COURSEWORK SPECIFICATION
ECM1400 – Introduction to Python
Module Leader: Dr. Abhijit Chatterjee
Academic Year: 2025/26

Title: **Algorithms Assessment**

Submission deadline: **The deadline is 10th November at 12:00 Midday.**

This assessment contributes **30%** of the total module mark and assesses the following **intended learning outcomes**:

- design an algorithm, using sequence, iteration and selection;
- write, compile, test, and debug a computer program;
- explain how a program written in a procedural language is translated into a form that allows it to be executed on a computer;
- systematically test your programs;
- document software to accepted standards;
- design an algorithm, using a divide and conquer strategy;
- demonstrate familiarity with basic numerical and discrete algorithms;
- systematically break down a problem into its components;
- understand and choose appropriate programming techniques.
- analyse a problem and synthesise a solution;
- use technical manuals and books to interpret specifications and technical errors.

This is an individual assessment and you are reminded of the University's regulations on collaboration and plagiarism. You must avoid plagiarism, collusion, and any academic misconduct behaviours. Further details about academic honesty and plagiarism can be found at https://ele.exeter.ac.uk/course/view.php?id=1957.

Use of AI tools in AI-Minimal Assessments.

Assessment Title: Algorithms Assessment

Module Code and Name: ECM1400 – Introduction to Python.

The University of Exeter is committed to the ethical and responsible use of Generative AI (GenAI) tools in teaching and learning, in line with our academic integrity policies where

the direct copying of AI-generated content is included under plagiarism, misrepresentation and contract cheating under definitions and offences in TQA Manual Chapter 12.3.

This assessment falls under the category of AI-Minimal in the University's Guidance on use of Gen AI in Assessment.

This means: You may use AI tools for checking spelling and grammar mistakes only, with no other impact on the structure or content of the assessment. This is because using GenAI tools outside of these uses prevents fair assessment of your ability to achieve module learning outcomes.

When writing your assessment, you must never use AI tools:

1. For uses other than checking your spelling and grammar.

2. To translate more than a word or short phrase into English.

3. To upload sensitive or identifying material to an AI tool

4. To present material that has been generated by AI as your own work or the work of someone else.

When submitting your assessment, you must:

1. Check the box during the submission process, that confirms you have adhered to the university's academic conduct policy and the expectations on use of GenAI in your assessment brief.

Instructions

1 Flowchart and pseudo code:

This question is designed to test the programming concepts around algorithm design and text handling that are covered in the first two weeks. This question requires flowchart and pseudo code representations, no Python code. Please write explanations of your thinking and reasoning.

1 Specification

An airline wants to implement a new booking system for passengers. The booking system needs to take account of the various possibilities when choosing seats on the plane, based on the party size. It needs to store a representation of the plane and to group the passengers together whilst that is possible. The steps required to implement the booking system are broken down broken down into four sections each worth 5 marks.

1.1 Creating a representation of the plane

Create a flow chart of a function (nested algorithm) called create_plane that sets up a list of lists representing 18 rows each containing 6 seats. It should have a loop that initially sets each seat to empty.

Create a pseudocode function called print_plane that prints a representation of the plane to the command line (as lines of text), with the aisle represented between seats 3 and 4 in each row, and the emergency exits between rows 10 and 11.

[5 Marks]

1.2 Choose seats

Create a flow chart for a function called seat_chooser that asks a user making a booking to pick their seat based on a series of questions and then shows the chosen seat(s) using print_plane. The questions asked will include how many passengers are required and their preference for front middle or back of the plane. The seats should be arranged in the correct section filling up the seats in a logical order.

[5 Marks]

1.3 Premium upgrade

Create a pseudocode module, called premium_seat for a user who has booked a premium ticket to choose their seat specifically. It should provide a representation of the plane where rows are marked with a letter and seats marked from 1 to 6, so seats can be given as a letter number combination, and allow the user to enter the specific seat required. It should also make a check to see the seat isn’t occupied and record the choice of inflight drink from the options (tea, coffee, fruit juice, champagne). There should be an additional check that a passenger is over 18 if champagne is chosen.

[5 Marks]

1.4 Menu system

Create a pseudocode module that will show a menu system that will provide access to the functions from sections 1.1 to 1.3 for each booking. It should check the number of passengers and check that there are seats available, before showing the menu for standard or premium seats. It should loop until the plane is full. After a booking is completed there should be a ticket printed to the command line showing all the choices made during the booking and the seat number(s) in the correct letter+number format.

[5 Marks]

2 Coding:

This question will require the application of fundamental programming constructs, functions, and control flow.

2.1 Specification

These exercises explore the foundations of programming in that they specify small units of functionality to be implemented. Small units of functionality are very compatible with unit testing and test-driven development so these questions will be marked using automated tests where possible. A sample of code for unit testing has been provided to give an example for the developer (you) to validate that the code is compatible with the testing infrastructure. The specification has been broken down into four exercises; each worth 5 marks. Proper comments on are expected on all the codes. 1.5 marks are going to be deducted if codes are submitted without comments.

2.1.1 ex1.py

Write a function called kelvin that takes a temperature (as an integer or float) and a to_kelvin argument. If the to_kelvin argument is true the function should return the temperature given plus 273.3 (the Celsius temperature converted to Kelvin), rounded to the nearest integer. If however the to_kelvin argument is false it should return the temperature – 273.3 rounded to the nearest integer (converting Kelvin back to Celsius). If to_kelvin is not specified it should default to True. This function should be saved in a file called ex1.py

The example data for this function is:

```
from ex1 import kelvin

assert kelvin (20.8 , True) == 294
```

[5 Marks]

2.1.2 ex2.py

Write a function called email addresses that takes two list arguments called first and last and an argument called a domain that has a default value '@exeter.ac.uk'. The function should be declared in a module saved as ex2.py.

Implement the function to use inputs from the corresponding items in the arguments first and last to return a list of suggested email addresses in lowercase that follow the convention of containing the first letter of the name in first, followed by a full stop and then the entire corresponding name from the argument last, and finally the domain.

The example data for this function is:

```
from ex2 import email_addresses

first = ["Jack" , "Bellatrix"]

last = ["Sparrow" , "Lestrange"]

expected_result = ["j.sparrow@exeter.ac.uk" , "b.lestrange@exeter.ac.uk"]

assertemail_addresses(first, last) == expected_result
```

[5 Marks]

2.1.3 ex3.py

Write a function called palindrome_detector that takes a string argument of a word or phrase. The function should be declared in a module saved as ex3.py. The function should return true if the word or phrase contains the same letters forwards as backwords. The function should be able to deal with any capitalisation and spaces or punctuation encountered.

The example data for this function is:

```
from ex3 import palindrome_detector

phrase = "A dog! A panic in a pagoda!"

expected_result = True

assert palindrome_detector(phrase) == expected_result
```

[5 Marks]

2.1.4 ex4.py

Write a function called obfuscate that takes one argument called text. The function should obfuscate the text in the argument using the process defined below in that exact order to systematically change the text and make it difficult to read. The function should be declared in a module saved as ex4.py.

• Swap every instance of the word ’the’ to ’and’ and ’and’ to ’the’

• Take every third letter and make it uppercase

• Reverse the letters in every fifth word

• Apply a shift cypher with key 1 to encrypt every other word The example data for this function is:

```
from ex4 import obfuscate

input_text = " Hello the world "

expected_result = " HeLlo boE wOrlD"

assert obfuscate(input_text) == expected_result
```

[5 Marks]