COESIONE ITALIA 21-27 PIEMONTE
Cofinanziato dall'Unione europea
REGIONE PIEMONTE

Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Functions

Code Reusability

Shadi Lahham - Web development

Functions in Javascript

Functions

Functions are reusable collections of statements.

```
// declare
function sayMyName() {
    console.log('Hi Bob!');
}

// use
sayMyName();

// use again
sayMyName();
```

Arguments

```
function sayMyName(name) {
    console.log('Hi, ' + name);
}

sayMyName('James');
sayMyName('Adam');
```

Arguments

```
function addNumbers(num1, num2) {
    let result = num1 + num2;
    console.log(result);
}

addNumbers(7, 21);
addNumbers(3, 10);
```

You can also pass variables:

```
let number = 10;
addNumbers(number, 2);
addNumbers(number, 4);
```

Return Values

The return keyword returns a value to whoever calls the function and exits the function

```
function addNumbers(num1, num2) {
    let result = num1 + num2;
    return result; // Anything after this line won't be executed
}

let sum = addNumbers(5, 2);
```

Return Values

You can use function calls in expressions

```
let biggerSum = addNumbers(2, 5) + addNumbers(3, 2);
```

You can even call functions inside function calls:

```
let hugeSum = addNumbers(addNumbers(5, 2), addNumbers(3, 7));
```

Circular Dependencies

```
function chicken() {
    egg();
}

function egg() {
    chicken();
}

egg();
```

Recursion

```
function fibonacci(n) {
  if (n < 2) {
    return n;
  }
  return fibonacci(n - 1) + fibonacci(n - 2);
}

fibonacci(30); // 1439 ms
fibonacci(35); // 12765 ms
fibonacci(40); // 121211 ms
```

Note: recursive functions can be exponentially slow

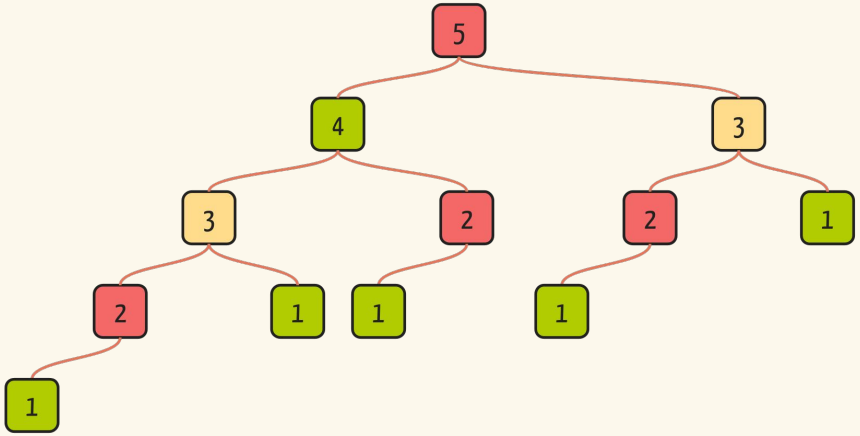
Recursion: The Pros and Cons

Big O Notation and the Nonsense Therein

Recursion

Fibonacci tree

Recursion
An exponential calculation



Variable Scope

- JS Variables are either "block scoped" or "function scope", depending on how they were declared, with let or var
- They are visible in the block or function where they're defined
- Variables can belong to the local or global scope

Local Scope

A variable with "local" scope

```
function addNumbers(num1, num2) {
  let localResult = num1 + num2;
  console.log("The local result is: " + localResult);
}

addNumbers(5, 7);
console.log(localResult);
```

Global Scope

A variable with "global" scope

```
let globalResult;

function addNumbers(num1, num2) {
  globalResult = num1 + num2;
  console.log("The global result is: " + globalResult);
}

addNumbers(5, 7);
console.log(globalResult);
```

Global Scope - side effects

Forgetting to use let has "global" consequences

```
function addNumbers(num1, num2) {
  localResult = num1 + num2;
  console.log("The local result is: " + localResult);
}

addNumbers(5, 7);
console.log(localResult);
```

Coding Conventions: Indentation

Use newlines between statements and use spaces or tabs to indent blocks.

Bad:

```
function addNumbers(num1,num2) {return num1 + num2;}
```

Bad:

```
function addNumbers(num1, num2) {
return num1 + num2;
}
```

Better:

```
function addNumbers(num1, num2) {
    return num1 + num2;
}
```

Convention: Comments & documentation

Comment functions properly.

OK, but not great:

```
/*
 * Adds two numbers and returns the sum
 */
function addNumbers(num1, num2) {
    return num1 + num2;
}
```

Convention: Comments & documentation

Comment functions properly. Use JSDoc

Much better:

```
/**
 * Returns the sum of num1 and num2
 * @param {number} num1 - the first number
 * @param {number} num2 - the second number
 * @returns {number} Sum of num1 and num2
 */
function addNumbers(num1, num2) {
    return num1 + num2;
}
```

Use JSDoc: Documentation
JSDoc on github

Let & var

Let vs var

```
function worker() {
   let x = 88;
   for (let i = 0; i < 4; i++) {
       console.log('i block =', i);
   }
   console.log('x func =', x);
   console.log('i !block =', i); // undefined
}

worker();
console.log('x !func =', x); // undefined
```

let: Block-scoped

Access restricted to nearest enclosing block

```
function worker() {
   var x = 88;
   for (var i = 0; i < 4; i++) {
       console.log('i block =', i);
   }
   console.log('x func =', x);
   console.log('i !block =', i); // output?
}
worker();
console.log('x !func =', x); // undefine
```

var: Function-scoped

Access restricted to nearest enclosing function
Common in older Javascript code

Advanced functions

Another way to look at functions

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

let mad = add;

let resultA = add(5, 4); // 9

let resultB = mad(21, 7); // 28

console.log(typeof add); // function
```

note: functions are regular objects with the additional capability of being callable

Another way to look at functions

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

let mult = function(a, b) {
 return a * b;
};

let calculate = function(fn, a, b) {
 console.log('This is your result:', fn(a, b));
};

calculate(add, 2, 4);
calculate(mult, 2, 4);
```

note: functions can be passed as parameters

Arrow Functions

Arrow Functions: Syntax

- A function shorthand
- Use the => syntax
- Share the same lexical **this** as their surrounding code

Syntax

```
(x, y, z) => { statements }
(x, y, z) => expression // same as: (x, y, z) => { return expression; }
```

Optional parentheses

```
(x) => { statements }
x => { statements }
```

No parameters syntax

```
() => { statements }
```

Arrow Functions: Variants

```
function square(a) {
 return a * a;
}

let square = (a) => {
 return a * a;
};

// equivalent
let square = (a) => a * a;

// equivalent
let square = a => a * a;
```

Arrow functions are functions

```
let add = (x, y) => { return x + y; };

console.log(typeof add); // function

console.log(add instanceof Function); // true
```

note: instanceof is a binary operator

Your turn

1.Variable Scope

- Recreate the local and global scope examples in your browser
- Try to call the function “addNumbers” a few more times
- Write a new .js file that uses both local and global variables in the same project
- Make sure that you understand exactly what’s happening at every stage

2.Fortune calculator

- Write a function named tellFortune that:
  - Takes 4 arguments: number of children, partner's name, geographic location, job title.
  - outputs your fortune to the screen like so: "You will be a X in Y, and married to Z with N kids."

- Call that function 3 times with 3 different values for the arguments

3.Dog age calculator

Calculate a puppy’s age in dog years

- Write a function named calculateDogAge that:
  - takes 1 argument: the dog's age in human years
  - calculates the dog's age based on the conversion rate of 1 human year to 7 dog years
  - outputs the result to the screen like so: "Your dog is NN years old in dog years!"

- Call the function three times with different sets of values
- Bonus:
  - Add another argument to the function that takes the conversion rate of human to dog years

4.Coffee supply calculator

- Write a function named calculateSupply that:
  - takes 2 arguments: age, amount per day.
  - calculates the amount consumed for rest of the life (based on a constant max age).
- outputs the result to the screen like so: "You will need NN cups of coffee to last you until the age of X"

- Call that function three times, passing in different values each time
- Bonus:
  - Calculate in liters, accepting floating point values for amount per day (0.3 liters of coffee)
  - Round the result to a round number

5.Geometry library

- Create a function called calcCircumfrence:
  - Pass the radius to the function
  - Calculate the circumference based on the radius, and output "The circumference is NN"
- Create a function called calcArea:
  - Pass the radius to the function.
  - Calculate the area based on the radius, and output "The area is NN"

Reference:

JavaScript Math Object

Circles

Bonus

6.Temperature conversion

Create a function called celsiusToFahrenheit:

- Store a celsius temperature into a variable.
- Convert it to fahrenheit and output "NN°C is NN°F".

Create a function called fahrenheitToCelsius:

- Now store a fahrenheit temperature into a variable.
- Convert it to celsius and output "NN°F is NN°C."

7.Math library

- Write a function called squareNumber that will take one argument (a number), square that number, and return the result. It should also log a string like "The result of squaring the number 3 is 9."

- Write a function called halfNumber that will take one argument (a number), divide it by 2, and return the result. It should also log a string like "Half of 5 is 2.5."

7.Math library

- Write a function called **percentOf** that will take two numbers, figure out what percent the first number represents of the second number, and return the result. It should also log a string like "2 is 50% of 4."
- Write a function called **areaOfCircle** that will take one argument (the radius), calculate the area based on that, and return the result. It should also log a string like "The area for a circle with radius 2 is 12.566370614359172."

- Bonus: round the result so there are only two digits after the decimal

8.Calculator

Write a function that will take one argument (a number) and perform the following operations, using the functions you wrote earlier:

- Take half of the number and store the result
- Square the result of #1 and store that result
- Calculate the area of a circle with the result of #2 as the radius
- Calculate what percentage that area is of the squared result (#3)

9.Merger

Write a function called merger() that takes two arguments and performs the following operation:

- If both arguments are numbers, return the sum
- If both arguments are strings, return the concatenation of the strings
- If the arguments are anything else, return null

Include a doc file in which you explain why two operators might have the same symbol but work differently based on the type of the parameters

References

JavaScript Function Definitions

JavaScript - Functions

Javascript validation

JSHint | code quality tool