> Female
</h3>

<h3>Your favourite colour:
  <select name="colours">
    <option value="None">
    <option value="red">Red
    <option value="green">Green
    <option value="blue">Blue
    <option value="pink">Pink
  </select>
</h3>
```

име

вид

Текстуално
поле

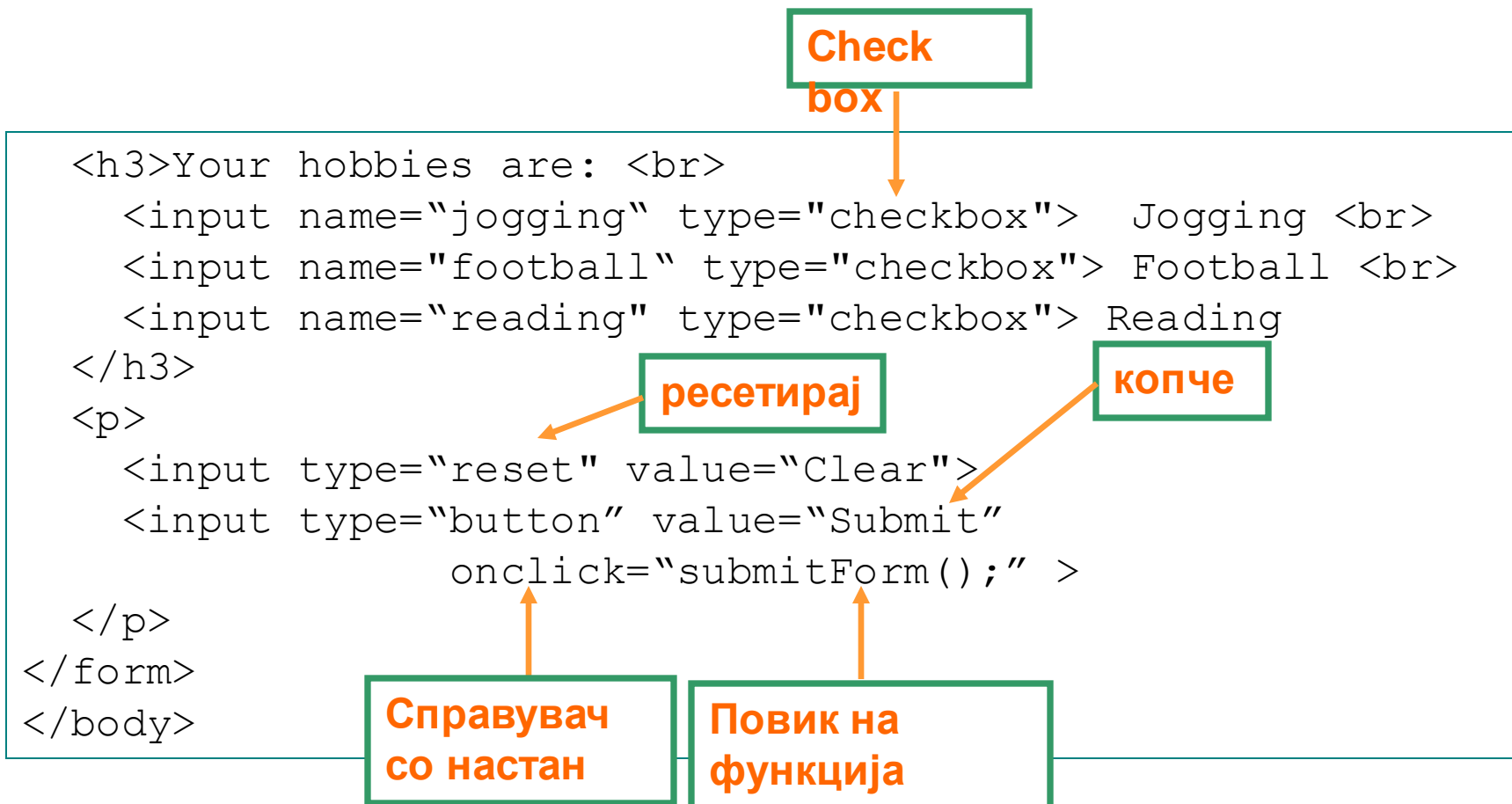
Валидирај
кориснички
внес

Радио копче

Избор од
листа

Клучен збор this се
однесува на тековниот
објект

HTML форма



Конечно функции за валидации

```
function check(obj) {
  let name = obj.name;
  let value = obj.value;
```

**Функција
check**

```
  switch (name) {
    case "yourname":
      if ( ! isValidName(value) ) {
        alert(value + " is not a valid name.");
        obj.value = "";
      }
      break;
    case "age":
      if ( !(value > 0 && value < 150) ) {
        alert(value + " is not a valid age.");
        obj.value = "";
      }
      break;
```

ФУНКЦИСКИ ПОВИК

**Реши
проблем на
погрешно
име**

**Реши
проблем на
погрешни
години**

Конечно употреба на JavaScript

функција submitForm

Локални
променливи

```
function submitForm() {  
    let name = document.exampleform.yourname.value;  
    let age = document.exampleform.age.value;  
    let gender = getGender();  
    let colour = document.exampleform.colours.value;  
    let hobbies = getHobby();  
  
    let msg = "<h2>Thank you!</h2>";  
    msg += "<h4>Name: " + name + "<br>";  
    msg += "Age: " + age + "<br>";  
    msg += "Favourite colour: " + colour + "<br>";  
    msg += "Hobbies: " + hobbies + "</h4>";  
  
    displayMessage(msg);  
}
```

Функции
повик

Функциски
повик

Функциски
повик



И уште JavaScript функции

Ф-ја
getGende

р

```
function getGender() {  
    let gender = document.exampleform.gender;  
    if (gender[0].checked) {  
        return "Male";  
    } else {  
        return "Female";  
    }  
}
```

Ф-ја
getHobb

у

```
function getHobby() {  
    let hobbies = "";  
    let obj = document.exampleform  
    if ( obj.jogging.checked ) hobbies += "Jogging ";  
    if ( obj.football.checked ) hobbies += "Football ";  
    if ( obj.reading.checked ) hobbies += "Reading ";  
    if ( hobbies.length == 0 ) hobbies = "None";  
    return hobbies;  
}
```


И уште една JavaScript функција

Функција
displayMessage
е

```
function displayMessage(msg) {  
    // open a new window  
    let msgWindow = window.open('', 'Message');  
  
    // write msg  
    msgWindow.document.write(msg);  
  
    // raise this window, in case it's not visible  
    msgWindow.focus();  
}
```

И за крај ...


Функција isValidName

```
function isValidName(str) {  
  let invalid = /^[a-zA-Z]+$/;  
  return invalid.test(str);  
}
```

не



Објект на
RegExp
(регуларен
израз)



Регуларни изрази

- Како да се провери структура на текст со малку команди
- <http://www.javascriptkit.com/javatutors/re.shtml>
- <http://www.javascriptkit.com/javatutors/redev2.shtml>
- <http://www.regular-expressions.info/javascript.html>
- <http://www.diveintojavascript.com/articles/javascript-regular-expressions>
- <http://www.javascripter.net/faq/creatingregularexpressions.htm>