# **Exercise: qUnit**

This document defines the lab exercise assignments for the   
**"QA Front-End Automation" Course @ SoftUni**

## Create Functions for Testing

Before we start testing with qUnit, we need to **create a few functions to use as our test objects**. Follow these steps to write your functions for testing:

1. **Create your VS Code folder and file for functions:**

* First, **create a folder** named **"function\_and\_tests";**
* Then **open this folder**. Choose "File", then "Open Folder" from the menu:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

* **Right click on "EXPLORER" tab and choose "New File…"**:

Картина, която съдържа текст, екранна снимка, софтуер, дизайн

Описанието е генерирано автоматично

1. **Create a JS file** and name it **"test\_functions.js"**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

1. **Open the file and write the functions we will test later. All function will be in one document, so we can easily export them later:**

* Let's **create a simple "sum" function that returns a number**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

* Under the first function, **write a second one that checks if a given number is even and returns a boolean**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

* **The third function will calculate the factorial of a number and return number**. Note that the function uses recursion, meaning it calls itself. All negative number parameters will aways return 1:

Картина, която съдържа текст, Шрифт, екранна снимка, номер

Описанието е генерирано автоматично

* The next **function is for finding palindromes and returns bool**. *A palindrome is a word, number, or phrase that reads the same backward and forward, such as “civic” or “2002”*. In our function, we convert the given string to lowercase, then replace all non-word characters (such as ‘@’, ‘-‘, ‘$’, ‘\_’, etc.) with an empty string. To check if the word is a palindrome, we reverse the word and check if the reversed word is equal to the original:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* Also write **a function that generates the Fibonacci sequence up to the specified number of terms (n parameter) and returns an array**. *The Fibonacci sequence is a series where each number is the sum of the two preceding ones*. The function uses for-loop to push all calculated values into the array:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* Write **a function which contains a nested function and returns number**. This **function should find the nth prime number based on the given n parameter**. *Example: if the function accepts value 5 as a parameter, it should return 11 as value. The sequence of prime numbers up to 100 is 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, so the 5th prime number is 11*. There is the implementation of both functions – the main and nested one:

Картина, която съдържа текст, екранна снимка, дисплей, софтуер

Описанието е генерирано автоматично

* Write another **function that generates Pascal's triangle up to the specified number of rows**. **The function accepts a number as a parameter and returns an array of arrays (jagged array)**. *You can learn about Pascal's triangle here:* [*Pascal's triangle - Wikipedia*](https://en.wikipedia.org/wiki/Pascal%27s_triangle)*. There is also an animated graphic about it:* [*Pascal's triangle link*](https://upload.wikimedia.org/wikipedia/commons/0/0d/PascalTriangleAnimated2.gif)*.*

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* The last function is simple. **It checks if the given parameter number is a perfect square and returns a boolean**:

Картина, която съдържа текст, Шрифт, екранна снимка, Графика

Описанието е генерирано автоматично

## Export / Import functions (CommonJS)

Now, we have functions ready for testing. It is important to **export them so we can import them into another file where we will write our tests later**.

In this exercise, **we will use CommonJS for export and import**. This means **we won't need an index.js page for running the tests because we will run them directly through Node.js, not through the browser**.

**In the same file where we wrote our function, we should add another piece of code for functions exporting**. To do this, we need to **create a single JS object that will contain all of the fuctions, expect the "isPrime" function, because it is a helper (nested) function for the "nthPrime" function**.

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Note that we **use the keyword "module.exports" to export the object.** This **object will contain all our functions for testing as properties**.

Now, let's import this object into another file. First, we need to create a file:

1. Create a **JS file named "sum\_tests.js"** because we will write test for the sum function in it:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

1. Write the code to import our exported object:



**Note! We are importing only the "sum" function in this file because we will test only this function.**

1. Check if import is working properly. Use the function to get a single result and execute the code by pressing CTRL + F5 (also choose Node.js):

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

1. If the result is written in the Debug Console, the import works as expected:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

1. Now we can remove the result variable and the console.log lines. We don’t need them for function testing.

## Install qUnit

To be able to write tests, we need a test framework**. We will install QUnit:**

1. **Open the Terminal in VS Code from "View" 🡪 "Terminal":**

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

1. Write the command **"npm init -y":**

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

**IMPORTANT! Make sure that you are in the correct folder. In our case it should be the "function\_and\_tests" folder.**

1. After running the command, you should see **the "package.json" file in the EXPLORER**:

Картина, която съдържа текст, Шрифт, екранна снимка, Графика

Описанието е генерирано автоматично

1. Now, **install QUnit through npm by executing the command "npm install --save-dev qunit"**:

Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

1. Let's write our test for "sum" function:

* First, we will **create a QUnit module**. This can be seen as a named test suite for our tests:

Картина, която съдържа Шрифт, екранна снимка, текст, Графика

Описанието е генерирано автоматично

* **In the QUnit module (test suit), we will write our tests**. Let's write the **first test which will check how the function works when two positive numbers are given as parameters**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* The second test should **check when positive and negative numbers are given as parameters**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

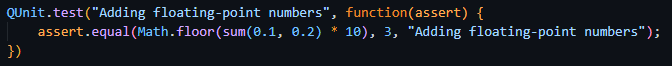
Описанието е генерирано автоматично

* Next will be for **two negative numbers**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* Lastly, we will test with **two real numbers**. Note that **comparing two real numbers in JS can be complex**, so **we need to cast them to integers and then to check if they are equal**:



1. The whole file should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. Now, it is time to **run our test**. **Open "package.json" file and make one change**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

In **the "scripts" property**, **change the value of the "test" sub-property to "qunit"**:Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

1. Save all **changes in the "package.json" file**, then open the Terminal and **execute the command "npm test <file\_name>"**, where "<file\_name" in our case should be replaced with "sum\_tests.js":



1. After running the command, **the Terminal should return a report**:

Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

1. Let's **create a test that will fail to see what to expect in this case from the report**:

* First, let's **add this test**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

It is **similar to the correct test for floating-point numbers, but we do not convert the result to integers**, so there will be a **difference between the result from the sum function and the expected one**. **The report will show one failed test**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Carefully **examine the report**. You will see:

* **The failed test highlighted in a different color**.
* The **severity marked as "failed"**.
* The actual result.
* The expected result.
* A summary of all test statuses.

## Write Tests for All Functions

Now that you're familiar with the QUnit framework and know how to use it, let's make tests for all other functions:

### Test "isEven" function

1. **Create new file named "isEven\_tests.js".**
2. **Import the "isEven" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns true when an even number is given as a parameter.
* The function returns false when an odd number is given as a parameter.
* The function returns true when zero is given as a parameter.
* **Optionally**, test with negative parameters.

1. **Run tests by executing the command "npm test isEven\_tests.js".**
2. **Check the report – all tests should pass.**

**NB! You will need to use *ok and notOk* to test bool values.**

### Test "factorial" function

1. **Create new file named "factorial\_tests.js".**
2. **Import the "factorial" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns 120 when the parameter is 5.
* The function returns 1 when the parameter is 0.
* The function returns 1 when the parameter is -1.

1. **Run tests by executing the command "npm test factorial\_tests.js".**
2. **Check the report – all tests should pass.**

### Test "isPalindrome" function

1. **Create new file named "isPalindrome\_tests.js".**
2. **Import the "isPalindrome" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns true when the parameter is "racecar".
* The function returns true when the parameter is "A man, a plan, a canal, Panama!".
* The function returns false when the parameter is "hello".
* The function returns false when the parameter is an empty string.

1. **Run tests by executing the command "npm test isPalindrome\_tests.js".**
2. **Check the report – all tests should pass.**

**NB! You will need to use 'ok' and 'notOk' to test boolean values.**

### Test "fibonacci" function

1. **Create new file named "fibonacci\_tests.js".**
2. **Import the "fibonacci" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns an empty array when the parameter is 0.
* The function returns an array with one element, 0, when the parameter is 1.
* Check the returned array when 5 is given as parameter.
* Check the returned array when 10 is given as parameter.

1. **Run tests by executing the command "npm test fibonacci\_tests.js".**
2. **Check the report – all tests should pass.**

**NB! You will need to use 'deepEqual' to compare actual and expected arrays.**

### Test "nthPrime" function

1. **Create new file named "nthPrime\_tests.js".**
2. **Import the "nthPrime" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns 2 when the parameter is 1.
* The function returns 11 when the parameter is 5.
* The function returns 29 when the parameter is 11.

1. **Feel free to write more tests for the function. The sequence of prime numbers up to 100 is 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97. Check the previous test cases and compare expected results to this sequence.**
2. **Run tests by executing the command 'npm test nthPrime\_tests.js".**
3. **Check the report – all tests should pass.**

### Test "pascalsTriangle" function

1. **Create new file named "pascalsTriangle\_tests.js".**
2. **Import the "pascalsTriangle" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns an empty jagged array when the parameter is 0.
* The function returns a jagged array with value [[1]] when given parameter is 1.
* The function returns a jagged array with value [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]] when given parameter is 5.
* The function returns a jagged array with value [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1], [1, 6, 15, 20, 15, 6, 1], [1, 7, 21, 35, 35, 21, 7, 1]] when given parameter is 8.

1. **Run tests by executing the command "npm test pascalsTriangle\_tests.js".**
2. **Check the report – all tests should pass.**

**NB! A jagged array is a normal array which contains arrays as its elements. You will need to use 'deepEqual' to compare actual and expected jagged arrays.**

### Test "isPerfectSquare" function

1. **Create new file named "isPerfectSquare\_tests.js".**
2. **Import the "isPerfectSquare" function.**
3. **Create a test suit with QUnit.module.**
4. **Write tests to check:**

* The function returns true when the parameter is 1.
* The function returns true when the parameter is 4.
* The function returns true when the parameter is 9.
* The function returns true when the parameter is 16.
* The function returns false when the parameter is 2.
* The function returns false when the parameter is 15.

1. **Run tests by executing the command "npm test isPerfectSquare\_tests.js".**
2. **Check the report – all tests should pass.**

**NB! You will need to use 'ok' and 'notOk' to test boolean values.**

## \*Asynchronous Testing

Let's also do some **asynchronous tests using the fetch API in JavaScript**. As you already know from the previous lectures and exercises, **the fetch API allows you to fetch data (to make request and to handle response) from different web APIs asynchronously**.

**IMPORTANT! You don't need the below configuration to execute tests. You might need it to run and debug functions.**

First, we need to **create another file named "async\_test\_functions.js"**. Making another file for our async functions requires **adding a configuration that will allow us to start and debug different files with F5 and CTRL + F5 shortcuts (in case we need to debug our functions)**:

1. **Go to "Run and Debug" tab** from the sidebar and **click on "create a launch.json file"**:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

1. **Choose "Node.js"** from the popup windows:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. This will create a **json file with configuration for Node.js**. You need to **change a single part of it to allow the currently opened file to be executed with shortcuts**. Check the object value of the key **"configurations"**. You will see a property named **"program"**. Its value should be something like this: **"${workspaceFolder}\\sum\_tests.js"**. So, we need to **change it to "${workspaceFolder}\\${file}"**. The whole configuration should look like the picture below:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

**NB! Don't forget to save the changes.**

1. Once you're done with this, there will be **a folder named .vscode with a launch.json file in it**:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

### Create Asynchronous Functions

Now let's create our asynchronous functions. **In the file "async\_test\_functions.js"** we have to write two async functions – one for asynchronous fetch and another to simulate an asynchronous delay.

**The first function will make a request to a given URL**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

**The second function will simulate an asynchronous delay**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

At the end, we need to export these functions to be able to import them in the test file:

Картина, която съдържа текст, Шрифт, екранна снимка, линия

Описанието е генерирано автоматично

The whole file should look like the picture below:

Картина, която съдържа текст, екранна снимка, дисплей, Шрифт

Описанието е генерирано автоматично

### 5.2. Test Asynchronous Functions

To write our test, we need to import our functions.

#### Test "fetchData" function

We will **use a web API to make a request and handle data for this function**. Let’s use the web API [Zippopotamus](https://www.zippopotam.us/). You can explore this web API to get more familiar with it.

1. Create **a file named "fetchData\_tests.js"**.
2. **Import the function for testing**:



1. Make a **test with endpoint (URL)** <https://www.zippopotam.us/bg/8000>. The response JSON should be:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

The **test should check each property key and its value**. Also, the property "places" is an array of objects, so we have to check its properties (each property key and its value). Here is the test:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

1. Make a **test with a wrong postcode**. Use the same endpoint but change the postcode to <https://www.zippopotam.us/bg/8000999>. It should returns "undefined":

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. Make a **test with wrong URL**. Change the endpoint URL to this one: <https://wwww.zippopotam.us/bg/8000> (there are four 'w' in the URL):

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. Use **the command "npm test .\fetchData\_tests.js" to execute the tests**.

**NB!!! You can group your tests in a test suit (QUnit.module).**

#### Test "fake\_delay" function

1. Create **a file named "fakeDelay\_tests.js"**.
2. **Import the function for testing**:



1. Make a single test with 1000 miliseconds:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. Execute the test with the command "npm test .\fakeDelay\_test.js".