Lancaster University

SCC 212 Advanced Programming – Weeks 8-10.

...GTCAGATCGTTCGAACGGGGCTCCAAACA...

JavaScript

NOTES:

You should attempt all four tasks here in a single program.

The testing framework will be looking for all the features in all the tasks.

REMEMBER - LOOP STRUCTURES ARE BANNED!

To stand ensure you can handle all tests, think thoroughly of various test cases.

Aim of the Exercise

Over the remaining weeks we will be applying the concepts we learnt about dynamic scripting in JavaScript. The aim is to introduce you to dynamic scripting, so as to provide you with insights into this programming paradigm. The labs are is for you to obtain experience of writing more algorithmically complex code in JavaScript. The coursework also serves as a good basis to prepare for any practical exam questions that may appear on dynamic scripting. All the task of the coursework will contribute to your portfolio assessment.

As in previous labs, you may need to refer to the lecture notes and do some independent research to complete these tasks.

Background:

We will work on the same task of DNA pattern matching we worked on in week7. As a reminder, a common task for DNA sequencing is the identification of patterns of nucleotides - the individual building blocks of both RNA and DNA. A DNA sequence is a represented as a succession of letters that indicate the order of the nucleotides within a DNA molecule. The letter used in the sequences are A, C, T and G. You can read more about this on this Wikipedia page: https://en.wikipedia.org/wiki/Nucleic_acid_sequence. In this coursework, we will focus on a simple pattern identification task. You'll write a function in JavaScript that finds patterns in given sequences then you will use that function to apply concepts of asynchronous execution, which will be introduced in week 9.

BUT

There is a special rule for these tasks - **Loop structures are not allowed** - this means you cannot use any `for`, `while`, or `do/while`, or any other form of these structures. Instead, you should make extensive use of the Array functions (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) and Object functions (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) to search, match and refine your data.

The Mozilla JavaScript documentation can help you with this, take a look at the links above, along with the documentation link here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference . You will have to do some reading around the documentation to be able to complete these tasks!

Task 1: Frequency Counts and Sequence Matching (Week 8)

You've been asked to produce the **counts** and **offsets** of particular patterns. For this, write a function `find(data,patterns)` that takes two inputs; `data` which is a string of characters and `patterns`, which is an array of patterns to search for.

The function should return an object containing the frequency and offset of each pattern in the following structure:

{

frequencies: {*pattern1*: *freq1, pattern2*: *freq1*,… },

offsets: { *pattern1*: [*offert1*, *offset2*,…], *pattern2*: [*offset1*,*offset2*,…],… }

}

The following table contains some example of input and expected output.

data	**patterns**	**output**
'AAA'	['AA', 'AC']	{ frequencies: { AA: 2, AC: 0 }, offsets: { AA: [0, 1], AC: [ ] } }
'AAC'	['AA', 'AC']	{ frequencies: { AA: 1, AC: 1 }, offsets: { AA: [0], AC: [1] } }

'CCAAATT'	['AA','AT','CCA']	{
		frequencies: { AA: 2, AT: 1, CCA: 1 },
		offsets: { AA: [2, 3], AT: [4], CCA: [0] }
		}

Notes:

- The returned object MUST have the mentioned structure. Otherwise, auto-testing script will fail to award marks. The returned object MUST contain two properties `frequencies` and `offsets`, DO NOT change the name of these properties.
- The offset of a pattern is the number of characters from the start of the data string to the start of the matching sequence.
- The patterns and data can be of arbitrary lengths.
- A pattern could exist in overlapping characters, e.g. 'AAA' contains two patterns of 'AA' at offsets 0 and 1.
- In the supplied .zip file, you should find a JavaScript file named 'cw.js'. This file is a template to base your implementation on. **You should not modify the declared variables or the functions signatures**

in this file. To run the file, execute 'node cw.js' from the command line or terminal in the program's folder.

Task 2: Asynchronous execution using callbacks (Week9)

In this task, you've been asked to read two files; 'file.data', which contains lines of sequences (data) and 'file.pattern', which contains patterns to look for. Write a function `analyse1(datafile, patternfile)` that takes the names of the two files and analyses them for pattern matching and offset finding using the `find()` function you developed in task 1.

The function `analyse1()` works as the following. It asynchronously reads the contents of the 'file.pattern' file. Once the reading is successful, it asynchronously reads the content of the 'file.data' file and calls the `find()` function for each data line in the data file.

The results of analysing the files should be stored in the `results1` object. This object should contain pairs of data lines from the 'file.data' file as keys and the object returned by the find function as a value. For example, if the data file contains AAA and AAC (each in a separate line) and the pattern file contains AA and AC (each in a separate line), then the object result1 should be

```
results1 = {
  AAA: { frequencies: { AA: 2, AC: 0 }, offsets: { AA: [Array], AC: [] } },
  AAC: { frequencies: { AA: 1, AC: 1 }, offsets: { AA: [Array], AC: [Array] } }
}
```

In the supplied .zip file, you should find the 'file.pattern' and 'file.data' files. These are just sample files to use. You can create your own data and pattern files (but with the same structure) to test your code.

Task 3: Asynchronous execution using promises (Week9)

This task is similar to task 2. The difference is that you've been asked to use promises to process files. For this purpose, you've been asked to do the following:

- Write a function, `readFilePromise()`, that takes a file name and returns a promise object. The promise should read the file content and returns its contents when fulfilled.
- Write a function `analyse2(datafile, patternfile)`, that reads and analyses the files for pattern matching (similar to analyse1()). The function should use the `readFilePromise` to read the files' contents and `find()` to find matches.

The results of analysing the files should be stored in the `results2` object, which has the same structure as the object `results1` described in Task2 (see above).

Task 4: Asynchronous execution using async/await (Week9/10)

In this task, you've been asked to write `analyse3(datafile, patternfile)` function that is similar to `analyse2()`. The difference is to replace the usage of promises with the `async/await`. You still need to use the `readFilePromise()` function to return a promise when reading files.

The results of analysing the files should be stored in the `results3` object, which has the same structure as the object `results1` described in Task2 (see above).

Test cases

You need to thoroughly test you code using various test cases. Here are some:

Data file	**Patterns file**	**output**

ACCA
AACGTCGACG
	C
	CG
		{
		ACCA: { frequencies: { C: 2, CG: 0 }, offsets: { C: [1,2], CG: [] } },
		AACGTCGACG: {
		frequencies: { C: 3, CG: 3 },
		offsets: { C: [2,5,8], CG: [2,5,8] }
		}
		}

TTGGTTGGTTGG
CCCTTGCGGTT
CCAAGGGGGTT
CCCGGGCGGTT
	TT
	GG
		{
		TTGGTTGGTTGG: {
		frequencies: { TT: 3, GG: 3 },
		offsets: { TT: [0,4,8], GG: [2,6,10] }
		},
		CCCTTGCGGTT: {
		frequencies: { TT: 2, GG: 1 },
		offsets: { TT: [3,9], GG: [7] }
		},
		CCAAGGGGGTT: {
		frequencies: { TT: 1, GG: 4 },
		offsets: { TT: [9], GG: [4,5,6,7] }
		},
		CCCGGGCGGTT: {
		frequencies: { TT: 1, GG: 3 },
		offsets: { TT: [9], GG: [3,4,7] }
		}
		}

TTGGTTGGTTGG
CCCTGGCGGTT
CCACTGGGGGGTT
CCCGTGGCGGTT
	TG
	GT
	CTG
		{
		TTGGTTGGTTGG: {
		frequencies: { TG: 3, GT: 2, CTG: 0 },
		offsets: { TG: [1,5,9], GT: [3,7], CTG: [] }
		},

		CCCTGGCGGTT: {
		frequencies: { TG: 1, GT: 1, CTG: 1 },
		offsets: { TG: [3], GT: [8], CTG: [2] }
		},
		CCACTGGGGGGTT: {
		frequencies: { TG: 1, GT: 1, CTG: 1 },
		offsets: { TG: [4], GT: [10], CTG: [3] }
		},
		CCCGTGGCGGTT: {
		frequencies: { TG: 1, GT: 2, CTG: 0 },
		offsets: { TG: [4], GT: [3,9], CTG: [] }
		}
		}

Marking

Marks will be awarded according to the following table:

Task	Marks /80
Task 1: Frequency Counts and Sequence Matching	30
Task 2: Asynchronous execution using callbacks	20
Task 3: Asynchronous execution using promises	20
Task 4: Asynchronous execution using async/await	10

Submission instruction

Your solution of the four tasks should be submitted as a sinlge .js file to the JavaScript submission point by week 10 (Friday 4pm). Please log in to the SCC.212 Moodle pages and find the JavaScript submission point in the coursework submission section.

Portfolio contribution

This piece of work will contribute to 80% of the Functional programming and Dynamic Scripting coursework which is due in week 10. It will carry marks for correct core functionality. It will be partly automatically-marked by using various test cases submitted to your code.

REMEMBER - LOOP STRUCTURES ARE BANNED!