JAVASCRIPT COURSE

PART TWO - 18.10.2017.

ICT HUB

JS

IN THIS CLASS

- Functions
- Function scopes
- Strict mode
- Closure
- Immediately Invoked Function Expressions (IIFEs)
- this

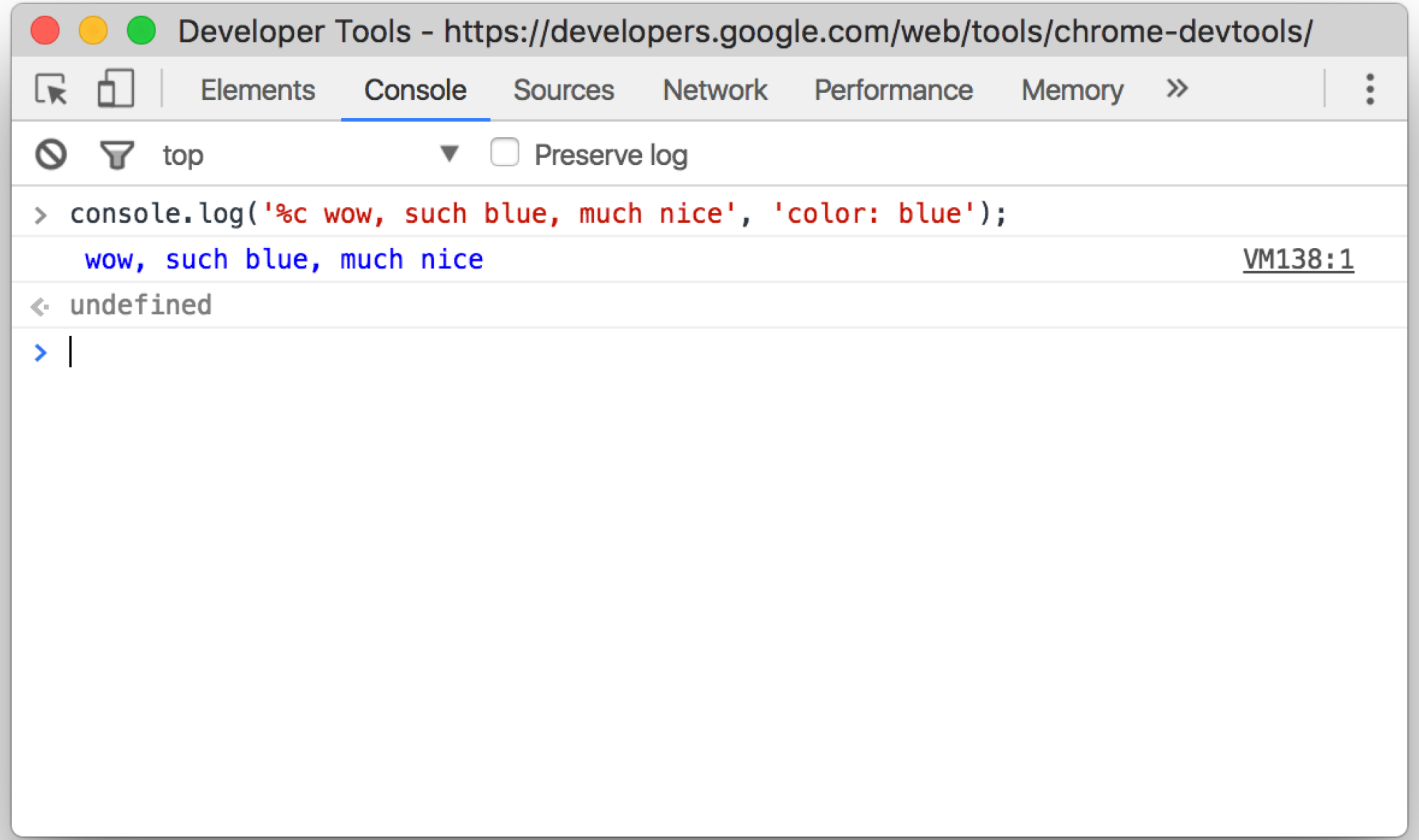
BEFORE WE BEGIN

Presentations

https://github.com/JSBelgrade/course-2017

REMINDER

Run the examples in
Chrome Dev Tools


Developer Tools - https://developers.google.com/web/tools/chrome-devtools/
Elements
Console
Sources
Network
Performance
Memory
top
Preserve log

```
> console.log('%c wow, such blue, much nice', 'color: blue');
   wow, such blue, much nice                                    VM138:1
<· undefined
>
```

- Chrome's Main Menu > More Tools > Developer Tools
- Right-click a page element and select Inspect
- Command+Option+I (Mac) or Control+Shift+I (Windows, Linux)

EXERCISE

Write a program that uses `console.log` to print all the numbers from 1 to 100, with two exceptions.

For numbers divisible by 3, print `"Fizz"` instead of the number, and for numbers divisible by 5 (and not 3), print `"Buzz"` instead.

Hint:

```
 9 % 3 = 0
10 % 3 = 1
11 % 3 = 2
```

Simple solution

```
for (let i = 1; i < 101; i++) {
  if (i % 3 === 0 && i % 5 === 0) {
    console.log('FizzBuzz');
  } else if (i % 3 === 0) {
    console.log('Fizz');
  } else if (i % 3 === 0) {
    console.log('Buzz');
  } else {
    console.log(i);
  }
}
```

FUNCTIONS

Functions are one of the fundamental building blocks in JavaScript.

A function is a JavaScript procedure,
a set of statements,
that performs a task or calculates a value.

```
function name(parameters) {
  function body
}
```

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

square(5); // 25
```

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

sum(2, 3); // 5
```

```
function log(something) {
  console.log('Log:');
  console.log(something);
}

log('hello');
// Log:
// hello
```

```javascript
function sayHello() {
  console.log('Hello!');
}

sayHello();
// Hello!
```

FUNCTION EXPRESSIONS

```
const name = function (parameters) {
  function body
}
```

Anonymous functions

```
const square = function(number) {
  return number * number;
}

square(5); // 25
```

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

sum(2, 3); // 5
```

```
const log = function(something) {
  console.log('Log:');
  console.log(something);
}

log('hello');
// Log:
// hello
```

```
const sayHello = function() {
  console.log(arguments);
}

sayHello();
// Hello!
```

CALLING FUNCTIONS

```
functionName(arguments);
```

```
square(5); // 25
square(5, 3); // ?
square(5, 3); // 25
square(); // ?
square(); // NaN
```

EXERCISE:
HELLO!

Loop through the array of full names.
Then invoke a function with each full name that will split it into first and last name, and print "Hello [first name]!".

Hint:

```
'Slobodan Stojanovic'.split(' ');
```

For array:

```
['James Bond', 'Sherlock Holmes']
```

Result should be:

```
Hello James!
Hello Sherlock!
```

```
const names = ['James Bond',
'Sherlock Holmes'];

function sayHello(fullName) { /*…*/ }

Output:
// Hello James!
// Hello Sherlock!

Hint: 'John Doe'.split(' ');
```

Simple solution

```javascript
const names = ['James Bond', 'Sherlock Holmes'];

for (let i = 0; i < names.length; i++) {
  const splitName = names[i].split(' ');
  console.log('Hello ' + splitName[0] + '!');
}
```

Simple solution with function

```
const names = ['James Bond', 'Sherlock Holmes'];

function sayHello(fullName) {
  const splitName = fullName.split(' ');
  console.log('Hello ' + splitName[0] + '!');
}

for (let i = 0; i < names.length; i++) {
  sayHello(names[i]);
}
```

THROWING ERRORS

The Error constructor creates an error object. Instances of Error objects are thrown when runtime errors occur.

Create new Error:

```
new Error(message, fileName, lineNum)
```

Error types

- TypeError
- ReferenceError
- RangeError
- URIError
- SyntaxError
- EvalError
- InternalError

`throw`

```
throw new Error('Whoops!');
throw new TypeError('Whoops!');
throw 'Whoops!';
```

Read more:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error

HOMEWORK EXERCISE: FACTORIAL FUNCTION

Create a factorial function.

In mathematics, the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.

For example,
5! = 5 * 4 * 3 * 2 * 1 = 120

Hint:
Recursion.

It's an important programming technique, in which a function calls itself.

```
function count(n) {
  console.log(n);

  if (n > 0) {
    count(n - 1);
  }
}

count(5);
```

FUNCTION SCOPES AND PARAMETERS

The parameters to a function behave like regular variables, but their initial values are given by the caller of the function, not the code in the function itself.

```
function sum(a, b) {
  let a = 0;

  return a + b;
}

sum(5, 5); // ?
```

```
function sum(a, b) {
  a = 0;

  return a + b;
}

sum(5, 5); // ?
```

```
function sum(a, b) {
  let c = 0;

  return a + b;
}

console.log(c); // ?
```

```
function sum(a, b) {
  var c = 0;

  return a + b;
}

console.log(c); // ?
```

Variables created inside functions and parameters are local to the function.

```
let x = 'outside';

function doSomething() {
  let x = 'inside';

  return x;
}

doSomething();
console.log(x); // ?
```

```
let x = 'outside';

function doSomething() {
  let x = 'inside';

  return x;
}

doSomething();
console.log(x); // outside
```

Nested scopes

```
let x = 'outside';

function doSomething() {
  x = 'inside';

  return x;
}

doSomething();
console.log(x); // ?
```

```
let x = 'outside';

function doSomething() {
  x = 'inside';

  return x;
}

doSomething();
console.log(x); // inside
```

Hoisting

```
square(3);
// 9

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

```
square(3);
// ?

var square = function(number) {
  return number * number;
}
```

```
square(3);
▶ Uncaught TypeError: square is
not a function

var square = function(number) {
  return number * number;
}
```

Why?

```
someVar = 2;

var someVar;

console.log(someVar);
// ?
```

```
someVar = 2;

var someVar;

console.log(someVar);
// 2
```

```
console.log(otherVar);
// ?

var otherVar = 2;
```

```
console.log(otherVar);
// undefined

var otherVar = 2;
```

What about `let` and `const`?

```
anotherVar = 2;

let anotherVar;

console.log(anotherVar);
// ?
```

```
anotherVar = 2;

let anotherVar;

console.log(anotherVar);
▶ ReferenceError: anotherVar
is not defined
```

STRICT MODE

```
"use strict";
```

Strict mode is added by ES5 to introduce better error-checking into your code.

```
'use strict';

function someFunc() {
  var testVar = 4;
  return testVar;
}

testVar = 5;
// Syntax error.
```

```
function someFunc() {
  'use strict';

  testVar = 4;
  // Syntax error.
  return testVar;
}

testVar = 5;
```

Why `'use strict';` instead of `useStrict();`?

Read more:

https://docs.microsoft.com/en-us/scripting/javascript/advanced/strict-mode-javascript

CLOSURES

"What happens to local variables when the function call that created them is no longer active?"

```
function wrapValue(n) {
  var local = n;
  return function() {
    return local;
  };
}

var wrap1 = wrapValue(1);
var wrap2 = wrapValue(2);
console.log(wrap1()); // 1
console.log(wrap2()); // 2
```

"Closure is one of the most important,
and often least understood,
concepts in JavaScript."

```
function multiplier(factor) {
  return function(number) {
    return number * factor;
  };
}

const twice = multiplier(2);
console.log(twice(5)); // 10
```

IMMEDIATELY INVOKED FUNCTION EXPRESSIONS (IIFEs)

```
(function IIFE(){
  console.log('Hello!');
})();

// Hello!
```

Why?

Scope isolation.

```
let a = 42;

(function IIFE(){
  let a = 10;
  console.log(a); // 10
})();

console.log(a); // 42
```

Example:
jQuery

THIS

If a function has a `this` reference inside it, that `this` reference usually points to an object.

But which object it points to depends on how the function was called.

`this` refers to object not function itself.

```
var bar = 'global';

function foo() {
  console.log(this);
  // Window
  console.log(this.bar);
  // global
}

foo();
```

```
const obj1 = {
  bar: 'In obj1',
  foo: foo
};

obj1.foo();
// obj1
// In obj1
```

```javascript
const obj2 = {
  bar: 'In obj2'
};

foo.call(obj2);
// obj2
// In obj2
```

`.call` and `.apply`

Arguments as array:

```
func.apply(thisArg, [argsArray])
```

Arguments as, well, arguments 😀

```
func.call(thisArg, arg1, arg2, ...)
```

READ MORE

Eloquent JavaScript

Marijn Haverbeke

https://eloquentjavascript.net

You Don’t know JavaScript

Kyle Simpson

https://github.com/getify/You-Dont-Know-JS

JavaScript: The Definitive Guide

David Flanagan

http://shop.oreilly.com/product/9780596805531.do

JavaScript: The Good Parts

Douglas Crockford

http://shop.oreilly.com/product/9780596517748.do

THE END

OF PART TWO