JAVASCRIPT COURSE

PART THREE - 26.10.2017.

ICT HUB

JS

IN THIS CLASS

- Arrow Functions
- Classes
- How to run JavaScript in the browser

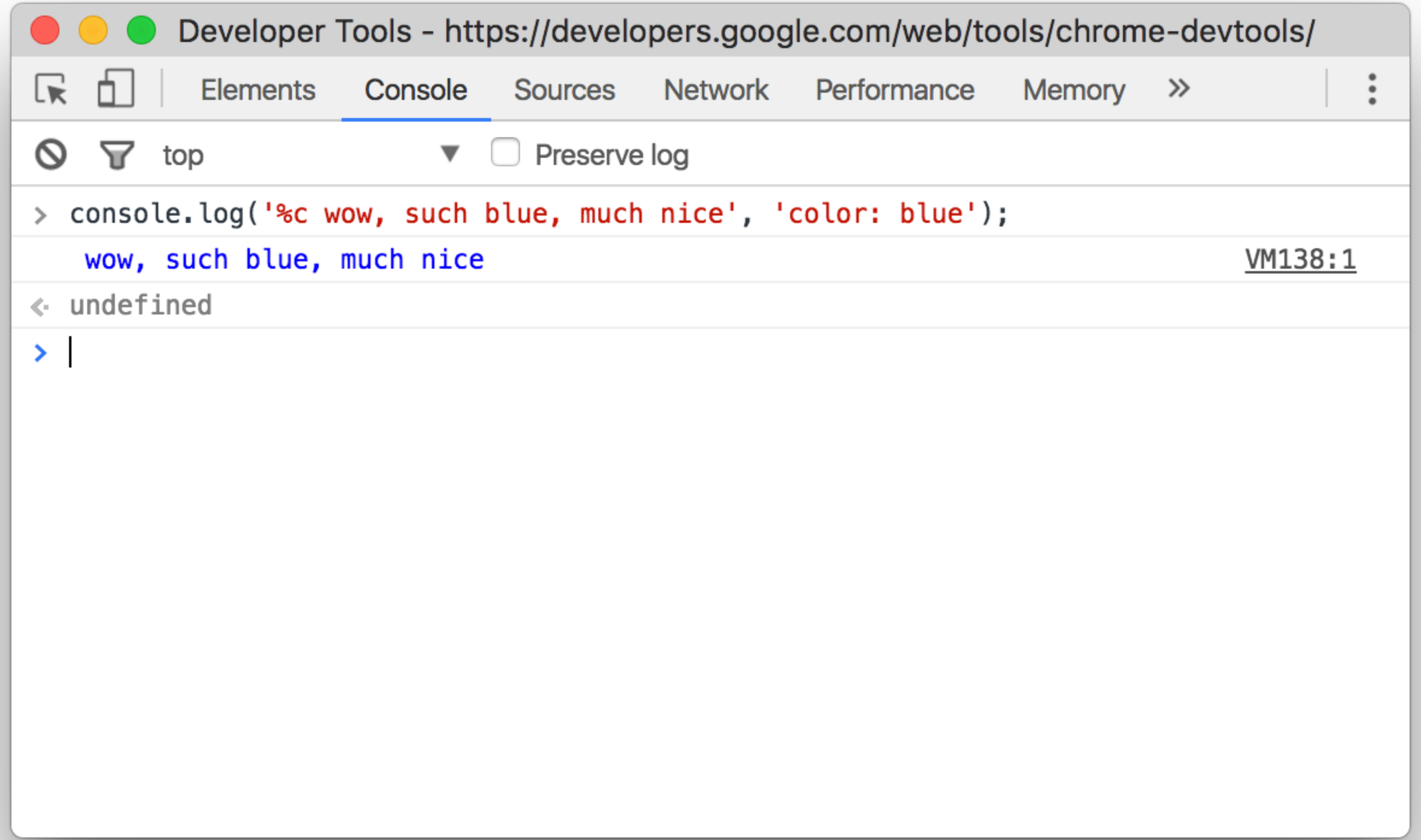
BEFORE WE BEGIN

Presentations and homeworks:

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)

PREVIOUS HOMEWORK

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.

SOLUTION

```
0! = 1                 //   1
1! = 1                 //   1
2! = 2 * 1             //   2
3! = 3 * 2 * 1         //   6
4! = 4 * 3 * 2 * 1     //  24
5! = 5 * 4 * 3 * 2 * 1 // 120
```

```
function factorial(n) {
  // Do something and return
}

factorial(0);   // 1
factorial(1);   // 1
factorial(2);   // 2
factorial(3);   // 6
factorial(4);   // 24
factorial(5);   // 120
factorial('A'); // TypeError
```

```
function factorial(n) {
  if (n === 0) {
    return 1;
  }
}
```

```
factorial(0);   // 1         [1]
factorial(1);   // undefined [1]
factorial(2);   // undefined [2]
factorial(3);   // undefined [6]
factorial(4);   // undefined [24]
factorial(5);   // undefined [120]
factorial('A'); // undefined
                   [TypeError]
```

```
function factorial(n) {
  if (n === 0) {
    return 1;
  }

  return n * factorial(n - 1);
}
```

```
factorial(0);   // 1         [1]
factorial(1);   // 1         [1]
factorial(2);   // 2         [2]
factorial(3);   // 6         [6]
factorial(4);   // 24.       [24]
factorial(5);   // 120       [120]
factorial('A'); // RangeError
                   [TypeError]
```

```
function factorial(n) {
  if (typeof n !== 'number') {
    throw new TypeError();
  }

  if (n === 0) {
    return 1;
  }

  return n * factorial(n - 1);
}
```

```
factorial(0);   // 1         [1]
factorial(1);   // 1         [1]
factorial(2);   // 2         [2]
factorial(3);   // 6         [6]
factorial(4);   // 24.       [24]
factorial(5);   // 120       [120]
factorial('A'); // TypeError
                   [TypeError]
```

SOFTWARE TESTING

Process of validating and verifying that a software program or application or product:

- Meets the business and technical requirements that guided it’s design and development
- Works as expected
- Can be implemented with the same characteristic.

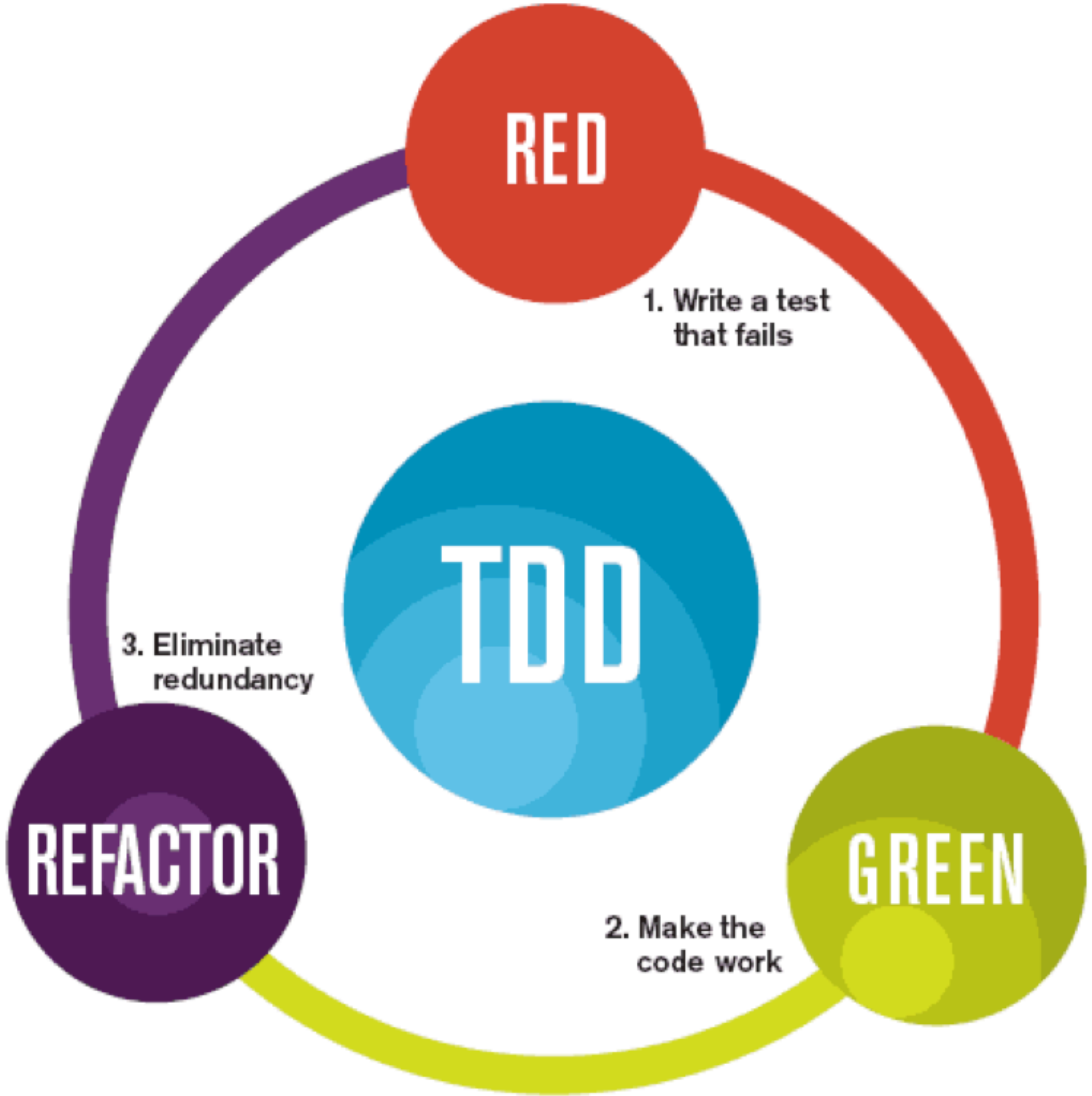
Automatic vs manual testing

Automatic vs manual testing

TEST DRIVEN DEVELOPMENT (TDD)

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle:

- Test
- Code
- Refactor



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

Requirements are turned into very specific test cases, then the software is improved to pass the new tests, only.

This is opposed to software development that allows software to be added that is not proven to meet requirements.

EXERCISE #1

FIND EDGE CASES FOR OUR FACTORIAL FUNCTION

```
function factorial(n) {
  if (typeof n !== 'number') {
    throw new TypeError();
  }

  if (n === 0) {
    return 1;
  }

  return n * factorial(n - 1);
}
```

```
factorial(-1);

factorial(10.23);

factorial(NaN);
```

QUICK REMINDER FROM PREVIOUS CLASS

FUNCTIONS

FUNCTION EXPRESSIONS

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

Anonymous functions

CALLING FUNCTIONS

```
functionName(arguments);
```

THROWING ERRORS

Create new Error:

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

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.

Nested scopes

```
let x = 'outside';

function doSomething() {
  x = 'inside';

  return x;
}

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

Hoisting

```
someVar = 2;

var someVar;

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

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

var otherVar = 2;
```

What about `let` and `const`?

```
anotherVar = 2;

let anotherVar;

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

STRICT MODE

```
"use strict";
```

CLOSURES

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

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

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

IMMEDIATELY INVOKED FUNCTION EXPRESSIONS (IIFEs)

Why?

Scope isolation.

```
let a = 42;

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

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

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.

CONTINUE

ARROW FUNCTIONS

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

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

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

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

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

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

Shorter syntax than a function expression
and does not bind its own this.

Syntax:

```
(arg1, arg2) => expression;
```

```
argument => expression;
argument => {
  expression1;
  expression2;
}
```

```
argument => {
  expression1;
  return expression2;
}
```

Examples:

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

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

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

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

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

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

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

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

CLASSES & PROTOTYPES

Functions are objects.

```
function foo() {
  return 'bar';
}

foo.buz = 5;
```

Doesn't seems useful?
How about this?

```
function Animal(type) {
  this.type = type;
}

Animal.prototype.getType =
  function() {
    return this.type;
  };

const cow = new Animal('cow');
cow.getType(); // cow
```

```
class Animal {
  constructor(type) {
    this.type = type;
  }

  getType() {
    return this.type;
  }
}

const cow = new Animal('cow');
```

Extending class:

```
class Cow extends Animal {
  constructor(name) {
    super('cow');
    this.name = name;
  }

  sound() {
    console.log('moo');
  }
}

const cow = new Cow('Milka');
```

RUN JavaScript FILE
IN THE BROWSER

```
<!doctype html>
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
  </body>
<html>
```

```
<!doctype html>
<html>
  <head>
    <title>Hello</title>
    <script src="file.js"></script>
  </head>
  <body>
  </body>
<html>
```

```
<!doctype html>
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <script src="file.js"></script>
  </body>
<html>
```

EXERCISES

EXERCISE 2

Write a function that converts Celsius temperature to Fahrenheit.

For °C to °F, multiply by 9, then divide by 5, then add 32 tddbin.com

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

EXERCISE 3

Write a function that converts Celsius to Fahrenheit or Fahrenheit to Celsius depending on the second argument.

For °C to °F, multiply by 9, then divide by 5, then add 32

EXERCISE 4

Write a function that returns the first n Fibonacci numbers.

The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . Each subsequent number is the sum of the previous two.

EXERCISE 5

Write a function to find the first not repeated character.

Sample arguments : 'abacddbec'
Expected output : 'e'

EXERCISE 6

Write a function that accepts a string as a parameter and converts the first letter of each word of the string in upper case.

Example string : 'the quick brown fox'
Expected Output : 'The Quick Brown Fox '

EXERCISE 7

Write a function that accepts the string and reverses it.

Input : 'hello'
Expected Output : 'olleh'

EXERCISE 8

Write a function that checks if provided string is palindrome.

A palindrome is a word or sentence that's spelled the same way both forward and backward, ignoring punctuation, case, and spacing.

HOMEWORK

Make Calculator UI (HTML and CSS).

Design (feel free to change it):

https://dribbble.com/shots/3344091-Daily-Ui-004-Calculator

READ (AND LEARN) MORE

Free Code Camp

Learn to code for free.

https://www.freecodecamp.org

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