Portfolio_summative

January 10, 2025

Computer Programming and Algorithms: Summative Assignment 1

The learning outcomes for this course are:

1. Describe the basic principles, concepts and terminology used in computer programming.
2. Apply the fundamental tools of computer programming to write simple computer programs.
3. Translate high-level "problem statements" into simple algorithms and implement these algorithms as a computer program.

This assignment will evaluate learning outcome 2. Learning outcomes 1 and 3 will be assessed by the individual programming assignment (and viva). This assignment is worth **20% of your overall mark** for this course.

Code Portfolio

This assignment will consist of 4 short exercises, each of which has been already introduced in our prior lab sessions as portfolio exercises 1-4. To assist with the marking process, we're asking you to combine your solutions to these exercises into a single file ("summative.py"). You should download this file from Blackboard, editing it to include your code from the portfolio exercises. It is possible you will need to make a small number of edits to make this work correctly, however, the core programming exercises are identical-please re-use your code from previous exercises in this submission.

Submission

This assignment is due **January 23rd 2025 at 1pm GMT.** Your code should be submitted as a single file (summative.py) via Blackboard.

Assessment

For this assignment, we are assessing your ability to write code that produces the correct output. This will be assessed by the test scripts included in the file "summative.py". Each assignment is worth 5 points (out of a total of 20). If you pass all the tests for an assignment, you will receive all five points. If you fail any of the tests for an assignment, you will lose those five points. Please feel free to inspect the test code, however, you **must not** edit the code in the test functions-we will review all submissions with a tool to check for changes to these sections of the code and any modifications will be considered attempts at cheating.

It is important to emphasize that the marks for this assignment will be entirely based on the output of our test scripts; we will not be giving marks for readability or partial credit for code that *almost* passes a test or for your *working* as you may get in e.g. a maths test. In part, this is because we will evaluate these aspects of programming with other assignments. However, in part, this is also because writing code that produces the correct output is a fundamental capability in programming;

most code you write will predominantly interact with other bits of code and these interactions will not work if the output is not exactly correct.

This may seem arbitrary and even a bit nitpicky at times ("I lost 5 marks because I had an extra space on a print statement…"), however, part of being a good programmer is understanding that computers are nitpicky, and learning techniques and tools for overcoming this. Here, we've given you full access to the test scripts used to evaluate your code, so the arbitrariness of the output is transparent and fully specified; if your code passes the test scripts, then you will receive full marks for the assignment. If it doesn't, you will receive zero marks for the assignment.

Libraries and AI tools

The use of AI to produce any code submitted in this assignment is *prohibited* (category 1 in UoB's policy on AI use). Use of AI tools will be considered contract cheating and dealt with via UoBs standard procedure for plagiarism. All libraries necessary for this assignment have already been included at the top of the "summative.py" file. You may not use additional libraries beyond those.

Exercise 1 - Comparisons

Edit the code below to print the statements only when they apply.

```
message = ''

if '???':
    message += "%.2f is greater than %.2f\n" % (a, b)
if '???':
    message += "%.2f is less than %.2f\n" % (a, b)
if '???':
    message += "%.2f is equal to %.2f\n" % (a, b)
if '???':
    message += " %.2f is within 0.1 of %.2f" % (a, b)

return message
```

Exercise 2 - Solving Kepler's equation

If you're trying to rendezvous with a space station, it helps to know where it is, at any given time. Sadly that's not straightforward, as it's governed by Kepler's equation:

$M = E - esinE$

$M = \frac{2\pi(t-t_0)}{T}$ is the *mean anomaly* that takes time since lowest altitude and coverts it to an angle. E is the *eccentric anomaly* that tells you where you actually are on the orbit ellipse… so you need to find E given M. But you can't re-arrange this to anything usable of the form $E = \ldots$, so it needs solving numerically.

Step forward *fixed point iteration* which aims to find an x such that $f(x) = x$ *i.e.* the fixed point of function f, where the "output" of f is unchanged from its "input". The algorithm is: 1. Guess an x 2. Calculate $f(x)$ 3. Set x equal to $f(x)$. 4. Repeat from 2. Think about it: if it ever converges such that x stops changing, you have found a fixed point.

Here we'll use a `for` loop to iterate for a fixed number of times.

Re-arrange Kepler's equation into the form $E = f(E)$ and implement it over `???` in the code below. Also implement the loop with sufficient iterations to converge to within 0.01% of the mean anomaly.

Optional extra: implement an early stopping criterion for the tolerance and test with multiple values of M.

Exercise 3 - Assigning Grades based on marks

Student	Question 1	Question 2	Question 3
Martin	6	1	4
Arthur	3	8	4
Hemma	7	4	5
Josh	4	7	3

The table above contains some marks for a test I gave to the CPA instructors. I'd like to turn this set of marks into a set of overall grades for each person. To do that, I'd like to you to write some code to create a new list containing the total marks achieved by each person. i.e for Martin, this would be a total = 6+1+4 = 11.

Next, I'd like you to write some code to calculate the mean and standard deviation of the total marks each person achieved. Recall that the standard deviation of some data is given by:

$\sigma = \sqrt{\sum_{i=1}^{N} \frac{(x_i-\mu)^2}{N}}$, where μ is the mean, and N is the number of students.

In Python, you can calculate a square root using the function `math.sqrt()`.

Next, you should write some code to normalise the marks by subtracting the mean score from each mark. i.e. here the mean mark is 14, so Martin's normalised mark should become 11-14=-3.

Then, I'd like to turn these marks into a grade according to the formula:

Normalised Mark	Grade
mark < -1 * standard deviation	Fail
-1 * standard deviation <= mark < 0	C
0 <= mark < 1 * standard deviation	B
1 * standard deviation <= mark	A

Finally, we need to save the output to a file ("grade_file.csv"). This file should contain 6 lines and adhere to the format:

```
mean, <mean>
standard deviation, <std. deviation>
martin, <grade>
arthur, <grade>
hemma, <grade>
josh, <grade>
```

with values in `<>` replaced by the values calculated by your code. e.g. if you calculate the mean to be 14.0, then the first line should read `mean, 14.0`.

Exercise 4 - Encoding and decoding messages

A Caesar cipher is a simple substitution cipher in which letters are shifted along by a certain amount. For example, with a shift of 3, the letter 'a' becomes 'd' and 'b' becomes 'e'. With a shift of 4, 'a' becomes 'e' and 'b' becomes 'f'.

Task 1

Write a function that implements the cipher. Your function should be called "caesar_cipher" and accept two mandatory inputs-the text to be encoded and the shift used. You can use the Python function ord() to convert a character into it's corresponding ASCII number, and the function chr() to convert back from an ASCII number into a character.

Task 2

Write a program which can decrypt messages encoded by a Caesar cipher. However, while you're given the message, we won't tell you the shift. To make this problem easier, we'll restrict ourselves to messages that contain words from a pre-defined list. This will list be provided to you as a txt file (word_list.txt) which you must read and use in your program.

This means we can decode a message with a simple brute force strategy. Our decoding algorithm will work as follows:

Break the message into words. For each word, loop over all possible shifts. For each shift, decode the word and see if the decoded word is in the wordlist. If it is, add the decoded word to the decoded message. If after trying all possible shifts, there is no match, your program should print an error. We won't worry about the possibility of a word matching multiple shifts- it's OK to assume the first match is the correct one.

Messages can only contain lower case characters-no number, symbols or upper case letters allowed. If an invalid message is provided, your code should print an error message.

Sample Inputs and Outputs:

- Message: khoor, Output: hello
- Message: Khoor, Output: Invalid character in message
- Message: puppy, Output: can't decode word: puppy
- Message: khoor khoor, Output: hello hello

[ ]: