# Cardiff School of Computer Science and Informatics

## Coursework Assessment Pro-forma

| | |
|---|---|
| **Module Code:** | CMT120 |
| **Module Title:** | Fundamentals of Programming |
| **Lecturers:** | Federico Liberatore, Martin Chorley, Natasha Edwards |
| **Assessment Title:** | Programming Challenges |
| **Date Set:** | 23rd November 2020 |
| **Submission date and Time:** | 14th December 2020 at 9:30AM |
| **Return Date:** | 20th January 2021 |

---

This assignment is worth 30% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here:
`https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf`

---

## Submission Instructions

All coursework should be submitted via upload to Learning Central.

| **Description** | **Type** | **Name** |
|---|---|---|
| *Cover sheet* | `.pdf` file | `[Student number].pdf` |
| *Python Code* | 1 `.py` file | `[Student number].py` |
| *JavaScript Code* | 1 `.js` file | `[Student number].js` |

Any code submitted will be run on a system equivalent to the laptops provided to the students, and must be submitted as stipulated in the instructions above. The code should run without any changes being required to the submitted code, including editing of filenames.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a deduction of 25% for that question or question part.

Staff reserve the right to invite students to a meeting to discuss coursework submissions.

---

## Assignment

To complete this coursework, you must complete a set of programming challenges in Python and/or JavaScript.

Each challenge can be awarded a maximum of 12.5 marks. The final mark is determined by the best eight challenges. Therefore, perfectly solving four exercises will give you 50 marks (pass), and perfectly solving at least eight exercises will give you 100 marks. Also, you cannot get more than 100 marks. An exercise is solved perfectly only if high-quality functional code is submitted in both Python and JavaScript. Providing high-quality functional code in only one programming language results in a lower mark (i.e., 10 marks). Therefore, you can still pass the coursework by only completing problems in one language.

The challenges are described in detail below, and you are also provided with a set of test cases that will check whether your code produces the required output or not. You should make sure that your submitted code passes the supplied tests to ensure it functions correctly. However, please note that your code may be tested against further/different test cases. You should therefore ensure that you try to cover all corner cases and additional inputs and that your code still functions correctly.

You will find template code for the assignment on Learning Central. This provides two folders, `python` and `js`. Inside each folder you will find a `template.{js/py}` file, in which you should complete your solutions. You will also find a `test_template.{js/py}` file containing the test cases that will check your code's functionality, along with a folder of test data required for some of the tests. You are also supplied with a `Readme.md` file containing detailed instructions on how to run the test cases to check your code.

In the templates, the functions' interfaces are given but the functions' bodies are empty. Solve the exercises by correctly filling in the functions' bodies. It is forbidden to change the functions' interfaces. However, new functions can be defined to support the solution of the exercises. These functions must have names that are different from those already present in the templates.

In all the exercises, you can assume that the inputs are provided in the appropriate format. Therefore, error-checking is not needed.

You will be given marks for solving each problem, and extra marks for completing the solution in both programming languages. Further marks will be awarded for solution style and quality. The mark scheme is described in further detail later.

### Exercise 1: Smallest Fraction Terms

Complete the function 'reduceFraction' that takes two positive integers as its parameters. The parameters represent the numerator and denominator of a fraction. The function reduces the fraction to lowest terms and then returns both the numerator and the denominator of the reduced fraction as integers.

**Python:** The numerator and denominator should be returned as a tuple.

**JavaScript:** The numerator and denominator should be returned as elements 0 and 1 of a 2-item array.

**Example:** reduceFraction(12,15) returns (4, 5) in Python and [4,5] in JavaScript.

### Exercise 2: Magical Dates

A magic date is a date where the day multiplied by the month is equal to the last two digits of the year.

**Example:** 10/6/1960 is a magic date because 6 times 10 is 60, which is equal to the last two digits of 1960.

Complete the function 'isMagicDate' that takes three positive integers as its parameters, day, month, and year, respectively, and returns True if the date is a magic date, or False otherwise.

### Exercise 3: Find All Sublists

A sublist is a list that makes up part of a larger list. A sublist may be a list containing a single element, multiple elements, or even no elements at all.

**Example:** `[1]`, `[2]`, `[3]` and `[4]` are all sublists of `[1, 2, 3, 4]`. The list `[2, 3]` is also a sublist of `[1, 2, 3, 4]`, but `[2, 4]` is not a sublist of `[1, 2, 3, 4]` because the elements 2 and 4 are not adjacent in the longer list. The empty list is a sublist of any list. As a result, `[]` is a sublist of `[1, 2, 3, 4]`. A list is a sublist of itself, meaning that `[1, 2, 3, 4]` is also a sublist of `[1, 2, 3, 4]`.

Using the above definition of a sublist, complete the function 'sublist' that takes a list as its only parameter, and returns a list containing every possible sublist of the input list.

**Example:** Given the input

```
['a', 2, (0,"zero")]
```

the function should return

```
[[], ['a'], [2], [(0,"zero")], ['a', 2], [2, (0,"zero")],
['a', 2, (0,"zero")]]|
```

The order of the elements of the list returned is not important. However, the order of the elements inside of each list should reflect the order in the original list. For example, `['a', 2]` is correct, while `[2, 'a']` is not.

## Exercise 4: English to Pig Latin Translator

Pig Latin is a language game or argot in which English words are altered, usually by adding a fabricated suffix or by moving the onset or initial consonant or consonant cluster of a word to the end of the word and adding a vocalic syllable to create such a suffix (Wikipedia).

The following rules are used to translate English into Pig Latin:

- If the word begins with a consonant (including 'y'), then all letters at the beginning of the word, up to the first vowel (excluding 'y'), are removed and then added to the end of the word, followed by 'ay'.
  **Example:** 'computer' becomes 'omputercay' and 'think' becomes 'inkthay'.

- If the word begins with a vowel (not including 'y'), then 'way' is added to the end of the word.
  **Example:** 'algorithm' becomes 'algorithmway' and 'office' becomes 'officeway'.

Complete the function 'pigLatin' that takes a string as the only parameter and return a string representing its Pig Latin translation.

The function should correctly handle uppercase letters and punctuation marks such as commas, periods, question marks and exclamation marks. You can assume that only the first letter can be uppercase and that punctuation marks can only be at the end of the word.

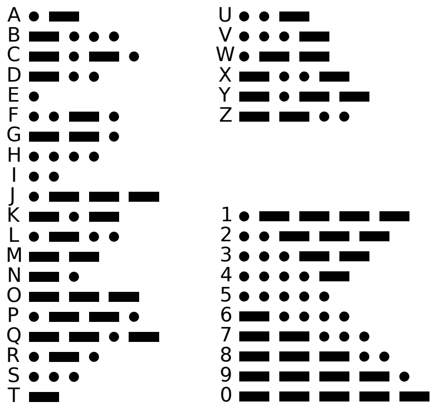
**Example:** if an English word begins with an uppercase letter, then its Pig Latin representation should also begin with an uppercase letter and the uppercase letter moved to the end of the word should be changed to lowercase. For example, 'Computer' should become 'Omputercay'. If a word ends in a punctuation mark, then the punctuation mark should remain at the end of the word after the transformation has been performed. For example, 'Science!' should become 'Iencescay!'.

**Exercise 5: Morse Code Encoder**

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes (or dits and dahs). Morse code is named after Samuel Morse, an inventor of the telegraph (Wikipedia).

Complete the function 'morseCode' that takes a string of letters and numbers as the only parameter and returns a string with its Morse code representation.

Use a period '.' to represent a dot, and a minus sign '-' to represent a dash. The mapping from letters and numbers to dashes and dots is illustrated in the following figure.



The function should leave a single space between each sequence of dashes and dots and it should ignore any characters that are not letters or numbers.

**Example:** The Morse code for 'Hello, World!' is shown below:

```
.... . .-.. .-.. --- .-- --- .-. .-.. -..
```

*Hint:* use dictionaries to represent the Morse code table.

**Exercise 6: Spelling Out Numbers**

Complete the function 'int2Text' that takes an integer between 0 and 999 as its only parameter, and returns a string containing the English words for that number.

**Example:** if the parameter to the function is 142 then 'int2Text' should return "one hundred fourty two".

*Hint:* Use one or more dictionaries to implement your solution rather than large if/elif/else constructs.

**Exercise 7: No Functions without Comments**

Complete the function ‘missingComment’ that takes a string containing a filename as its only parameter. The filename should be a source file. The function reads the file, identifies functions that are not immediately preceded by a comment and returns a list of their name (represented as strings).

**Python:** For the purposes of this exercise, assume that any line that begins with ‘def’, followed by a space, is the beginning of a function definition. Also, assume that, when the function has a comment, the comment character, ‘#’, will be the first character on the previous line.

**JavaScript:** For the purposes of this exercise, assume that any line that begins with ‘function’, followed by a space, is the beginning of a function definition. Also, assume that, when the function has a comment, the single-line comment symbol, ‘//’, will be the first character on the previous line.

**Exercise 8: Justify any Text**

Complete the function ‘consistentLineLength’ taking two parameters, a filename, and a maximum length (i.e., a strictly positive integer number). The function opens the file, reads every line, and returns a list of strings where every string represents a line that is filled as much as possible without exceeding the given maximum length.

**Example:** consider a file containing the following lines from ‘Alice’s Adventures in Wonderland’.

```
Alice was
beginning to get very tired of sitting by her
sister
on the bank, and of having nothing to do: once
or twice she had peeped into the book her sister
was reading, but it had
no
pictures or conversations in it,"and what is
the use of a book," thought Alice, "without
pictures or conversations?"
```

The corresponding output for `length = 50` would be:

```
['Alice was beginning to get very tired of sitting',
'by her sister on the bank, and of having nothing',
'to do: once or twice she had peeped into the book',
'her sister was reading, but it had no pictures or',
'conversations in it, "and what is the use of a',
'book," thought Alice, "without pictures or',
'conversations?"']
```

You do not need to deal with multiple paragraphs of text. That is, you can group together words from different paragraphs. Finally, you can assume that no word is longer than the maximum length.

### Exercise 9: Knight's Challenge

Complete the function 'knight' that takes three parameters: an initial position, a final position, and a number of moves. The function returns True if a knight on an empty chessboard can get to the final position from the starting position in at most the given number of moves; otherwise, the function returns False. Useful facts:

- A chessboard is an 8x8 square board.
- Each cell of the chessboard is identified by its coordinates: a letter from 'a' to 'h' that identifies the column, and a number from 1 to 8 that identifies the row. The positions are provided to the functions using this format.
- The chessboard contains only the knight, which is located at the specified initial position.
- A knight may move two squares vertically and one square horizontally, or two squares horizontally and one square vertically.

**Examples:**

- `knight('a1', 'c5', 2)` $\mapsto$ True.
- `knight('c2', 'e3', 3)` $\mapsto$ True.
- `knight('c6', 'h1', 1)` $\mapsto$ False.

*Hint:* You may want to use recursion.

**Exercise 10: War of Species**

Complete the function ‘warOfSpecies’ that takes as input a list of strings of the same length, representing a rectangle grid. The characters in the list represent the cells of the grid. The cells can take three possible values:

- `'X'`, representing an individual of the species X.
- `'O'`, representing an individual of the species O.
- `'.'`, representing an empty cell.

**Example:**

```
["X.......", "........", ".......O"]
```

is a 3x8 grid with an `'X'` in position (1,1) and a `'O'` in position (3,8).

The configuration provided in the list represents the current state of an environment with two competing species. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. The function should return a list of strings representing the next state of the environment according to the following rules:

- An empty cell becomes non-empty if it is surrounded by at least two individuals of the same species. In particular, it becomes an individual of the most frequent species in its neighbourhood. In case of a draw between the species, the cell remains empty.
- A non-empty cell becomes empty if it is surrounded by more than six non-empty cells, regardless of their species.
- A non-empty cell becomes empty if it is surrounded by less than three members of its species.
- A non-empty cell becomes empty if it is surrounded by more members of the opposite species than members of its species.
- In any other circumstances, a cell does not change its value.

You can assume that the format of the input list of strings is correct.

**Example:** The initial environment

```
["XX......", "XX....O.", ".....OOO"]
```

becomes

```
["XXX.....", "XXX..OOO", "XX....O."]
```

## Learning Outcomes Assessed

- LO1: Use high-level programming languages to complete programming tasks.
- LO2: Demonstrate familiarity with programming concepts, simple data-structures and algorithms.

---

## Criteria for assessment

Each exercise can be awarded a maximum of 12.5 marks. The final mark is determined by the best eight exercises. Therefore, perfectly solving four exercises will give you 50 marks (pass), and perfectly solving at least eight exercise will give you 100 marks. You cannot get more than 100 marks.

Each exercise is marked for both function (10 marks max) and style/quality (2.5 marks max). You will gain 7.5 marks for solving an exercise in one language, and the full 10 functional marks for solving it in both languages.

The functional part of the submission is automatically marked by scripts that run the completed function against a set of test cases. An exercise is considered solved in one language only if all the tests in the benchmark are passed successfully. For each exercise, the following marking scheme is applied:

- Correctly solving the exercise in both languages: 10 marks.
- Correctly solving the exercise in only one language: 7.5 marks.
- Failing to solve the exercise in either language: 0 marks.

The quality of the code is assessed by the following criteria:

| High quality and style (50-100%, 1.25-2.5 marks per exercise) | Low quality and style (0-50%, 0-1.25 marks per exercise) |
|---|---|
| Code is elegant<br>Code has no redundancies<br>Code is well commented<br>Code is perfectly modular (i.e., appropriate functions and/or classes defined)<br>Code makes smart use of built-in language features and classes. | Code is messy or overly verbose<br>Code has multiple redundancies and repetitions<br>Code is lacking in meaningful comments<br>Code is disorganised<br>Code does not make use of language features |

Only fully working code is evaluated for style and quality and the above table assumes that an exercise has been correctly solved in both languages. Therefore, an exercise correctly solved in only one language is awarded half of the style/quality marks and an exercise that has not been correctly solved in any language is not awarded any style/quality mark.

---

## Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on the return date via Learning Central and/or email.

The feedback from this assignment will be useful for your second programming assignment, and will also be relevant for any future programming tasks.