

**ANL252**

**Python for Data Analytics**

**Tutor-Marked Assignment**

**July 2023 Presentation**

Chua Qi Ying (E2081908)

13 September 2023

**Question 1(a)**

**Reasons For Plagiarism in Coding:**

* **Availability And Accessibility of Code**
  + Individuals are tempted to copy and reuse existing code without attributions due to the ease of access to open-source software, which is free and available for use to anyone, regardless of credit (Lee, 2020).
* **Lack Of Understanding**
  + Coding plagiarism occurs when individuals copy the code as they lack the skills or do not understand the concepts. Thus, individuals struggling with assignments are tempted to use open-source software and resort to plagiarism as a quick solution to complete their assignments without putting much effort (Lee, 2020).

**Ways To Avoid:**

* **Brainstorm Own Code**
  + Instead of relying on existing code, individuals should deepen their understanding of programming concepts and techniques to come up with original and unique code from scratch (Lee, 2020).
* **Cite The Source**
  + It is important to cite the sources when utilising someone’s code by adding a comment in the code that gives credit to the original author and providing the link to the original source (LinkedIn, n.d.).
* **Check For Plagiarism**
  + Run the code a through plagiarism checker to identify any similarities with existing codes and make necessary changes to avoid unintentional plagiarism (LinkedIn, n.d.).

**Word Count: 199**

**Question 1(b)**

**Python Code:**

***# Honours Degree Classification***

classofhonour\_thresholds = [(4.50, 'First Class'), (4.00, 'Second Upper'), (3.50, 'Second Lower'), (3.00, 'Third Class'), (2.00, 'Pass'), (0.00, 'Fail')]

***# Input CGPA***

CGPA = input("Enter your CGPA:")

***# Convert CGPA to a float***

CGPA = float(CGPA)

classofhonour = None

if CGPA < 0 or CGPA > 5.00:

print("Invalid CGPA")

else:

for threshold, classofhonour\_value in classofhonour\_thresholds:

if CGPA >= threshold:

classofhonour = classofhonour\_value

break

if classofhonour is None:

print("Invalid CGPA")

else:

print(f"The Class of Honour for a CGPA of {CGPA} is {classofhonour}.")

This piece of Python code determines the class of honour that student will receive based on a given CGPA using a list of tuples, if-else block, and for-loop.

The code defines a list called ‘classofhonour\_thresholds’. Each element in this list is a tuple that indicates a CGPA threshold and the corresponding class of honour. Using the ‘input()’ function, the program prompts the user to enter their CGPA and converts the ‘CGPA’ string to a float using the ‘float()’ function. The variable ‘ClassofHonour’ is initially set to ‘None’, to store the class of honour based on the CGPA. If the CGPA is less than 0 or more than 5, it prints the message “Invalid CGPA” and does not proceed. If the CGPA falls within the valid range, the code uses a ‘for’ loop to iterate over the ‘classofhonour\_thresholds’ list. Inside the loop, if the CGPA is greater than or equal to a threshold, it assigns the corresponding class of honour to the `classofhonour` variable and prints the message displaying the CGPA and the class of honour. If the ‘classofhonour’ variable remains ‘None’ after the loop, the CGPA did not meet any of the threshold conditions and prints the “Invalid CGPA” message.

**Word Count: 200**

**Question 1(c)**

**Python Code:**

***# Honours Degree Classification***

class\_of\_honour\_thresholds = [

(4.50, 'First Class'),

(4.00, 'Second Upper'),

(3.50, 'Second Lower'),

(3.00, 'Third Class'),

(2.00, 'Pass'),

(0.00, 'Fail')

]

***# Input CGPA***

cgpa = float(input("Enter your CGPA: "))

***# Variable to store the determined Class of Honour***

class\_of\_honour = None

***# Check if CGPA is within the valid range***

if cgpa < 0 or cgpa > 5.00:

print("Invalid CGPA")

***# Iterate through thresholds to determine the class of honour***

else:

for threshold, class\_of\_honour\_value in class\_of\_honour\_thresholds:

if cgpa >= threshold:

class\_of\_honour = class\_of\_honour\_value

break

***# If no class\_of\_honour is assigned, CGPA is invalid***

if class\_of\_honour is None:

print("Invalid CGPA")

else:

print(f"The Class of Honour for a CGPA of {cgpa} is {class\_of\_honour}.")

**Output:**

Enter your CGPA:3.90

The Class of Honour for a CGPA of 3.9 is Second Lower.

**Rationale For Rewriting the Code:**

* **Improved Readability**
  + In the Python coding convention, variable names and functions should be lowercase (ZoeZbar, 2020). Hence, the rewritten code follows the coding convention and CGPA is changed to cgpa. This will improve code readability and make it easier to understand.
* **Consistent Naming Convention**
  + The suggested format for variable names is to write in lowercase with words separated by underscores (DEMİR, 2020). The rewritten code in the above example adheres to a consistent variable name convention, such as snake\_case, and classofhonour was changed to class\_of\_honour (DEMİR, 2020). By using a consistent naming convention, the code is more readable and consistent for others to understand.
* **Input Handling**
  + In the original code, it is assumed that when the user inputs a value as the CGPA using input(), the value is stored as a string. If the user inputs a non-numeric value, a ValueError will be generated. The rewritten code improves input handling by using the input() function to read input and immediately convert it to a float using the float() function (IncludeHelp, 2019). In the example above, when the user enters “4.20” as the CGPA, it is directly converted to a float. The rewritten code ensures that the input is accurately converted to a float and avoids errors by applying the float(input)) immediately upon taking user input (IncludeHelp, 2019).
* **Code Understanding**
  + Comments are provided in the rewritten code to provide context and explain each section of the code, making it easier to understand, debug and modify. It allows developers to enhance code understanding, facilitate collaboration and serve as a reference for future use, allowing users to comprehend the intention behind these codes (Manoor, 2023). Thus, rewriting the code to include comments offers valuable insights and ensures the sustainability of the code.

**Word Count: 300**

**Question 2**

To improve the code in Appendix 1, the following must be implemented:

**Meaningful Variable Names**

Variable names should describe what the variable represents to enhance code readability and maintainability (W3docs, n.d.). For example, Appendix 1 is a shopping list program where users can select and add items to their shopping list. Variable names can include “shopping\_list” or “continue\_shopping” instead of generic names such as “query” and “updated\_items”.

**Add Comments**

Comments can be included to provide context and explain the purpose of the code. This makes it easier for developers to understand and modify the functionality of code segments, increasing readability and maintainability of the code (Manoor, 2023).

**Dictionaries**

Dictionaries are used to correlate each product with its price. As products and their prices are stored together, users do not need to input the price, making it easy to get the accurate and reliable pricing information for each product and reduce inconsistencies.

**Input Validation**

Input validation ensures that user input is valid to prevent code execution. If the input is invalid, an error message will be displayed and the program will prompt the user for input until they enter a valid response for the ‘continue\_shopping’ variable, either ‘yes’ or ‘no’.

**Word Count: 199**

**Improvised Python Code**

***# Dictionary with the items and their price***

products = {'laptop': 1000, 'mouse': 30, 'webcam': 50, 'keyboard': 45, 'speaker': 95}

continue\_shopping = 'yes'

shopping\_list = []

print(f"We have a list of products here: {products}.")

while continue\_shopping == 'yes':

***# Prompt user to enter the item they want to buy***

item = str(input("Hello! What do you want to buy? "))

***# Check if the item is in the product list***

if item not in products:

print(f'Wrong product! Please try again.')

continue

***# Create a dictionary with the items and its price***

entered\_input = {'item': item, 'price': products[item]}

shopping\_list.append(entered\_input)

***# Prompt user to continue shopping or exit***

while True:

continue\_shopping = str(input("Would you like to continue? (yes/no) "))

if continue\_shopping in ['yes','no']:

break

print("Invalid input! Please enter 'yes' or 'no'.")

***# Display the final shopping list***

print(f"This is our updated shopping list: {shopping\_list}")

**References**

*Best Practices for Python Variable Naming*. (n.d.). W3docs. <https://www.w3docs.com/learn-python/variable-names.html>

DEMİR, E. (2020, January 16). *NAMING CONVENTION.* Medium. <https://medium.com/@e.demir/naming-convention-9b9663d5fb8a>

Lee, C. (2020, July 28). *What is Programming Plagiarism? Why Is It on the Rise?* turnitin. <https://www.turnitin.com/blog/what-is-programming-plagiarism-why-is-it-on-the-rise>

Manoor, S. (2023, June 3). *The Importance of Comments in Python Programming*. LinkedIn. <https://www.linkedin.com/pulse/importance-comments-python-programming-swarooprani-manoor/>

*Read input as a float in Python.* (2019, April 2). IncludeHelp. <https://www.includehelp.com/python/read-input-as-a-float.aspx>

*What's your process for avoiding plagiarism when using code from other sources?* (n.d.). LinkedIn. <https://www.linkedin.com/advice/1/whats-your-process-avoiding-plagiarism-when-using>

ZoeZbar. (2020, December 22). *Best Naming Conventions When Writing Python Code.* Data Science Central. <https://www.datasciencecentral.com/best-naming-conventions-when-writing-python-code/>