{POWER.CODERS}

JavaScript exercises with functions

Agenda

Today we will recap what we learned so far and do exercises in JavaScript. Also we will look into functions.

REMEMBER THIS EXAMPLE

The **Euclidean algorithm**

Find the greatest common divisor of 2 positive integers.

In math, the GCD of 2 or more positive integers is the largest positive integer that divides each of the integers.

The pseudo-code on Github

JAVASCRIPT FUNCTIONS

What is a function?

A block of code designed to perform actions. The purpose is to perform a particular task (one piece of solving the problem).

What is a function?

A block of code designed to perform actions. The purpose is to perform a particular task (one piece of solving the problem).

- **Code reuse**: define the code once and use it many times

What is a function?

A block of code designed to perform actions. The purpose is to perform a particular task (one piece of solving the problem).

- **Code reuse**: define the code once and use it many times
- **Different arguments**: used with the same code will produce different results.

Functions we know

Functions we know

- `window.alert()`
- `window.prompt()`
- `console.log()`
- `document.write()`

Functions we know

```
> window.alert()
> window.prompt()
> console.log()
> document.write()
```

See the () at the end. You need these to execute the function.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.
- `name` of the function the same rules apply as in naming variables.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.
- `name` of the function the same rules apply as in naming variables.
- `(arg1, arg2, arg3)` directly after the **name** for (optional) **arguments**. The "placeholders" are called parameters.

  You can have as many as you wish.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.
- `name` of the function the same rules apply as in naming variables.
- `(arg1, arg2, arg3)` directly after the **name** for (optional) **arguments**. The "placeholders" are called parameters.

  You can have as many as you wish.
- `{}` holds the code to be executed when the function is called.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.
- `name` of the function the same rules apply as in naming variables.
- `(arg1, arg2, arg3)` directly after the **name** for (optional) **arguments**. The "placeholders" are called parameters.

  You can have as many as you wish.
- `{}` holds the code to be executed when the function is called.
- `return` keyword (optional) breaks the execution and returns the value.

Function declaration

```
function myFunction(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- `function` keyword to tell JS that a function declaration starts.
- `name` of the function the same rules apply as in naming variables.
- `(arg1, arg2, arg3)` directly after the **name** for (optional) **arguments**. The "placeholders" are called parameters.

  You can have as many as you wish.
- `{}` holds the code to be executed when the function is called.
- `return` keyword (optional) breaks the execution and returns the value.
- `arg1 + arg2 * arg3` is called the **return value**.

  You can only have **one** return value.

Calling a function

To execute the code inside a function, you need to call it.

```
myFunction(item1, item2, item3);
```

Calling a function

To execute the code inside a function, you need to call it.

```
myFunction(item1, item2, item3);
```

- `name` - start by writing the name of the function
- `(arguments)` - add the arguments in parentheses
- `;` - always remember to use a semicolon

Calling a function

When calling a function, provide the arguments in the same order in which you defined them.

Calling a function

When calling a function, provide the arguments in the same order in which you defined them.

- If there a fewer arguments when defined, they will be declared without a value and get the datatype **undefined**.

Calling a function

When calling a function, provide the arguments in the same order in which you defined them.

- If there a fewer arguments when defined, they will be declared without a value and get the datatype **undefined**.
- If there are more, they will be added inside the array **arguments**.

What character is used to separate arguments from each other?

Calling a function

Once the function is declared, you can call it **everywhere** as many times as you want:

Calling a function

Once the function is declared, you can call it **everywhere** as many times as you want:

- before the function declaration
- after the function declaration
- inside the function declaration (=recursive functions)

Recursive functions

A recursive function calls itself inside its code block.

```
var countdown = function(value) {
if (value > 0) {
  console.log(value);
  return countdown(value - 1);
} else {
  return value;
}
};
countdown(10);
```

Default arguments

You can give default arguments when declaring a function.

```
var myFunction = function(arg1=5, arg2=3, arg3=1){
  return arg1 + arg2 * arg3;
}

myFunction();
myFunction(20);
```

FUNCTION EXPRESSION

```
var myFunction = function(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

FUNCTION EXPRESSION

```
var myFunction = function(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- A JavaScript function can also be defined using an **expression**.

FUNCTION EXPRESSION

```
var myFunction = function(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- A JavaScript function can also be defined using an **expression**.
- A function expression can be stored in a variable with the keyword `var`.

FUNCTION EXPRESSION

```
var myFunction = function(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- A JavaScript function can also be defined using an **expression**.
- A function expression can be stored in a variable with the keyword `var`.
- After a function expression has been stored in a variable, it can be referenced (called) by it.

Function expression

```
var myFunction = function(arg1, arg2, arg3){
  return arg1 + arg2 * arg3;
}
```

- A JavaScript function can also be defined using an **expression**.
- A function expression can be stored in a variable with the keyword `var`.
- After a function expression has been stored in a variable, it can be referenced (called) by it.
- Functions stored in variables do not need function names. They are called **anonymous** functions.

DECLARATION VS. EXPRESSION

- Function **declarations** load **before** any code is executed.
- Function **expressions** load only when the interpreter reaches **that line** of code.

DECLARATION VS. EXPRESSION

- Function **declarations** load **before** any code is executed.
- Function **expressions** load only when the interpreter reaches **that line** of code.

Function expressions become sometimes more useful than function declarations, e.g. if they are used as arguments to other functions.

Arrow function

New in ES6

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

Arrow function

New in ES6

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

- An arrow function is a compact alternative.

Arrow function

New in ES6

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

- An arrow function is a compact alternative.
- An arrow function cannot be used as a method.

Comparing functions

Breakdown to get an arrow function

```
function (a){
    return a + 100;
  }

  /**
  * 1. Remove the word "function" and place arrow between
  * the argument and opening body bracket */
  (a) => {
    return a + 100;
  }
```

Comparing functions

Breakdown to get an arrow function

```
/**
  * 1. Remove the word "function" and place arrow between
  * the argument and opening body bracket */
  (a) => {
    return a + 100;
  }

  /**
  * 2. Remove the body brackets and word "return" -- the return is implied.
  */
  (a) => a + 100;
```

Comparing functions

Breakdown to get an arrow function

```
/**
  * 2. Remove the body brackets and word "return" -- the return is implied.
  */
  (a) => a + 100;

  /**
  * 3. Remove the argument parentheses
  */
  a => a + 100;
```

Arrow functions

Each step results in a valid arrow function

Arrow functions

Each step results in a valid arrow function

- If there are no parameters: `() => { statements }`

Arrow functions

Each step results in a valid arrow function

- If there are no parameters: `() => { statements }`
- If there is 1 parameter: `parameter => { statements }`

Arrow functions

Each step results in a valid arrow function

- If there are no parameters: `() => { statements }`
- If there is 1 parameter: `parameter => { statements }`
- If an espression is returned: `parameter => expression`

No parameters

```
() => {
  let x = 5 + 100;
  alert(x);
}
```

1 PARAMATER

```
a => {
  let x = a + 100;
  alert(x);
}
```

More paramaters

```
(a, b) => {
  let x = a + b;
  alert(x);
}
```

Expression

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

EXERCISES

SAY HELLO

Create a program in JavaScript that prompts for your name and prints a greeting using your name.

Improve your code by turning it to function

Counting the number of characters

Create a program in JavaScript that prompts for an input string and displays output that shows the input string and the number of characters the string contains.

Improve your code by turning it to function

Printing quotes

Create a program that prompts for a quote and an author. Display the quotation and author as shown here:
[Author] says, "[Quote]" (Replace [Author] and [Quote] with the actual values

Improve your code by turning it to function

Age calculator

Improve your age calculator

- First add an input for the month
- Then change that to using a function
- Call that function several times

Improve your code by turning it to function

What number is bigger?

Write a function named greaterNum that:

- Takes 2 arguments, both numbers.
- Returns whichever number is the greater (higher) number.
- Call that function 2 times with different number pairs
- Log the output, e.g. "The greater number of 5 and 10 is 5."

RETIREMENT CALCULATOR

Improve your retirement calculator by using function(s)

TIP CALCULATOR

Improve your tip calculator by using function(s).

THE WORLD TRANSLATOR

Write a function named helloWorld that:

- Takes 1 argument, a language code (e.g. "es", "de", "en")
- Returns "Hello, World" for the given language, for at least 3 languages. It should default to returning English.
- Call that function for each of the supported languages and log the result to make sure it works

Work on your project