Universiteti i Prishtinës "Hasan Prishtina" Kosovë

Programimi në www

Pjesa 10 – JS (2)

Prof. Asoc. Dr. Ermir Rogova

Lecture content

- In this lecture we will cover the following:
  - Functions
  - Scope of variables
  - Loops

Intro to functions

- A function is a block of code that will be executed when "someone" calls it

- Example:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("Hello World!");
}
</script>
</head>

<body>
<button onclick="myFunction()">Try it</button>
</body>
</html>
```

Function syntax

- In JavaScript, a function is written as a code block (inside curly { } braces), preceded by the **function** keyword:

- function *functionname*()
  {
  some code to be executed
  }

- The function can be called directly when an event occurs (like when a user clicks a button), and it can be called from "anywhere" by JavaScript code.

Function syntax specifics

- JavaScript is case sensitive. The function keyword must be written in lowercase letters, and the function must be called with the same capitals as used in the function name.

- Examples:

  ~~Function AddTwoNumbers()~~

  function AddTwoNumbers()

- When called:

  <button onclick="~~Addtwonumbers(~~)">Try it</button>

  <button onclick="~~ADDTWONUMBERS(~~)">Try it</button>

  <button onclick="AddTwoNumbers()">Try it</button>

Calling a function with arguments

- When we call a function, we can pass along some values to it, these values are called *arguments* or *parameters*.
- These arguments can be used inside the function.
- We can send as many arguments as we like, separated by commas (,)

myFunction(*argument1,argument2*)

Declaring a function with arguments

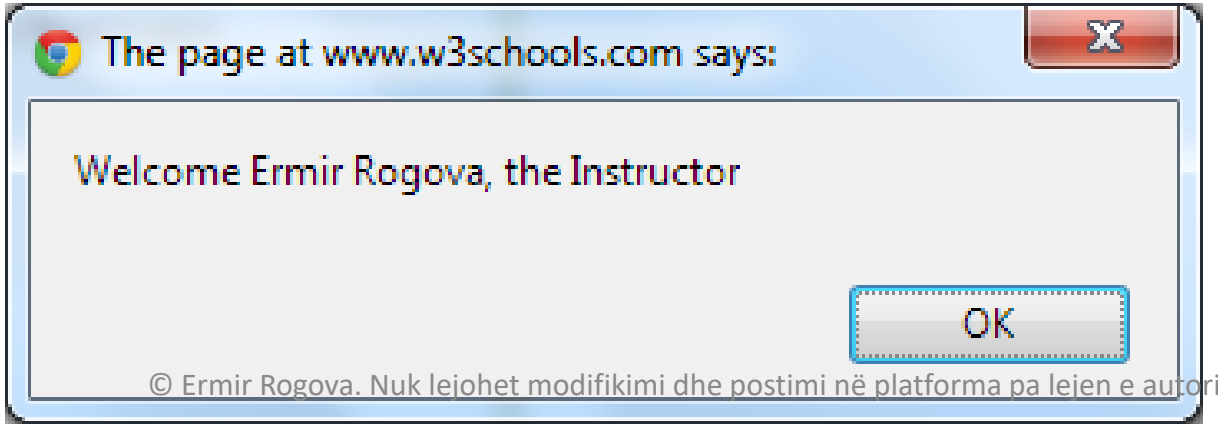
- If we plan to pass along arguments to a function, we must make sure that we set it properly
- We must declare the arguments, as variables, when we declare the function:
  - function myFunction(*var1*,*var2*)
    {
    some code
    }
- The variables and the arguments must be in the expected order. The first variable is given the value of the first passed argument etc.

Passing arguments to a function

- Example:

```
<button onclick="myFunction('Ermir Rogova','Instructor')">Try it</button>

<script>
function myFunction(name,job)
{
alert("Welcome
}
</script>
```



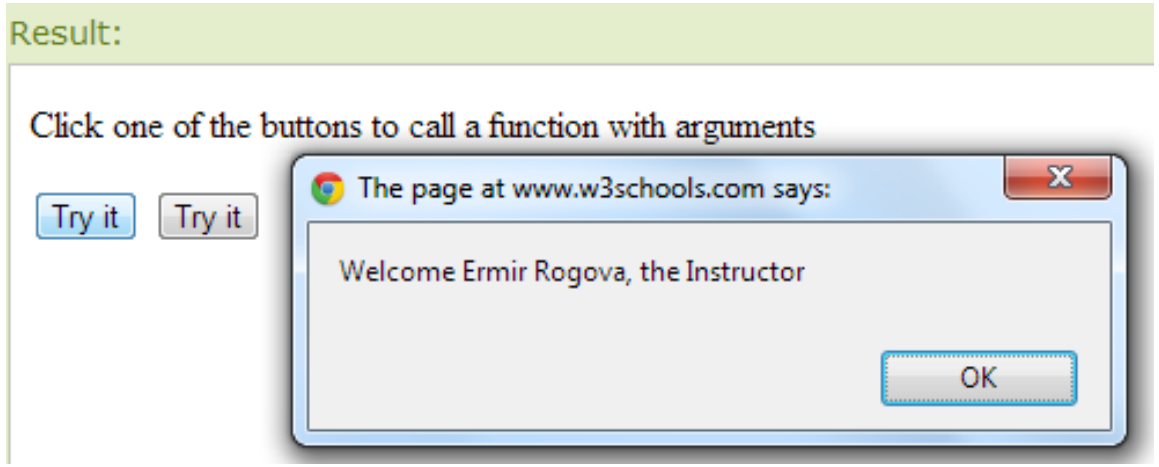
Reuse of functions

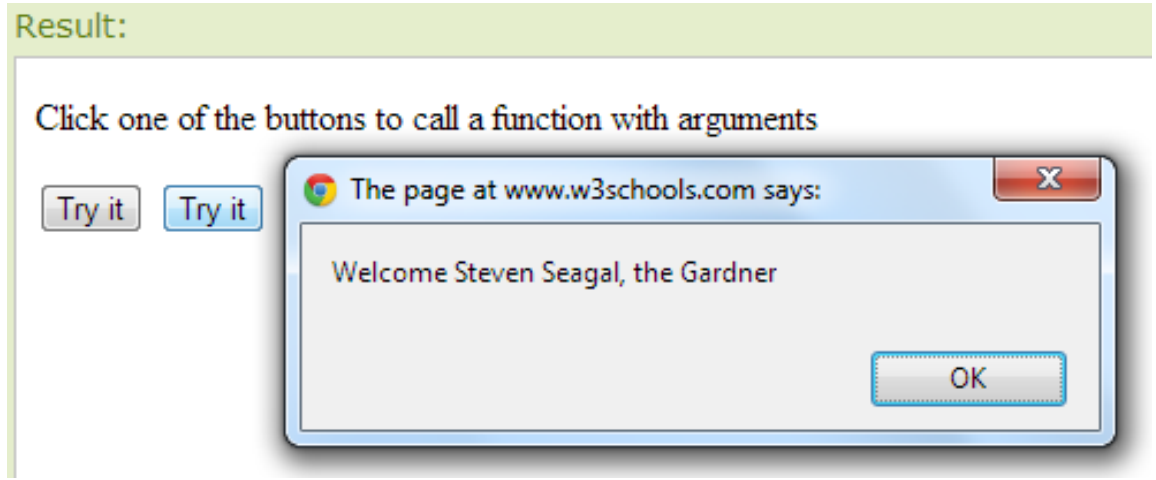
- A function is flexible, we can call the function using different arguments, and different welcome messages will be given:

```
<button onclick="DisplayData('Ermir Rogova', 'Instructor')">Try it</button>
<button onclick="DisplayData('Steven Seagal', 'Gardner')">Try it</button>

<script>
function DisplayData(name,job)
{
alert("Welcome " + name + ", the " + job);
}
</script>
```

Reuse of functions - example





Functions With a Return Value (1)

- Sometimes we want our function to return a value back to where the call was made.
- This is possible by using the *return* statement.
- When using the *return* statement, the function will stop executing, and return the specified value.
- It is not the entire JavaScript that will stop executing, only the function. JavaScript will continue executing code, where the function-call was made from.

Functions With a Return Value (2)

- Example:

```
function myFunction()
{
var x=5;
return x;
}
```

- The function above will return the value 5.
- var myNumber=myFunction()
- The variable myNumber holds the value 5, which is what the function "myFunction()" returns.

Functions With a Return Value (3)

- We can also use the return value without storing it as a variable:

```
document.getElementById("demo").innerHTML=myFunction();
```

- The innerHTML of the "demo" element will be 5, which is what the function "myFunction()" returns.

- We can also make a return value based on arguments passed into the function

Functions With a Return Value (4)

Example

Calculate the product of two numbers, and return the result:

```
function myFunction(a,b)
{
return a*b;
}

document.getElementById("demo").innerHTML=myFunction(4,3);
```

The innerHTML of the "demo" element will be:

```
12
```

Exiting a function

- The return statement is also used when we simply want to exit a function.
- The return *value* is optional:

```
function myFunction(a,b)
{
if (a>b)
  {
  return;
  }
x=a+b
}
```

- The function above will exit the function if a>b, and will not calculate the sum of a and b.

JavaScript variable scope

- A variable declared (using var) within a JavaScript function becomes **LOCAL** and can only be accessed from within that function. (the variable has local scope).
- We can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
- Local variables are deleted as soon as the function is completed.
- Variables declared outside a function, become **GLOBAL**, and all scripts and functions on the web page can access it.

The lifetime of JavaScript Variables

- The lifetime JavaScript variables starts when they are declared.
- Local variables are deleted when the function is completed.
- Global variables are deleted when we close the page.

Undeclared JavaScript Variables

- If we assign a value to variable that has not yet been declared, the variable will automatically be declared as a **GLOBAL** variable.
- Example:
  - carname="BMW"
- The above statement will declare the variable *carname* as a global variable , even if it is executed inside a function.

Intro to loops

- Image we have to repeat a piece of code 100 times.
- We can either copy and paste the code - very unprofessional, ugly and redundant
- Or we could use loops ☺
- The looping statements are those that bend that path back upon itself to repeat portions of our code.

Intro to loops - example

- ```
  document.write(cars[0] + "<br>");
  document.write(cars[1] + "<br>");
  document.write(cars[2] + "<br>");
  document.write(cars[3] + "<br>");
  document.write(cars[4] + "<br>");
  document.write(cars[5] + "<br>");
  ```

- ```
  for (var i=0;i<6;i++)
  {
  document.write(cars[i] + "<br>");
  }
  ```

Types of loops

- One common use for loops is to iterate over the elements of an array.

- JavaScript has four looping statements:
  - for
  - for/in
  - while
  - do/while

The For Loop

- The for loop is often the tool we will use when we want to create a loop.
- The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3)
 {
 the code block to be executed
 }
```

  - **Statement 1** is executed before the loop (the code block) starts.
  - **Statement 2** defines the condition for running the loop (the code block).
  - **Statement 3** is executed each time after the loop (the code block) has been executed
- Statements 1, 2 and 3 are also called initialize, test, and increment

The For Loop - Example

- for (var i=0; i<5; i++)
  {
  x=x + "The number is " + i + "
";
  }

- From the example above, we can read:
  - Statement 1 sets a variable before the loop starts (var i=0).
  - Statement 2 defines the condition for the loop to run (i must be less than 5).
  - Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Statement 1 (Initialize)

- Normally we will use statement 1 to initiate the variable used in the loop (var i=0).
- This is not always the case, JavaScript doesn't care, and statement 1 is optional.
- We can initiate any (or many) values in statement 1

Statement 1 - examples

- for (var i=0,j=cars.length; i<j; i++)
  {
  document.write(cars[i] + "
");
  }

- var i=2,j=cars.length;
  for (; i<j; i++)
  {
  document.write(cars[i] + "
");
  }

Statement 2 (test)

- Often statement 2 is used to evaluate the condition of the initial variable.
- This is not always the case, JavaScript doesn't care, and statement 2 is optional.
- If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.
- If you omit statement 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash the browser.

Statement 3 (increment)

- Often statement 3 increases the initial variable.
- This is not always the case, JavaScript doesn't care, and statement 3 is optional.
- Statement 3 could do anything. The increment could be negative (i--), or larger (i=i+15).
- Statement 3 can also be omitted (like when you have corresponding code inside the loop)

Statement 3 – example

- ```
  var i=0,j=cars.length;
  for (; i<j; )
  {
  document.write(cars[i] + "<br>");
  i++;
  }
  ```
  ```

The For/in loop

- The JavaScript for/in statement loops through the properties of an object

- var person={fname:"John",lname:"Doe",age:25};

```
for (x in person)
  {
  txt=txt + " " + person[x];
  }
```

- Displays: John Doe 25

The While loop

- The while loop loops through a block of code as long as a specified condition is true.

- Syntax:

```
while (condition)
 {
 code block to be executed
 }
```

The While loop - example

- while (i<5)
  {
  x=x + "The number is " + i + "
";
  i++;
  }

- The loop in this example will continue to run as long as the variable i is less than 5

- If we forget to increase the variable used in the condition, the loop will never end. This will crash the browser.

The Do/While loop

- The do/while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax:
  - do
    {
    code block to be executed
    }
    while (*condition*);

Do/While - example

- do
  {
  x=x + "The number is " + i + "
";
  i++;
  }
  while (i<5);

- The example above uses a do/while loop.

- The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested

Comparing For and While

- If we pay close attention, we will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

- ```
  Cars=["BMW","Volvo","Saab","Ford"];
  var i=0;
  for (;cars[i];)
  {
  document.write(cars[i] + "<br>");
  i++;
  }
  ```

- ```
  cars=["BMW","Volvo","Saab","Ford"];
  var i=0;
  while (cars[i])
  {
  document.write(cars[i] + "<br>");
  i++;
  }
  ```

Break

- The break statement, used alone, causes the innermost enclosing loop or switch statement to exit immediately.
- Syntax:
  - break;
- Because it causes a loop or switch to exit, in this form, the break statement is legal only if it appears inside one of these statements.
- It is typically used to exit prematurely when, for whatever reason, there is no longer any need to complete the loop.
- When a loop has complex termination conditions, it is often easier to implement some of these conditions with break statements rather than trying to express them all in a single loop expression.

Break - example

- The following code searches the elements of an array for a particular value.

- The loop terminates in the normal way when it reaches the end of the array; it terminates with a break statement if it finds what it is looking for in the array:

```
for (i=0;i<10;i++)
  {
  if (i==3) break;
  x=x + "The number is " + i + "<br>";
  }
```

Continue

- The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

- This example skips the value of 3

```
for (i=0;i<=10;i++)
 {
 if (i==3) continue;
  x=x + "The number is " + i + "<br>";
  }
```

Nesting

- The placing of one loop inside the body of another loop is called **nesting**.
- When we "**nest**" two loops, the outer loop takes control of the number of complete repetitions of the inner loop.
- While all types of loops may be nested, the most commonly nested loops are **for** loops.
- Example: Multiplication table

Pyetje???