
Sprint 02
Marathon Full Stack
September 9, 2021
ucode

Contents

Engage

DESCRIPTION

Welcome to Sprint02!

You now have a good foundation in HTML and CSS, and you can develop attractive and informative web pages. However, so far, you have been creating only static pages that always look and behave the same each time they are loaded into the browser.

A dynamic website contains both client-side and/or server-side scripting to generate changeable content. You can use JavaScript, or other scripting languages, to dynamically change the data of a web page.

Dynamic web pages include the following features:

- content that can be changed dynamically
- dynamic positioning of web page elements
- dynamic style

With just a few lines of JavaScript, your web page will become dynamic and functional.

So, let's get started with learning JS!

BIG IDEA

Dynamic web pages.

ESSENTIAL QUESTION

How to create an interactive web page?

CHALLENGE

Explore JavaScript basics.

ucode

Sprint 02 | Marathon Full Stack **> 2**

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What are the differences between Web 1.0 and Web 2.0?
- What does DHTML mean?
- What is JavaScript usually used for?
- What is the difference between JavaScript and Java?
- Which extension do JavaScript files use?
- Is Javascript a compiled or an interpreted language?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

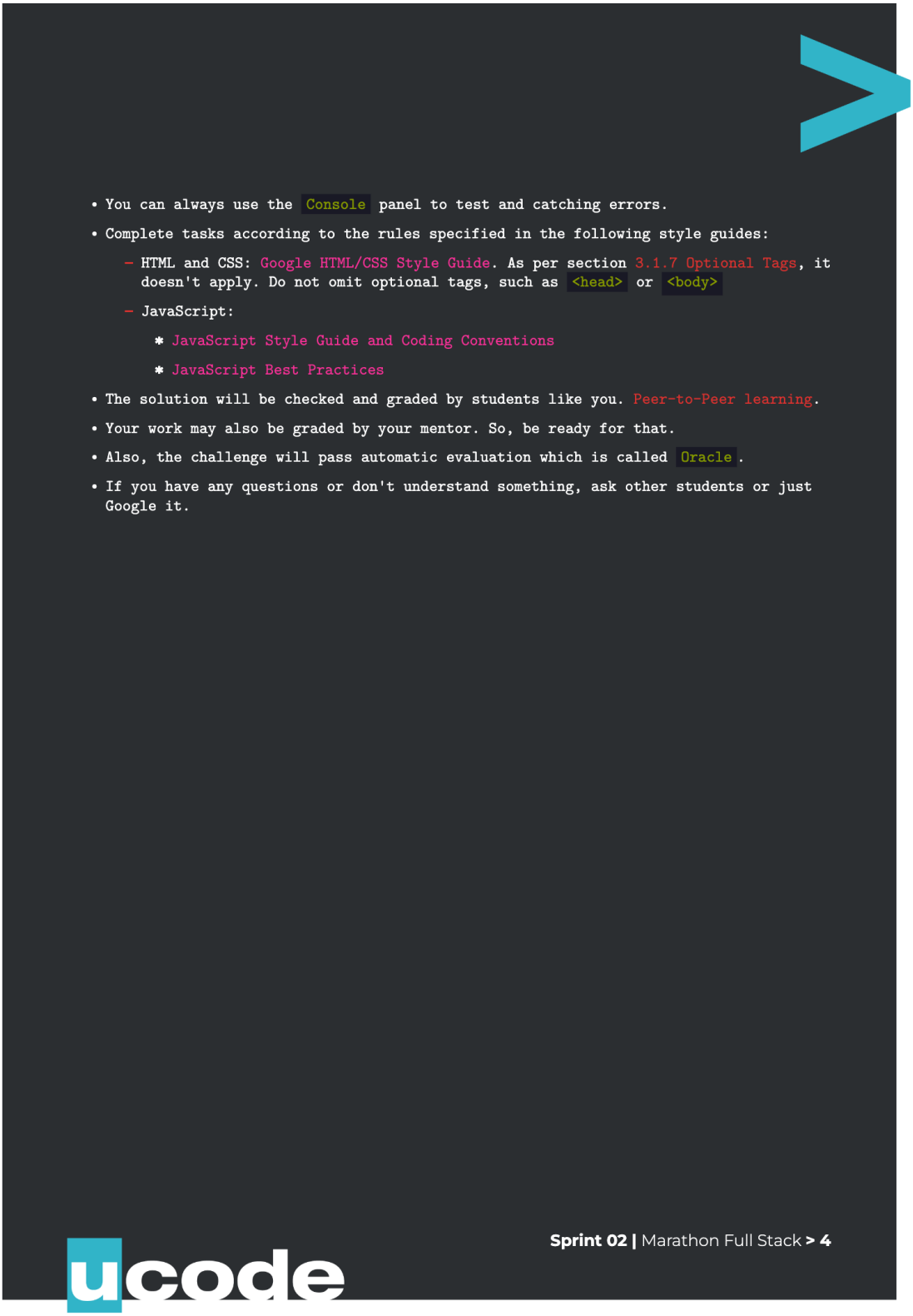
- Take into account: most of you questions about JS can be answered here and here.
- Read about debugging inside a browser (e.g Chrome).
- Get acquainted with JavaScript Style Guide and Coding Conventions and JavaScript Best Practices.
- Clone your git repository, issued on the challenge page in the LMS.
- Employ the full power of P2P by brainstorming with other students.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story. Examine the given examples carefully. They may contain details that are not mentioned in the task.
- All tasks are divided into Act Basic and Act Advanced. You need to complete all basic tasks to validate the Sprint. But to achieve maximum points and more knowledge, consider accomplishing advanced tasks also.
- Analyze all information you have collected during the preparation stages.
- Perform only those tasks that are given in this document.
- Submit only the specified files in the required directory and nothing else. Garbage shall not pass.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- The web page in the browser must open through `index.html`.
- The scripts must be written outside the HTML file - in a separate JS file (script.js).

Sprint 02 | Marathon Full Stack **> 3**


• You can always use the Console panel to test and catching errors.
• Complete tasks according to the rules specified in the following style guides:
– HTML and CSS: Google HTML/CSS Style Guide. As per section 3.1.7 Optional Tags, it doesn't apply. Do not omit optional tags, such as <head> or <body>
– JavaScript:
* JavaScript Style Guide and Coding Conventions
* JavaScript Best Practices
• The solution will be checked and graded by students like you. Peer-to-Peer learning.
• Your work may also be graded by your mentor. So, be ready for that.
• Also, the challenge will pass automatic evaluation which is called Oracle.
• If you have any questions or don't understand something, ask other students or just Google it.
ucode
Sprint 02 | Marathon Full Stack > 4

Act Basic: Task 00

NAME

Hello, JavaScript!

DIRECTORY

`t00_hello_javascript/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert()`

DESCRIPTION

Create a web page that runs two JS scripts. One of the scripts must be written inside the HTML file, and the other - outside, as a separate JS file.

The script that is inside the HTML file

- shows a message `'Hello JavaScript from inside!'`
- contains a 1-row comment with a description of the `alert` function

Keep in mind that the Google HTML/CSS Style Guide advises against the practice of mixing HTML and CSS or JS in one document. Implement it here as an exercise in order to know how it can be done, but avoid doing it in the future.

The script that is outside the HTML file, in the JS file

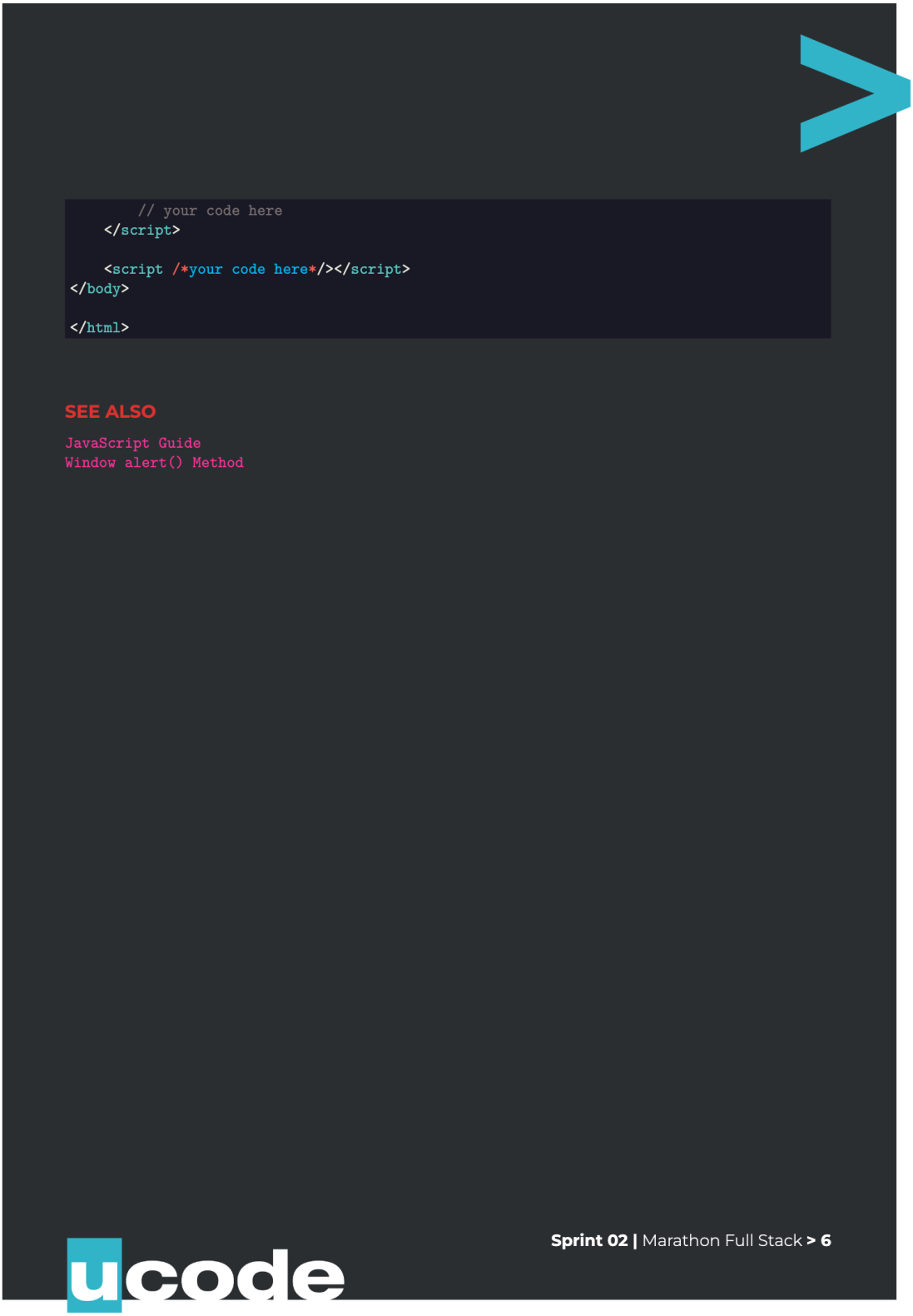
- contains a 2-row comment
- has a variable with the value `'Hello JavaScript from outside!'`
- shows a message with the value of the variable

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Hello, JavaScript!</title>
    <meta name="description" content="t00. Hello, JavaScript!">
</head>

<body>
    <h1>Hello, JavaScript!</h1>
    <script>
```

ucode

Sprint 02 | Marathon Full Stack **> 5**


// your code here
</script>
<script /*your code here*/></script>
</body>
</html>
SEE ALSO
JavaScript Guide
Window alert() Method
ucode
Sprint 02 | Marathon Full Stack > 6

Act Basic: Task 01

NAME

What type of data?

DIRECTORY

`t01_what_type_of_data/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert()`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must:

- create variables for the following data types in JavaScript and assign the appropriate values:
  - Number
  - BigInt
  - String
  - Boolean
  - Null
  - Undefined
  - Object
  - Symbol
  - Function
- display, at once, all the variable names and their data types in the following format: `variable_name is data_type\n` with `alert()` method

Note: in this task `typeof` will help you. One of the outputs may surprise you, because there is a known error in the JS language. Don't be scared and read SEE ALSO.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>What type of data?</title>
```

ucode

Sprint 02 | Marathon Full Stack **> 7**

```
  <meta name="description" content="t01. What type of data?">
</head>

<body>
  <h1>What type of data?</h1>

  <script src="js/script.js"></script>
</body>

</html>
```

SEE ALSO

JavaScript Fundamentals Data types
String Interpolation in JavaScript

Act Basic: Task 02

NAME

Superhero name maker

DIRECTORY

`t02_superhero_name_maker/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), prompt(), RegExp.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
Make a script that generates superhero names based on input.
The script must:

- prompts the user to enter input three times:
  1. to enter an animal name: `What animal is the superhero most similar to?`
     - Input requirements: length <= 20, only one word that contains only letters
  2. to enter gender: `Is the superhero male or female? Leave blank if unknown or other.`
     - Input requirements: accepts only `male`, `female`, or blank (not case sensitive)
  3. to enter age: `How old is the superhero?`
     - Input requirements: length <= 5, only digits, cannot start with a zero
- checks input for validity using regular expression (also known as regex)
- if the input is not valid, displays an error message using `alert` and stops executing
- generates a description for the superhero depending on the entered gender and age:
  - `boy` if `male` + younger than 18
  - `man` if `male` + at least 18
  - `girl` if `female` + younger than 18
  - `woman` if `female` + at least 18
  - `kid` if gender was left blank + younger than 18
  - `hero` if gender was left blank + at least 18
- displays `The superhero name is: [enteredAnimal]-[description]!`

So, for example, if the user entered: "bat", "Male", "25", the message will be:
`The superhero name is: bat-man!`

ucode

Sprint 02 | Marathon Full Stack **> 9**

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Superhero name maker</title>
  <meta name="description" content="t02. Superhero name maker">
</head>

<body>
  <h1>Superhero name maker</h1>

  <script src="js/script.js"></script>
</body>

</html>
```

SEE ALSO

Making decisions in your code - conditionals
Regular expressions

ucode

Sprint 02 | Marathon Full Stack **> 10**

Act Basic: Task 03

NAME

What kind of idiom?

DIRECTORY

`t03_what_kind_of_idiom/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), prompt(), Number.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must:

- call a `prompt()` method and take a number from 1 to 10 as an input value
- check that the input value is a number, and exactly from 1 to 10. If the input value will be not 1-10 - the `prompt()` method must ask for a number again
- show an idiom with `alert()` method

The idiom must depend on the input value in the following way:

1 - `Back to square 1`
2 - `Goody 2-shoes`
3 or 6 - `Two's company, three's a crowd`
4 or 9 - `Counting sheep`
5 - `Take five`
7 - `Seventh heaven`
8 - `Behind the eight-ball`
10 - `Cheaper by the dozen`

Note: You must use a `switch` statement for implementation. The conditional operator `if` is FORBIDDEN for this task.

Sprint 02 | Marathon Full Stack **> 11**

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>What kind of idiom?</title>
  <meta name="description" content="t03. What kind of idiom?">
</head>

<body>
  <h1>What kind of idiom?</h1>

  <script src="js/script.js"></script>
</body>

</html>
```

SEE ALSO

JavaScript Switch Statement
Window prompt() Method
JavaScript Number isFinite() Method

ucode

Sprint 02 | Marathon Full Stack **> 12**

Act Basic: Task 04

NAME

Numbers

DIRECTORY

`t04_numbers/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`prompt(), console.log(), String.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must:

- call `prompt()` and take the numbers for the beginning and end of a range
- contain a function that:
  - takes two number variables (inclusive range), and prints suitable descriptions for all numbers in the range to the Console panel. Descriptions:
    - 'number' is even
    - 'number' is a multiple of 3
    - 'number' is a multiple of 10
  - has default range of `1 - 100`

Look at the EXAMPLE of how the result may look like.

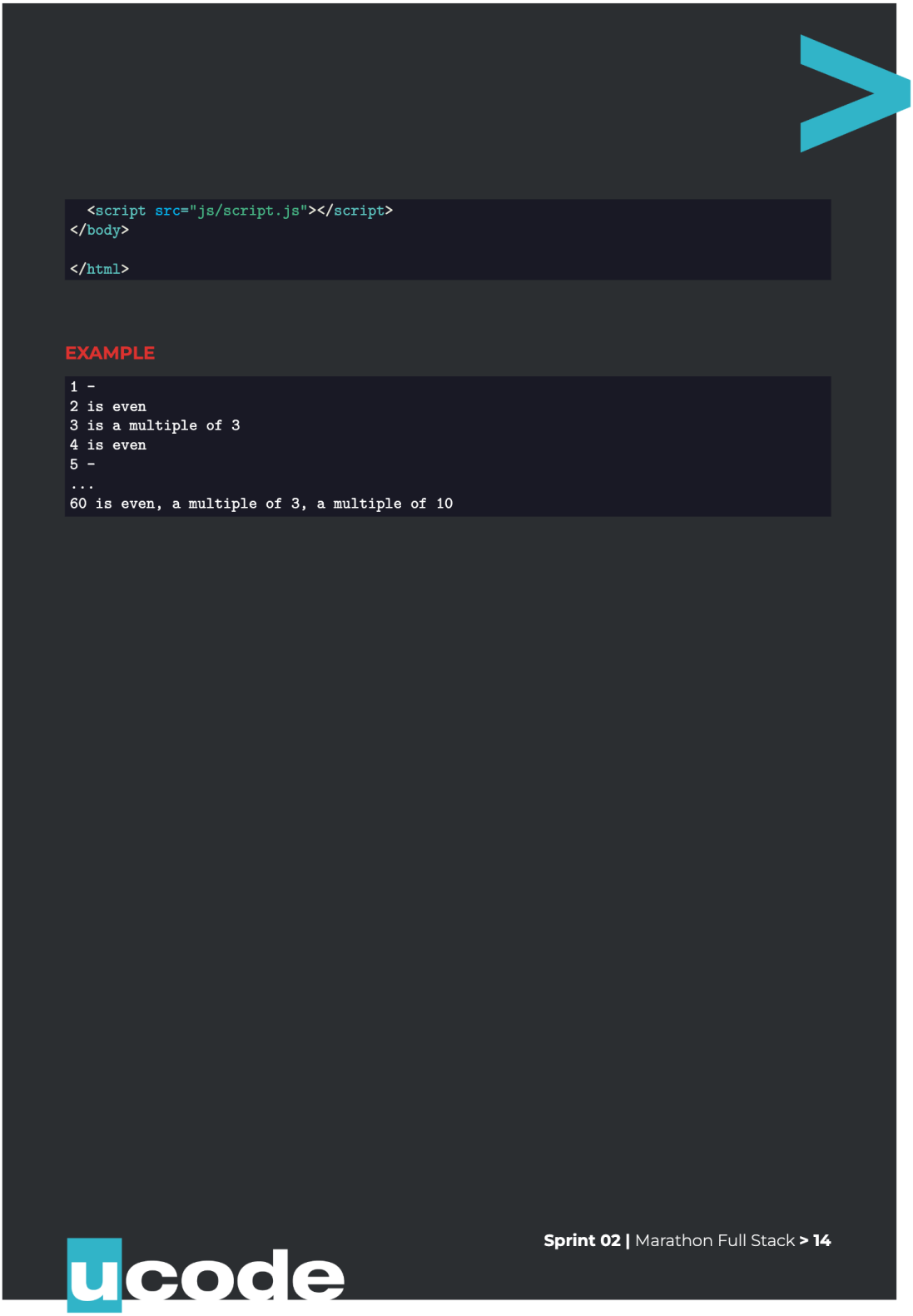
SYNOPSIS

```
checkDivision(beginRange, endRange)
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Numbers</title>
  <meta name="description" content="t04. Numbers">
</head>

<body>
  <h1>Numbers</h1>
```

ucode

Sprint 02 | Marathon Full Stack **> 13**


<script src="js/script.js"></script>
</body>
</html>
EXAMPLE
1 -
2 is even
3 is a multiple of 3
4 is even
5 -
...
60 is even, a multiple of 3, a multiple of 10
ucode
Sprint 02 | Marathon Full Stack > 14

Act Basic: Task 05

NAME

Total price

DIRECTORY

`t05_total_price/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`Number.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
Imagine that you're shopping online, and every time you add something to your cart, this function is called. The script must contain a function that:

- takes three parameters:
  - Number of items
  - The price per item
  - The current total of the price
- returns the total order sum

Display and track the result in the Console panel.

You can test your function using the test.js file written in the EXAMPLE. It is appropriate to use default parameter in this task. Add more test cases of your own.

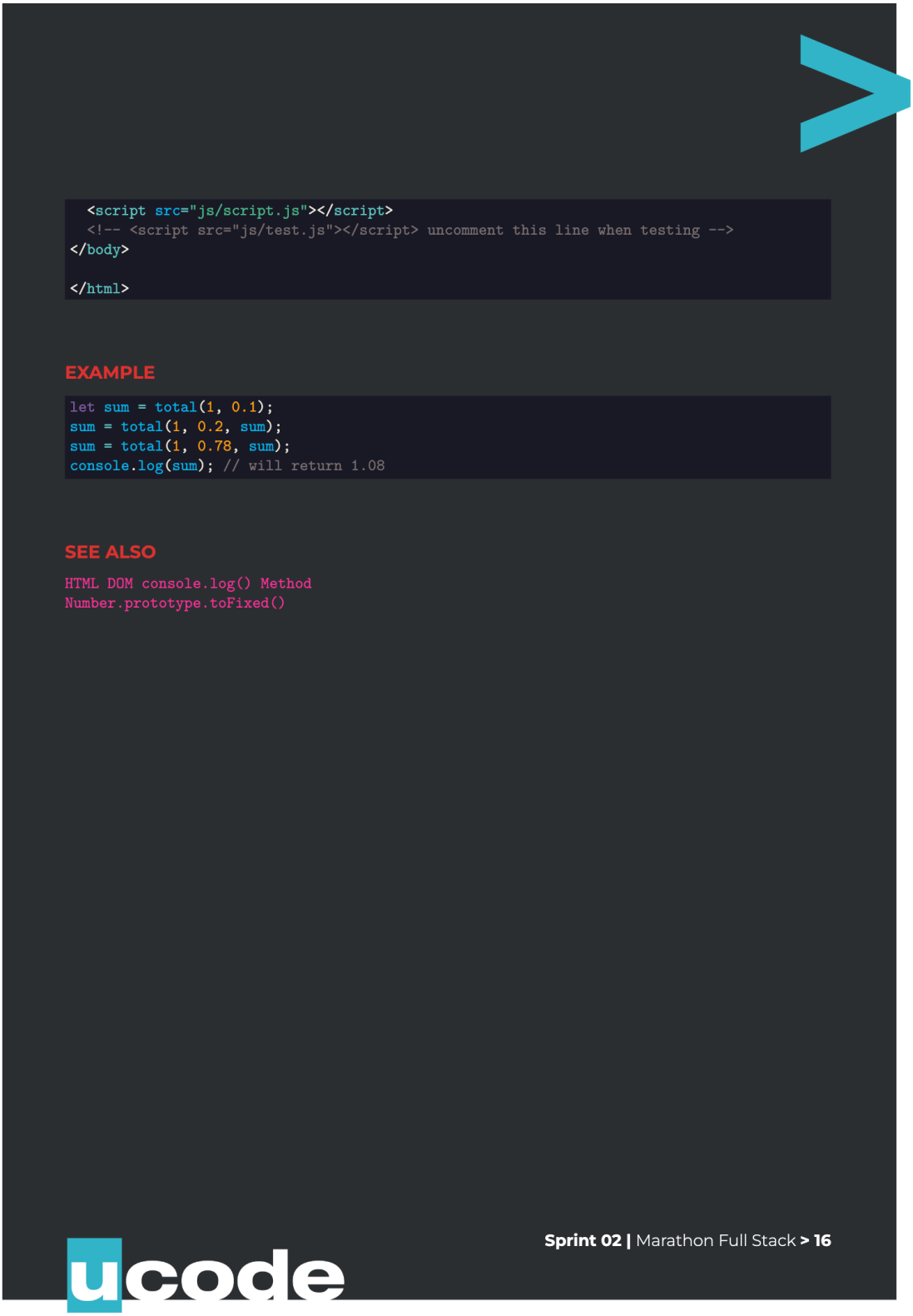
SYNOPSIS

```
total(addCount, addPrice, currentTotal) : number
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Total price</title>
  <meta name="description" content="t05. Total price">
</head>

<body>
  <h1>Total price</h1>
```

ucode

Sprint 02 | Marathon Full Stack **> 15**



```
  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

EXAMPLE

```
let sum = total(1, 0.1);
sum = total(1, 0.2, sum);
sum = total(1, 0.78, sum);
console.log(sum); // will return 1.08
```

SEE ALSO

HTML DOM console.log() Method
Number.prototype.toFixed()

ucode

Sprint 02 | Marathon Full Stack **> 16**

Act Basic: Task 06

NAME

Greeting

DIRECTORY

`t06_greeting/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), prompt(), console.log(), isNaN(), String.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must:

- prompt the user to enter their first name and last name
- check if the input is valid
- capitalize the first letter of the first and last name if it is not
- use `alert()` and the `Console` panel to greet the user using their full name
- display `Wrong input!` both to the `Console` panel and using `alert()` if a line contains a digit or other incorrect input

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Greeting</title>
  <meta name="description" content="t06. Greeting">
</head>

<body>
  <h1>Greeting</h1>

  <script src="js/script.js"></script>
</body>

</html>
```

ucode

Sprint 02 | Marathon Full Stack **> 17**

Act Basic: Task 07

NAME

Date

DIRECTORY

`t07_date/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`String.*, Array.*, Date.*, Object.*`

DESCRIPTION

Create a function `getFormattedDate()` that takes a date and formats it into a particular way as shown in the EXAMPLE.

Your JS file with the function should be included in the HTML file written in the SYNOPSIS. The test file in the EXAMPLE section is an example of how to test your function. Also, see the SYNOPSIS for the function prototype.

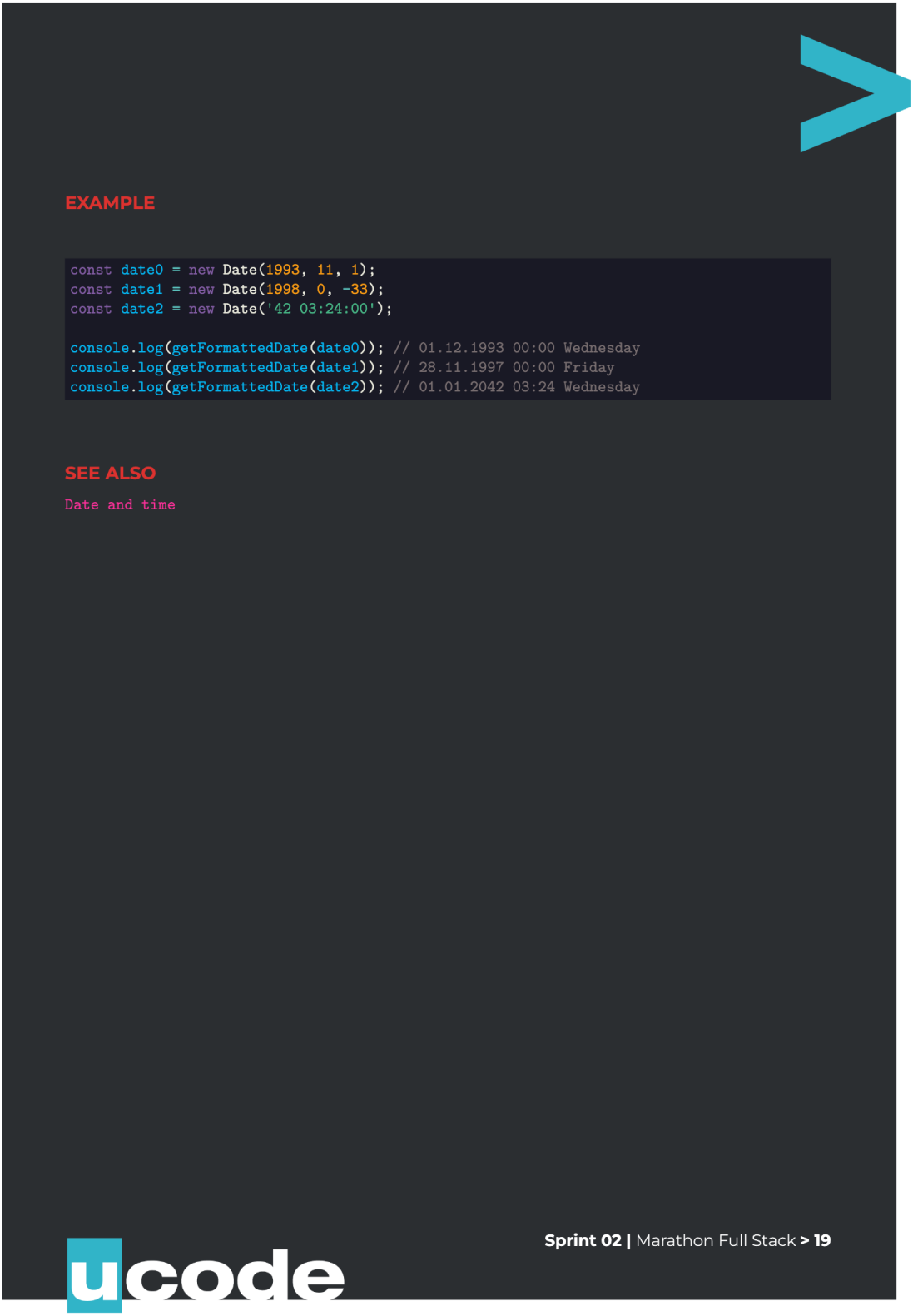
SYNOPSIS

```
getFormattedDate(dateObject) : string
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Date</title>
  <meta name="description" content="t07. Date">
</head>

<body>
  <h1>Date</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

ucode

Sprint 02 | Marathon Full Stack **> 18**



EXAMPLE

```
const date0 = new Date(1993, 11, 1);
const date1 = new Date(1998, 0, -33);
const date2 = new Date('42 03:24:00');

console.log(getFormattedDate(date0)); // 01.12.1993 00:00 Wednesday
console.log(getFormattedDate(date1)); // 28.11.1997 00:00 Friday
console.log(getFormattedDate(date2)); // 01.01.2042 03:24 Wednesday
```

SEE ALSO

Date and time

ucode

Sprint 02 | Marathon Full Stack **> 19**

Act Advanced: Task 08

NAME

Place all

DIRECTORY

`t08_place_all/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`Array.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must contain a function that:

- sorts a number array depending on whether the numbers are odd or even
- places all:
  - even numbers on the left
  - odd numbers on the right

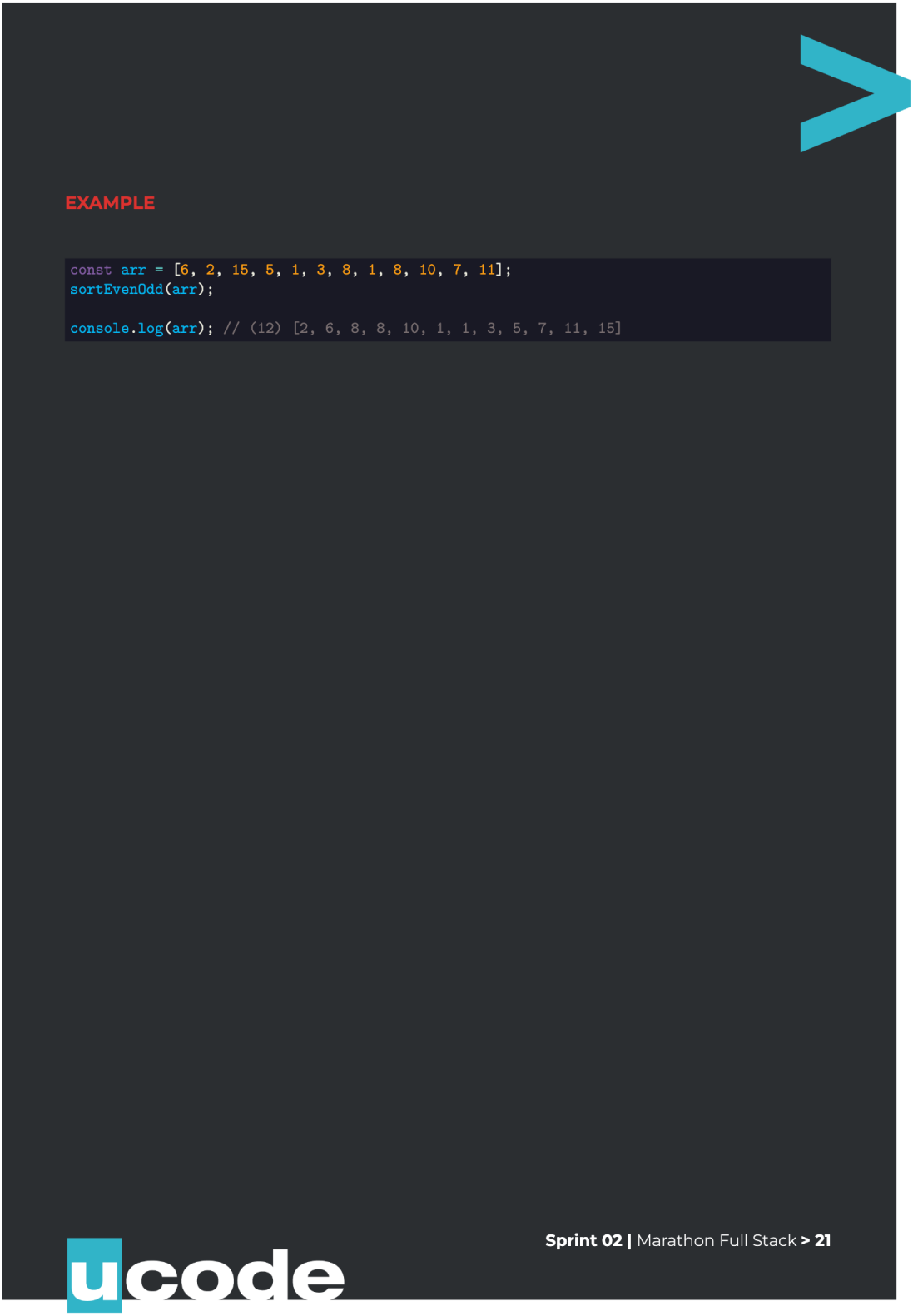
See the EXAMPLE.
Note: Don't use a temporary array.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Place all</title>
  <meta name="description" content="t08. Place all">
</head>

<body>
  <h1>Place all</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

ucode

Sprint 02 | Marathon Full Stack **> 20**


EXAMPLE
const arr = [6, 2, 15, 5, 1, 3, 8, 1, 8, 10, 7, 11];
sortEvenOdd(arr);
console.log(arr); // (12) [2, 6, 8, 8, 10, 1, 1, 3, 5, 7, 11, 15]
ucode
Sprint 02 | Marathon Full Stack > 21

Act Advanced: Task 09

NAME

Clone of Steve Rogers

DIRECTORY

`t09_clone_of_steve_rogers/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`Object.*`

BEFORE YOU BEGIN

You need to understand how to create a full duplicate of the object. To do so you need to create a new object and replicate the structure of the existing one by iterating over its properties and copying them on the primitive level.

DESCRIPTION

Create a JS file that will be included in the HTML page written in the SYNOPSIS.
The script must contain a function that makes a copy of the `user` object and its properties.

You can test your function using the test.js file written in the EXAMPLE. Add more test cases of your own.
Note: a copy of an object is not the same thing as a link to an object.

SYNOPSIS

```
function copyObj(obj) : obj
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Clone of Steve Rogers</title>
  <meta name="description" content="t09. Clone of Steve Rogers">
</head>

<body>
  <h1>Clone of Steve Rogers</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>
```

ucode

Sprint 02 | Marathon Full Stack **> 22**

```
</html>
```

EXAMPLE

```
const user = {
  name: 'Steve',
  surname: 'Rogers',
  age: 101,
  city: 'New York'
};

console.log(user);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}
let cpy = copyObj(user);
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}


user.name = 'John';
console.log(user);
// {name: "John", surname: "Rogers", age: 101, city: "New York"}
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 101, city: "New York"}

cpy.age = 59;
console.log(user);
//{name: "John", surname: "Rogers", age: 101, city: "New York"}
console.log(cpy);
// {name: "Steve", surname: "Rogers", age: 59, city: "New York"}
```

Act Advanced: Task 10

NAME

Word for word

DIRECTORY

`t10_word_for_word/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`String.*, Array.*, Object.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS and tested using a test JS file, an example of which is available in the EXAMPLE section.

Implement several functions that manipulate the object `obj`. This object has the property `words`. A word is a string separated by a single space. Whitespace is not a word.

Create the following functions:

- `addWords(obj, wrds)` - add a string with words to the object's property
- `removeWords(obj, wrds)` - remove specified words from the object's property
- `changeWords(obj, oldWrds, newWrds)` - change one or more words in the object's property

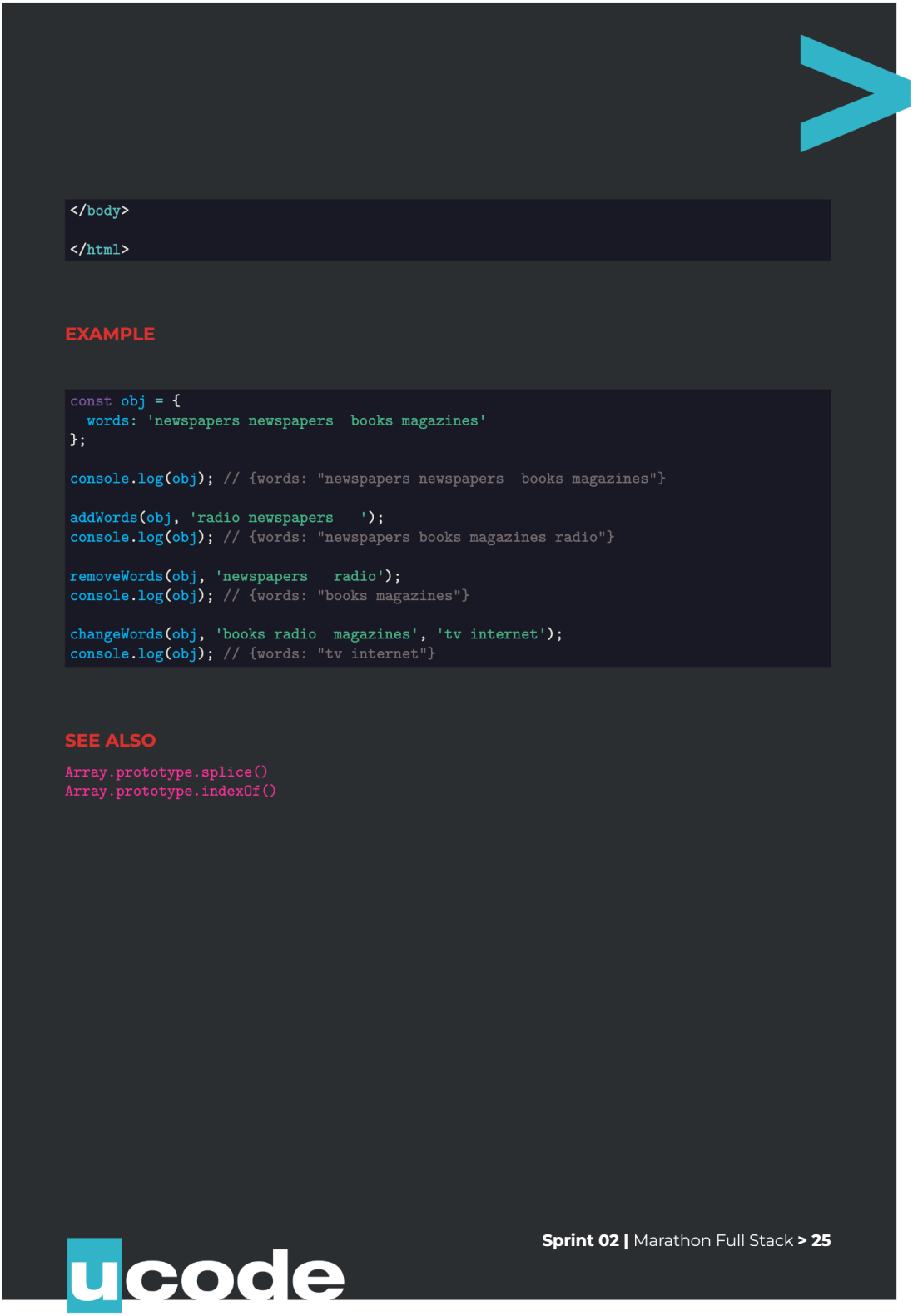
Clear duplicates and spaces in the returned object.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Word for word</title>
  <meta name="description" content="t10. Word for word">
</head>

<body>
  <h1>Word for word</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
```

ucode

Sprint 02 | Marathon Full Stack **> 24**



```
</body>

</html>
```

EXAMPLE

```
const obj = {
  words: 'newspapers newspapers  books magazines'
};

console.log(obj); // {words: "newspapers newspapers  books magazines"}

addWords(obj, 'radio newspapers   ');
console.log(obj); // {words: "newspapers books magazines radio"}

removeWords(obj, 'newspapers   radio');
console.log(obj); // {words: "books magazines"}

changeWords(obj, 'books radio  magazines', 'tv internet');
console.log(obj); // {words: "tv internet"}
```

SEE ALSO

Array.prototype.splice()
Array.prototype.indexOf()

ucode

Sprint 02 | Marathon Full Stack **> 25**

Act Advanced: Task 11

NAME

Brackets

DIRECTORY

`t11_brackets/`

SUBMIT

`index.html, js/script.js, js/test.js`

ALLOWED FUNCTIONS

`console.log(), String.*`

DESCRIPTION

Create a script with a function `checkBrackets(str)` that checks if brackets match.
A string without brackets is to be considered invalid.

Check only `(` and `)` brackets. The function returns the minimum number of brackets that must be added to make the string correct.

If the parameter is invalid (not a string or doesn't contain `(` and `)` brackets), the function returns `-1`.

Also, create tests (Chai and Mocha(bdd)) to check:

- minimum 5 incorrect cases with different data types
- minimum 10 correct cases

EXAMPLE

```
console.log(checkBrackets('1)()(())2(()')); // 2
console.log(checkBrackets(NaN)); // -1
```

SEE ALSO

Chai
Mocha-JavaScript test framework

ucode

Sprint 02 | Marathon Full Stack **> 26**

Act Advanced: Task 12

NAME

Hulk Closure

DIRECTORY

`t12_hulk_closure/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`prompt(), String.*`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS.
The script must contain a function that concatenates two strings in two different ways:

- if a function is called with two strings, the result is a concatenated string
- if a function is called with one string, the result is a function that prompts for a second string

Also, a function must have the property `count` that counts the sub function call.

You can test your function using the test.js file written in the EXAMPLE. Add more test cases of your own.

In this task, you must use the `Closure` concept of JavaScript.

SYNOPSIS

```
concat(string1, string2) : string
concat(string1) : func1
func1.count : number
```

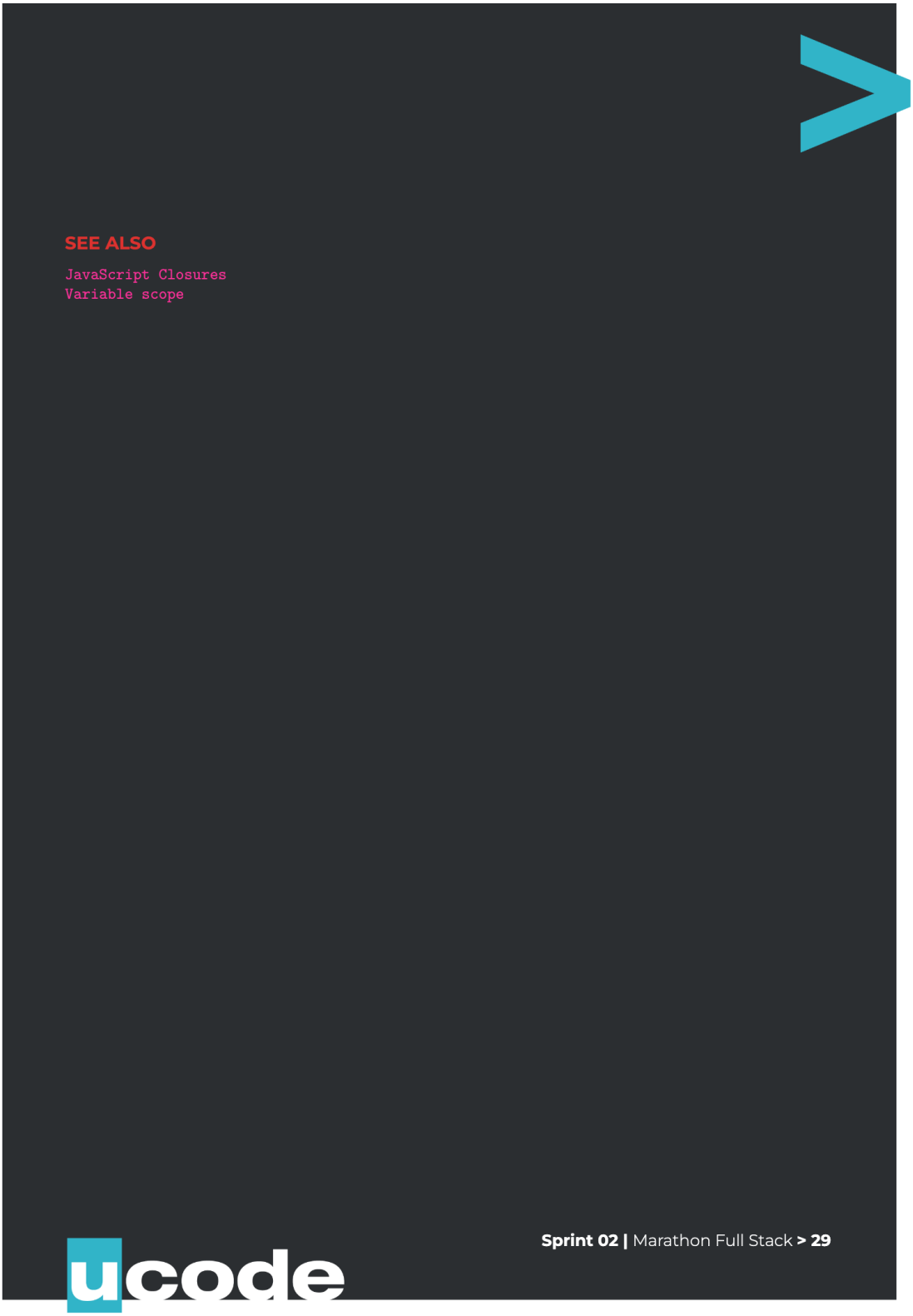
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Hulk Closure</title>
  <meta name="description" content="t12. Hulk Closure">
</head>

<body>
  <h1>Hulk Closure</h1>
```

Sprint 02 | Marathon Full Stack **> 27**

```
  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

EXAMPLE

```
let phrase1 = concat("Hulk", "smash!");
let output = phrase1;
console.log(output); // Hulk smash!

let phrase2 = concat("Leave");
output = phrase2();
// a prompt appears. Enter "Hulk alone!" into the prompt

console.log(output); // Leave Hulk alone!
console.log(phrase2.count); // 1

output = phrase2();
// a prompt appears. Enter "me alone, please!" into the prompt

console.log(output); // Leave me alone, please!

output = phrase2();
// a prompt appears. Enter "HULK ALONE!" into the prompt

console.log(output); // Leave HULK ALONE!
console.log(phrase2.count); // 3

let phrase3 = concat("Go");
output = phrase3();
// a prompt appears. Enter "away!" into the prompt

console.log(output); // Go away!
console.log(phrase3.count); // 1
console.log(phrase2.count); // 3

/* Result in Console panel:
Hulk smash!
Leave Hulk alone!
1
Leave me alone, please!
Leave HULK ALONE!
3
Go away!
1
3
*/
```


SEE ALSO
JavaScript Closures
Variable scope
ucode
Sprint 02 | Marathon Full Stack > 29

Act Advanced: Task 13

NAME

Calculator

DIRECTORY

`t13_calculator/`

SUBMIT

`index.html, js/script.js`

ALLOWED FUNCTIONS

`alert(), setTimeout()`

DESCRIPTION

Create a JS file that will be included into the HTML page written in the SYNOPSIS and tested using a test JS file, an example of which is available in the EXAMPLE section.

In your JS file, create a function-constructor that creates a calculator object with the following methods:

- `init(num)` - set value for calculating
- `add(num)` - addition
- `sub(num)` - subtraction
- `mul(num)` - multiplication
- `div(num)` - division
- `alert()` - alert-message with the current result after a 5 seconds delay

The calculator has the property: `result`.

Alert from the `Calculator.alert()` method must show with a 5 seconds delay.

In this task, you can use the `Closure` and `Chaining` concepts of JavaScript.

SYNOPSIS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Calculator</title>
  <meta name="description" content="t13. Calculator">
</head>
```

ucode

Sprint 02 | Marathon Full Stack **> 30**

```
<body>
  <h1>Calculator</h1>

  <script src="js/script.js"></script>
  <!-- <script src="js/test.js"></script> uncomment this line when testing -->
</body>

</html>
```

EXAMPLE

```
const calc = new Calculator();

console.log(
    calc
        .init(2)
        .add(2)
        .mul(3)
        .div(4)
        .sub(2).result // 1
);

calc.alert();
```

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio (macOS)
- ScreenToGif - screen, webcam, and sketchboard recorder with an integrated editor (Windows)

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.

Sprint 02 | Marathon Full Stack **> 32**