Telerik Academy

Functions

Reusable parts of Code

JavaScript Fundamentals

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

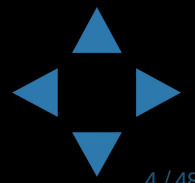
Functions Overview

What is a function?

Telerik Academy

What is a Function?

- A function is a kind of building block that solves a small problem
  - A piece of code that has a name and can be called from other code
  - Can take parameters and return a value
- Functions allow programmers to construct large programs from simple pieces



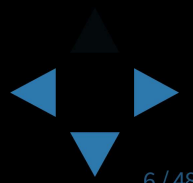
Telerik Academy

Why to Use Functions?

- More manageable programming
  - Split large problems into small pieces
  - Better organization of the program
  - Improve code readability and understandability
  - Enhance abstraction
- Avoiding repeating code
  - Improve code maintainability
- Code reusability
  - Using existing functions several times

Telerik Academy

Declaring and Creating Functions

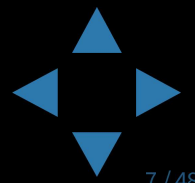


Telerik Academy

Declaring and Creating Functions

- Each function has a name
  - It is used to call the function
  - Describes its purpose
- Functions in JavaScript do not explicitly define return type
  - Each function always returns a value

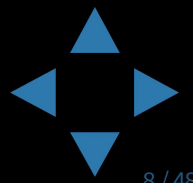
```
function printLogo() {
    console.log("JavaScript Fundamentals");
    console.log("Telerik Software Academy");
}
```

Telerik Academy

Declaring and Creating Functions

Demo



Telerik Academy Ways of Defining a Function

- Functions can be defined in three ways:
  - Using the constructor of the Function object

```
var print = new Function("console.log('Hello')");
```

  - By function declaration

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

  - By function expression

```
var print = function() { console.log("Hello") };
var print = function printFunc() { console.log("Hello"
```

Telerik Academy

Calling Functions

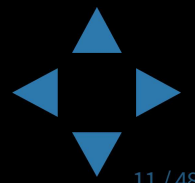
Executing the Function Code

Telerik Academy

Calling Functions

- To call a function, simply use:
  - The function's name
  - Parentheses
  - A semicolon (;)
    - Optional, but preferred
- This will execute the code in the function's body and will result in printing the following:

```
printLogo();
//JavaScript Fundamentals
//Telerik Software Academy
```

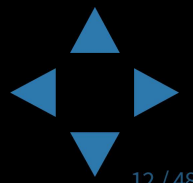



Telerik Academy

Calling Functions

- A function can be called from:
  - Any other function
  - Itself (process known as recursion)

```
function print(){
    console.log("printed");
}

function anotherPrint(){
    print();
    anotherPrint();
}
```

Telerik Academy

Declaring and Calling Functions

Demo

Telerik Academy

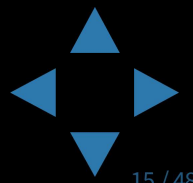
Functions with Parameters

Passing information to functions

Telerik Academy

Function Parameters

- To pass information to a function, you can use parameters (also known as arguments)
  - You can pass zero or several input values
  - Each parameter has a name
  - Parameters are assigned to particular values when the function is called
- Parameters change the function behavior depending on the passed values

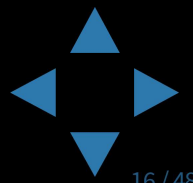


Telerik Academy

Defining and Using Function Parameters

- Function's behavior depends on its parameters
- Parameters can be of any type
  - `Number`, `String`, `Object`, `Array`, etc.
  - Even `Function`

```
function printSign(number) {
    if (number > 0) {
        console.log("Positive");
    } else if (number < 0) {
        console.log("Negative");
    } else {
        console.log("Zero");
    }
}
```

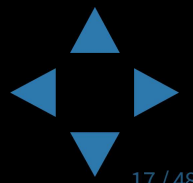



Telerik Academy

Defining and Using Function Parameters

- Functions can have as many parameters as needed:

```
function printMax(x, y) {
  var max;
  x = +x; y = +y;
  max = x;

  if (y > max) {
    max = y;
  }

  console.log(`Maximal number: ${max}`);
}
```

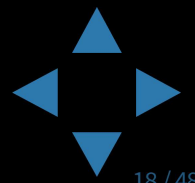



Telerik Academy

Calling Functions with Parameters

- To call a function and pass values to its parameters:
  - Use the function's name, followed by a list of expressions for each parameter
- *Example*:

```
printSign(-5);
printSign(balance);
printSign(2 + 3);
printMax(100, 200);
printMax(oldQuantity * 1.5, quantity * 2);
```

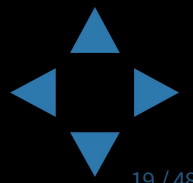



Telerik Academy

Functions Parameters – *Example*

- *Example:* print the sign of a number

```
function printSign(number) {
  number = +number;

  if (number > 0) {
    console.log(`The number ${number} is positive.`);
  } else if (number < 0) {
    console.log(`The number ${number} is negative.`);
  } else {
    console.log(`The number ${number} is zero.`);
  }
}
```



Telerik Academy

Functions Parameters – *Example*

- *Example:* print the max between 2 numbers

```
function printMax(x, y) {
  var max = x;

  if (max < y) {
    max = y;
  }

  console.log(`Maximal number: ${max}`);
}
```


Telerik Academy

Function Parameters

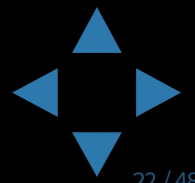
Demo

Telerik Academy Printing Triangle – *Example*

- Creating a program for printing triangles as shown below:

```
n = 6

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
n = 5


1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

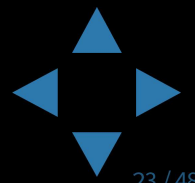


Telerik Academy

Printing Triangle: *Example*

- *Example:* Printing the Triangle:

```
function pringTriangle(n) {
  var line;
  n = +n;

  for (line = 1; line <= n; line += 1) {
    printLine(1, line);
  }

  for (line = n-1; line >= 1; line -= 1) {
    printLine(1, line);
  }

function printLine(start, end) {
  var line = "",
      i;
  start = +start; end = +end;
  for (i = start; i <= end; i += 1){
    line += " " + i;
  }
  console.log(line);
}
```



Telerik Academy

Printing Triangle

Demo

Telerik Academy

The arguments Object

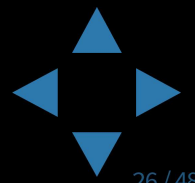
Access to all function parameters

Telerik Academy

arguments Object

- Every function in JavaScript has an implicit parameter `arguments`
  - It holds information about the function and all the parameters passed to the function
  - No need to be explicitly declared
    - It exists in every function

```
function printArguments() {
  var i;
  for(i in arguments) {
    console.log(arguments[i]);
  }
}
printArguments(1, 2, 3, 4); //1, 2, 3, 4
```



Telerik Academy

The arguments Object

- The arguments object is not an array
  - It just has some of the array functionality
- If in need to iterate it, better parse it to an array:

```
function printArguments() {
  var i,
      args;

  args = [].slice.apply(arguments);
  for(i in args) {
    console.log(args[i]);
  }
}

printArguments(1, 2, 3, 4); //1, 2, 3, 4
```

Telerik Academy

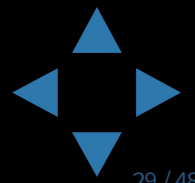
Returning Values From Functions

Telerik Academy

Returning Values from Functions

- Every function in JavaScript returns a value
  - Returns `undefined` implicitly
  - Can be set explicitly
  - The return value can be of any type
    - `Number`, `String`, `Object`, `Function`
    - *Examples:*

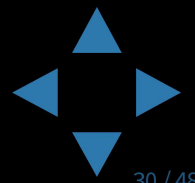
```
var head = arr.shift();
var price = getPrice() * quantity * 1.20;
var noValue = arr.sort();
```

Defining Functions That Return a Value

- Functions can return any type of data:
  - `Number`, `String`, `Object`, etc...
- Use `return` keyword to return a result

```
function multiply (firstNum, secondNum) {
    return firstNum * secondNum;
}

function sum (numbers) {
  var sum = 0, number;
  for(number of numbers){
    sum += number;
  }
  return sum;
}
```



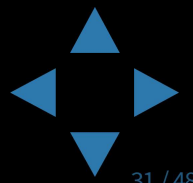
Telerik Academy

The return Statement

- The return statement:
  - Immediately terminates function's execution
  - Returns specified expression to the caller
- To terminate function execution, use just:

```
return;
```

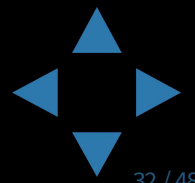
- Return can be used several times in a function body
  - To return a different value in different cases



Telerik Academy The return Statement: *Example*

- *Example:* Check if a number is prime:

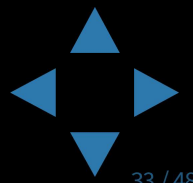
```
function isPrime(number){
  var divider,
      maxDivider;

  number = +number;
  maxDivider = Math.sqrt(number);

  for(divider = 2; divider <= maxDivider; divider += 1){
    if(number % divider === 0) {
      //Divider found, no need to continue execution;
      return false;
    }
  }

  //All dividers tested and none is found
  //The number is prime
  return true;
}
```

Telerik Academy

Return Value

Demo



Sum Even Numbers– *Example*

- Calculate the sum of all even numbers in an array

```
function sum(numbers) {
  var number,
    sum = 0;

  for (number of numbers) {
    if (0 === number % 2) {
      sum += number;
    }
  }
  return sum;
}
```

Telerik Academy

Sum of Even Numbers

Demo

Telerik Academy

Function Scope

Scope of variables and functions

Telerik Academy

Function Scope

- Every variable has its scope of usage
  - A scope defines where the variable is accessible
  - Generally there are local and global scope

```
var arr = [1, 2, 3, 4, 5, 6, 7];

function countOccurences (value) {
  var item,
    count = 0;
  for (item of arr) {
    if (item === value) {
      count++;
    }
  }

  return count;
}
```

arr is in the global scope (it is accessible from anywhere)

count is declared inside countOccurences and it can be used only inside it

Try removing the var before count

Telerik Academy

Function Scope

Demo

Telerik Academy

Function Overloading

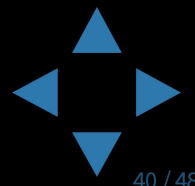
Many functions with the same name

Telerik Academy

Function Overloading

- JavaScript does not support function overloading
  - i.e. functions with the same name hide each other

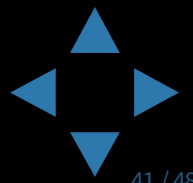
```
function print(number) {
  console.log(`Number: ${number}`);
}

function print(number, text) {
  console.log(`Number: ${number}\nText: ${text}`);
}

print(2);
//prints:
//Number: 2
//Text: undefined`
```



Telerik Academy

Function Overloading in JavaScript

- Function overloading in JavaScript must be faked
  - i.e. make it look like overloading
- Many ways of fake function overloading exist
  - Different number of parameters
  - Different type of parameters
  - Options parameter (preferred)



Function Overloading: Different Number of Parameters

- Different number of parameters:
  - A simple switch by the length of the arguments

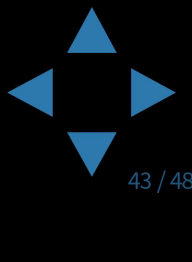
```
function printText (number, text) {
  switch (arguments.length) {
    case 1 : console.log (`Number: ${number}`);
      break;
    case 2 :
      console.log (`Number: ${number}`);
      console.log (`Text: ${text}`);
      break;
  }
}

printText (5); //logs 5
printText (5, "Lorem Ipsum"); //logs 5 and Lorem Ipsum
```

Telerik Academy

Function Overloading: Different Types of Parameters

- Different type of the parameters:
  - A switch on the type of the parameter

```
function printValue (value) {
  switch (typeof value) {
    case "number" : console.log (`Number: ${value}`); break;
    case "string" : console.log (`String: ${value}`); break;
    case "object" : console.log (`Object: ${value}`); break;
    case "boolean" : console.log (`Number: ${value}`); break;
  }
}

printValue (5);
printValue ("Lorem Ipsum");
printValue ([1, 2, 3, 4]);
printValue (true);
```

Telerik Academy

Function Overloading with Default Parameters

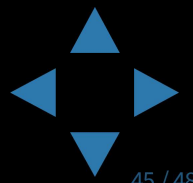
- In JavaScript all parameters are optional
  - i.e. functions can be invoked without them
- Yet, there is a reason behind requesting parameters
  - Maybe the function's behavior depends on it?

Telerik Academy

Function Overloading with Default Parameters

- Default parameters are checked in the function body
  - If the parameter is not present - assign a value

```
//only the str parameter is required
function getRandomValue(str, start, end){
  start = start || 0;
  end = end || str.length;
  //function code
}
```

Telerik Academy

Function Overloading: Options parameter

- To create functions with options parameter
  - Create the function take a single parameter
  - Each parameter is a property of the options parameter
- *Example:*

```
function getRandomValue(opt) {
  var min = +opt.min || Number.MIN_VALUE;
  var max = +opt.max || Number.MAX_VALUE;

  return (Math.random() * (max - min + 1) + min) | 0;
}

console.log(getRandomValue({min:0, max: 15}));
```