Course Code: ANL252

Title of TMA: TMA01

SUSS PI No.: Z1910329

Name: Muhammad Fithri Bin Fadiliah

Submission Date: 13 September 2023

**1a)**

Coding plagiarism causes:

* Lacking citation standards due to programming being a new creativity form; some ambiguity exists between coding and plagiarism (Bailey, 2017).
* Convenient online access to coding repositories and forums – higher likelihoods of plagiarism when students misuse them as shortcuts for assessments (Ngo, 2016).
  + Desire for better grades without much time or effort (Bowyer & Hall, 2001).
* Procrastination when completing assignments leading to rushed timeline and use of shortcuts (Steel, 2007).
  + Misleading perceptions that tasks can be completed within a small timeframe.

Avoiding coding plagiarism:

* Students provided plagiarism education to ensure understanding of what constitutes a plagiarism act (Abraham & Milligan, 2008).
  + Designing lessons to advance learning outcomes (e.g. through worked examples) and peer learning agendas; maximise understanding of program coding (Abdul-Rahman & Du Boulay, 2014; Ngo, 2016).
* Employing writing cleanroom where uncited texts is avoided by including citation/attribution into the code writing process; prevents accidental plagiarism (Bailey, 2016).
  + Texts from outside sources (lifted or not) are immediately cited when inserted.
* Employing comment habits throughout coding process – higher awareness of what codes are (or are not) original (Bailey, 2017).
  + Demonstration of comprehension of written code from external sources.

**1b)**

# This code was obtained from Wu & Zhu (2023), study guide for ANL252, Python for data analytics, p. SU1-38.

name1 = input ("What is your name?")

score1 = int (input("What is your score?"))

if score1<40:

grade1 = "F"

elif score1>=40 and score1<50:

grade1 = "D"

elif score1>=50 and score1<60:

grade1 = "C"

elif score1>=60 and score1<80:

grade1 = "B"

elif score1>=79 and score1<100:

grade1 = "A"

print(f"{name1}, your grade is {grade1}.")

name2 = input ("What is your name?")

score2 = int (input(f"Hello {name1}. What is your score?"))

if score2<40:

grade2 = "F"

elif score2>=40 and score2<50:

grade2 = "D"

elif score2>=50 and score2<60:

grade2 = "C"

elif score2>=60 and score2<80:

grade2 = "B"

elif score2>=79 and score2<100:

grade2 = "A"

print(f"{name2}, your grade is {grade2}.")

Output:

What is your name?Brian

What is your score?65

Brian, your grade is B.

What is your name?Samantha

Hello Brian. What is your score?89

Samantha, your grade is A.

According to Wu and Zhu (2023), above code-piece gathers data, particularly the score, to produce a corresponding grade. Value for name variable (e.g., name1 & name2) is first gathered before scores (e.g., score1 & score2), whereupon the scores are then graded according to the different if-conditions. In this case, the grades are A (between 80 & 100), B (between 59 & 80), C (between 49 & 60), D (between 39 & 50), and F (below 40). The variables, however, are not linked to each other (e.g., name1 not linked to score1) and may be used independently of each other.

If-conditions written (e.g., lines 3 & 18) signify conditions that if the score of user is below 40, grade given would be F. The elif-conditions (e.g., lines 5-12 & lines 20-27) written thereafter in the two code-blocks signify how if scores do not fulfil preceding conditions, Python would move to the next elif-conditions until a condition is satisfied. If score is beyond 100, Python returns error as no values are given to grade variables (e.g., grade1 & grade2).

Print functions at the end of each code-block give output of user’s name and grade received from score provided and if/elif-conditions.

**1c)**

# This code-piece was based on <https://stackoverflow.com/questions/36716503/how-to-create-a-grading-system-in-python> – accessed on 12/09/2023.

# User-defined function “fetch\_grades” to retrieve grade according to test scores given by user input...

# called when grade retrieval is required.

def fetch\_grades(test\_score):

if test\_score >= 0 and test\_score <= 39:

return "F"

elif test\_score >= 40 and test\_score <= 49:

return "D"

elif test\_score >= 50 and test\_score <= 59:

return "C"

elif test\_score >= 60 and test\_score <= 79:

return "B"

elif test\_score >= 80 and test\_score <= 100:

return "A"

number\_of\_students = 2

# For-loop below asks students their name and score before grade is given and printed out.

# Change number\_of\_students to adjust number of respondents.

for i in range(number\_of\_students):

student\_name = str(input("Please enter your name: "))

test\_score = int(input("Please enter your test score (0-100): "))

# Variable for first student’s grade calls on the “fetch\_grades” user-defined function and takes resultant value.

student\_grade = fetch\_grades(test\_score)

print(f"Greetings, {student\_name}. With a score of {test\_score}, your grade is {student\_grade}.")

Output:

Please enter your name: Brian

Please enter your test score (0-100): 65

Greetings, Brian. With a score of 65, your grade is B.

Please enter your name: Samantha

Please enter your test score (0-100): 89

Greetings, Samantha. With a score of 89, your grade is A.

Rewriting code rationale:

* Better readability (Jayasinghe, 2018):
  + Most of the variables are rewritten to better indicate what the code is asking for from the user (e.g., name1 to student\_name1) , allowing better readability of the code.
  + Inserting comments to explain functions of the code so that the code could be understood more easily, building awareness as to the code’s functionality.
* Original code is cumbersome (Kadosh, 2019):
  + Original code is needlessly long in that the code for the second user is a repeat of the code for the first user.
  + Rewriting involved plucking out the needlessly long code block, replacing it with a user-defined function “fetch\_grades”, then applying it when user inputs their score after their name.
  + Rewritten code is much shorter and less cumbersome to read and process.
* Applying new technique; requires code change (Sharovar, 2022):
  + As extension to efficiency argument, more advanced coding techniques to handle repeated inputs helps build efficiency.
  + Use of user-defined function “fetch\_grades” helps shorten entire code-piece due to its modular function – allowing its use whenever called without re-typing code-block for each user.
  + Addition of a for-loop to allow better maintainability in cases where number of students may change and more (or less) inputs are required – also eliminating code redundancies.
* Better error-handling (Kadosh, 2019):
  + Errors can be found in the original code especially with regards to the if- and elif-condition statements.
    - For example, “if score1<40:” in the original code allows for any negative integer to be input, reducing robustness of the code.
  + Furthermore in the original code, changes to one code block for one user may not be reflected in the code of the other – possibly leading to higher inconsistencies and errors.

**2)**

# This code piece was based on ANL252 Tutor-marked assignment, July 2023 presentation.

available\_products = ['Laptop', 'Mouse', 'Webcam', 'Keyboard', 'Speaker']

purchase\_query = 'Yes'

updated\_items\_list = [] # Empty list created for appending of selected item and price below.

# While-loop to keep user within shopping instance should they desire to buy more than one item.

while purchase\_query == 'Yes':

print(f'Hello! Our list of available products are as follows: {available\_products}.')

item\_to\_purchase = str.title(input("What would you like to purchase? "))

# If-condition below; if item is not avaiable, user is given a choice to continue shopping for other items.

# While condition below in case user enters invalid entry; provided error and prompted to re-enter.

if item\_to\_purchase not in available\_products:

print(f'Product is unavailable! Please try again.')

purchase\_query = str.title(input("Would you like to continue shopping? (Yes/No) "))

while purchase\_query != "Yes" and purchase\_query != "No":

purchase\_query = str.title(input("Invalid entry. Would you like to continue shopping? (Yes/No) "))

# Elif-condition for items that are available, providing user choice to input price of product.

elif item\_to\_purchase in available\_products:

item\_price = input("How much is it (in SGD)? ")

# while-loop and try-except block to ensure that amount entered is integer/float

while type(item\_price) != float and type(item\_price) != int:

try:

item\_price = float(item\_price) # item\_price given would be converted to a float

except:

item\_price = input("Invalid input. Please re-enter amount in SGD ") # item\_price not float or integer, user to re-enter amount

# Values for item\_to\_purchase and item\_price variables added to list named chosen\_item\_and\_price.

chosen\_item\_and\_price = [item\_to\_purchase, item\_price]

# updated\_items\_list.append(chosen\_item\_and\_price) inserts item & price into updated\_items\_list

updated\_items\_list.append(chosen\_item\_and\_price)

print(f'This is your updated shopping list: {updated\_items\_list}')

purchase\_query = str.title(input("Would you like to continue shopping? (Yes/No) "))

# While condition below in case user enters invalid entry; provided error and prompted to re-enter.

while purchase\_query != "Yes" and purchase\_query != "No":

purchase\_query = str.title(input("Invalid entry. Would you like to continue shopping? (Yes/No) "))

print(f'This is your final shopping list: {updated\_items\_list}')

To improve reliability, the code was thoroughly reviewed tested for every possible path to ensure that possible errors are properly handled (Jayasinghe, 2018). This includes ensuring integer/float entry of item price (e.g., lines 18-25), case-sensitivity of product names (e.g., addition of .title() at lines 14 & 16), and yes/no answers (e.g., lines 15-16 & 32-35).

To improve readability, Jayasinghe (2018) suggested thoroughly supplementing the code with relevant comments to ensure that those who read the code can easily understand what the code is trying to do and what results it would give. This includes a more specific naming of variables like “purchase\_query” and “chosen\_item\_and\_price”.

To improve maintainability, the codes are kept simple and easily understood to allow more easy additions of other functions or fixes of errors. This includes the more specific naming conventions, proper indentations, more spaced out formatting style, and more thorough and precise comments additions as suggested by Jayasinghe (2018).

Overall, the revised code enhances the functional quality the original code seeks to achieve by providing better error handling, more convenient entry for the user in their purchase listing experience, more information as to what the codes do via thorough commenting process, and better maintainability for future handling of errors or refinements.

**References**

Abdul-Rahman, S. S., & Du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, *30*, 286–298. <https://doi.org/10.1016/j.chb.2013.09.007>

Abraham, S., & Milligan, G. (2008). Software plagiarism in undergraduate programming classes. *Information Systems Education Conference*, *11*(6), 2008–2011.

Bailey, J. (2016, August 10). *How to use a writing cleanroom to prevent plagiarism*. PlagiarismToday. <https://www.plagiarismtoday.com/2016/08/10/how-to-use-an-writing-cleanroom-to-prevent-plagiarism>

Bailey, J. (2017, November 17). *Plagiarism and programming: How to code without plagiarizing*. Turnitin. <https://www.turnitin.com/blog/plagiarism-and-programming-how-to-code-without-plagiarizing-2>

Bowyer, K. W., & Hall, L. O. (2001). Reducing effects of plagiarism in programming classes. *Journal of Information Systems Education*, *12*(3), 141–148. <https://doi.org/10.7763/IJIET.2016.V6.808>

Jayasinghe, S. (2018, July 16). *The importance of code quality*. Medium. <https://medium.com/emblatech/the-importance-of-code-quality-ac7afa598c0d>

Kadosh, Y. (2019, September 1). *Why rewriting is essential for keeping software alive*. Medium. <https://betterprogramming.pub/why-code-rewriting-is-essential-for-keeping-software-alive-81c7307e7f6>

Sharovar, E. (2022, December 17). *Code rewriting: When and why*. Waverley Software. <https://waverleysoftware.com/blog/code-rewriting-when-and-why>

Steel, P. (2007). The nature of procrastination: A meta-analytic and theoretical review of quintessential self-regulatory failure. *Psychological Bulletin*, *133*(1), 65–94. <https://doi.org/10.1037/0033-2909.133.1.65>

Wu, K. Y., & Zhu, S. (2023). *ANL252 Python for data analytics*. Singapore University of Social Sciences.