****

VCE Software Development Unit 3

Unit 3 Outcome 1: Programming skills test

Name: Garv Shah

**Outcome Statement**

*On completion of this unit, the student should be able to interpret teacher-provided solution requirements and designs, and apply a range of functions and techniques using a programming language to develop and test working software modules.*

**Task Conditions**

**Allowed resources:** Teacher-provided specifications, Python reference material

**Time allocated for this task:** 2 lessons per module + 1 lesson for test

**Marks allocated:** 25 each module

**Submitting work:** All files related to this task must be uploaded to your folder on Office365 OneDrive shared folder **SAC1** **Your name**. This folder will be cleared at the conclusion of class.

**Task Outline**

You are required to produce a folio of modules, using only the provided requirements and designs. You will need to debug and test the module to ensure it functions as expected. You must ensure it is error-free using the test table provided.

The marking scheme can be found on the last page.

Software Development Test

Total: 25 marks

**INTERNAL DOCUMENTATION**

**Question 1**

Explain the code below. In your explanation, provide an example of its output.

The code below generates three random passwords, each using different methods in the code.

Method 1:

The first variable determines the length of the password, by randomly selecting an integer from 18 – 20. The symbols variable also gets all the possible characters that can be used, using the string library to get a range of characters: (abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~)

A password is then generated by selecting a random character from the string above and adding it to the password, repeating this process for the randomly selected length above.

Method 2:

The second method is similar, but instead of using the string library to get a choice of characters, it is manually defined inside the code. As seen, 4 variables, lower, upper, numbers and symbols are defined, all combined to create the “all” variable, which is analogous to the symbols variable from the previous method, albeit with a few minor differences in the characters chosen. A fixed length of 16 is set for the password in this method, and a different function is used to concatenate the randomly selected character. The sample() function is used for random sampling without replacement, and as such, gets 16 random characters from the “all” variable and concatenates them into the final password.

Method 3:

This method defined the string of characters in a very similar way to the first method, resulting in the exact same string, as s.ascii\_letters is equivalent to s.ascii\_lowercase + s.ascii\_uppercase. This method also has a fixed length of 16 for the password, and uses the same sample() function to get a random selection from the available characters. As such, it is a combination of the first and second method, combining their input characters and random selection method. Note that the first method can contain repeats of characters, as the choice may be the same as the previous, but the second two methods will never have a repeated character.

Example Output:

Password Gen 1: .c3@s,ls\_\<4xT2yihZc

password gen 2: V0.gLIQpwq7xk6Ky

password gen 3: mCU;SoT1.'`u)Rj+

import secrets

import string as s

import random

length = secrets.choice(range(18,21))

symbols = s.ascii\_letters + s.digits + s.punctuation

password = ""

for i in range(length):

password += "".join(secrets.choice(symbols))

print("Password Gen 1: " + password)

lower = "abcdefghijklmnopqrstuvwxyz"

upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

numbers = "0123456789"

symbols = "[]#()\*;/,.\_-"

all = lower+upper+numbers+symbols

length2 = 16

password2 = "".join(random.sample(all,length2))

print("password gen 2: "+password2)

lower = s.ascii\_lowercase

upper = s.ascii\_uppercase

digits = s.digits

symbols = s.punctuation

all3 = lower + upper + digits + symbols

length3 = 16

password3 = "".join(random.sample(all3, length3))

print("password gen 3: " + password3)

**Question 2**

Briefly explain the concept of internal documentation. Provide two reasons why internal documentation is important.

Internal documentation is the documentation of code found within the codebase, for use with other team-members of programmers. This commonly takes the form of comments within the code, explaining how certain processes work and why they were implemented in the way they were. They should not explain everything, but give a brief idea of how things work, especially when code isn’t very easily readable. Internal documentation is important as it helps others understand your code, and work with you. Being faced with a huge project with no documentation is incredibly daunting, and it can be almost impossible to understand everything just by looking at the code itself. Explanations of what code does helps others looking at the code understand what is happening, why it is happening, and lets them better understand the thoughts behind the creator. This being said, internal documentation is not only useful for others, but for yourself too. It saves a lot of time in the long run, because when you visit a project again after a while of not having touched the code, it can be incredibly hard to understand what is happening with the spaghetti mess of code you wrote at 2am just trying to get it to work in any way possible (or any code, it’s hard to understand any code you write if you don’t explain it). Documenting what you have done saves you from having to go through the whole codebase again to start working on it; instead, you can just read some of the comments you made last time to quickly understand what a block of code does so you can get started right away. Finally, internal documentation also serves as a much faster and efficient way to test code in an organisation, and by yourself. The documentation helps organise the code, and if there is a bug that has been found, it helps pinpoint where the functionality is going wrong, and what code is responsible, without having to go through the computer’s logic in your head. This helps the writer of the code, as the organisation allows for quick and easy debugging, but also allows other testers to easily see what section of the code is going wrong to maybe even fix it themselves. Overall, internal documentation is a vital part of any project, which not only helps keep the code organised and readable for you, but also allows others to understand your code with less difficulty, being integral in group environments.

**DESIGN TOOLS**

**Question 3**

Write a data dictionary for the code below.

years = [2000,2001,2000,1999,2002,1998,2001,2000,1997,2001,2001,2003]

ages = []

for year in years:

ages.append(2019-year)

total = 0

for age in ages:

total += age

average = total / len(ages)

print(ages)

print('Average age is',average)

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| years | List or Array | A list of random years, with various repeats. This is most likely the year that certain people were born in. |
| ages | List or Array | An initially empty array, which is used to store the calculated ages of each birth year from the years array |
| total | integer | Used to store the total age of all the people. This is analogous to sum(ages) |
| averages | float | The mean of all the ages, using the formula for the mean of a dataset, it divides the sum of ages with the length of ages |

**TESTING**

**Question 4 (a)**

What is the purpose of testing code?

Programmers test their code to make sure their code works correctly, in the way they intend it to. This helps to prevent and find bugs before they are shipped to production, or even after the fact, to ensure that the user experience works like it was intended. The process of testing can also help the programmer find ways in which their program can be improved, as to “idiot-proof” their code. The person who created the product knows how to use it, but others may not, or it may not work on their hardware or in their scenario. To make sure the program is robust enough to work in any situation, or guide a user to make it work in scenarios where it doesn’t, testing is required to iron out bugs and create a nice user experience.

**Question 4 (b)**

How can you ensure, as a programmer, that your test cases allow for robust testing of your code? Provide examples with your explanation.

For a program to be robust, it must be able to attempt to function correctly or as intended under “stressful” situations or with invalid inputs. As such, the program should have a good amount of error handling, so if a certain part of the input is invalid, it will be able to handle that, work as intended if possible, and if not guide the user to fix the invalid input. To be able to test this to allow for more robust code, you must undertake the process of fault injection, which is a testing method where invalid inputs are injected into a system to observe resilience. If the program does not work as intended, it should be fixed and addressed in the code, as to not allow critical faults to make a whole system fail.

**PROGRAMMING FEATURES**

**Question 5**

Why do we have different data types and data structures for programming? How do these different types and structure assist in writing efficient code? Provide examples in your answer.

Data types are used to model different ideas, concepts, and real-world scenarios in code. Think about why we have multiple words in the English language. If we only had one word, we would be much more limited in our capability to express ideas and work efficiently with others. In the same way, if we didn’t have many different data types, we would not be able to efficiently communicate with a computer and work with it to get it to do what we would like. Data structures are also very important, as they allow an organised storage of data, making programs faster and more efficient. It is the same in real life; in life we have different types of data: a number is not the same as someone’s name. A date is not the same as a house. As such, to model the real world, we need different data types.

These different structures and types assist in writing efficient code as they better allow the code to model the real world. Humans are good at thinking like humans, and if we can use a wide array of data types and structures that function the way humans would, the code becomes not only more readable for others and efficiently understandable, it becomes more efficient for the computer, as it allows us to come up with and think of algorithms that we otherwise would not have. As such, the use of the correct data type and structure is a way of organisation code for human interpretation, and it allows us to better come up with ways to manipulate and improve our code.

For example, classes are abstract data types, unlike numbers of strings. You can’t have a physical representation for them, so why are they useful? Well, they allow us to model more complex ideas, such as houses or streets. They encompass other smaller data types, as each street has a house, which has people, which have their own personality and traits. If these were all numbers on a scale of 1 – 10, the code would very quickly become unreadable with a whole lot of variables for different people and streets and nesting, and the organisation becomes so terrible that no one can read it, and the flexibility of the code becomes very hard to manage, to improve it in the future. With more data types, especially abstract ones, we can have specific human-like models for different scenarios which allow us to better understand the structure of code so we can better represent what we would like.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| ***Unit 3***  ***Outcome 1***  ***Interpret teacher-provided solution requirements and designs, and apply a range of functions and techniques using a programming language to develop and test working software modules.*** | **DESCRIPTOR: typical performance in each range** | | | | |
| **Very low** | **Low** | **Medium** | **High** | **Very high** |
| Limited interpretation of solution requirements and designs to develop working modules. | Some interpretation of solution requirements and designs to develop working modules. | Sound interpretation of solution requirements and designs to develop working modules. | Most solution requirements and designs are interpreted accurately to developing working modules. | All solution requirements and designs are interpreted accurately to developing working modules. |
| Limited selection and use of data types and data structures. | Some selection and use of appropriate data types and data structures. | Sound selection and use of data types and data structures to develop working modules. | Detailed selection of relevant data types and data structures to develop working modules. | Comprehensive selection of relevant data types and data structures to develop working modules. |
| Limited selection and use of processing features of the programming language to develop some working modules. | Some selection and use of appropriate processing features of the programming language to develop some working modules. | Sound selection and use of appropriate processing features of the programming language to develop some working modules. | Most processing features of the programming language have been selected and used to develop all working modules. | Comprehensive selection and use of relevant processing features of the programming language to develop all working modules. |
| Limited explanation of how the selected processing features are used to develop working modules. | Some justification and explanation of how the selected processing features are used to develop working modules. | Sound justification and explanation of how the selection of appropriate processing features are used to develop working modules. | Detailed justification and explanation of how the selection of appropriate processing features of the programming language are used to develop working modules. | Comprehensive justification and explanation of how the selection of appropriate processing features of the programming language are used to develop working modules. |
| Limited data validation techniques are applied to check the reasonableness of some input data. | Some data validation techniques are effectively applied to check the reasonableness of some input data. | Sound use of data validation techniques are effectively applied to check the reasonableness of input data. | Detailed use of relevant data validation techniques are applied to efficiently and effectively check the reasonableness of all input data. | Comprehensive use of relevant data validation techniques are applied efficiently and effectively to check the reasonableness of all input data. |
| Limited range of test data is expressed in a testing table, with incomplete or missing results. | Some testing of test data is expressed in a testing table with actual output stated. | Sound range of testing of test data is expressed in a testing table, with both expected and actual output stated and some evidence of debugging. | Detailed use of test data is expressed in a testing table, with both expected and actual output stated with evidence of debugging. | Comprehensive use of test data is expressed in a testing table, with both expected and actual output stated, and showing detailed evidence of debugging. |
| Limited internal documentation with few comments regarding the use of the selected processing features. | Some internal documentation with comments regarding the functioning of modules and the use selected processing features. | Sound use of internal documentation with comments regarding the functioning of modules and the use of selected processing features. | Most software modules include detailed internal documentation regarding the functioning of modules and use of selected processing features | All software modules include comprehensive internal documentation regarding the functioning of modules and use of selected processing features. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Low 1–20 | Low 21–40 | Medium 41–60 | High 61–80 | Very High 81–100 |