**Course code:** ANL252

**Title of the TMA:** ANL252 Python for Data Analytics Tutor-Marked Assignment

**SUSS PI No.:** Q2310757

**Name:** Pang Shi Da Lawrence

**Submission Date:** 8 September 2023

**Q1 a) Plagiarism in coding happens due to various reasons, including:**

* Lack of Knowledge: A weak understanding of foundational concepts, such as arrays, may prompt individuals to copy code.
* Time Constraints: Meeting strict deadlines sometimes pushes individuals towards unoriginal solutions
* Lack of Originality: A classic instance is sorting algorithms; many simply replicate standard methods found online rather than creating their own (Bazaluk et al., 2023).
* Perceived Easiness: The vast availability of online resources may tempt individuals to use code snippets without giving due credit (Puri & Mulay, 2017).
* Unawareness: Some might be oblivious to the boundaries of plagiarism in coding (Joy et al., 2013).

**To avoid plagiarism in coding, you can follow these best practices:**

* Understand the Basics: Grasping core concepts, like basic algorithm design, can prevent reliance on copied code
* Proper Citations: Always credit the source within code comments when adopting code.
* Paraphrase and Understand: Thoroughly review borrowed code, rewrite it in your unique style, and annotate with comments detailing each logical step
* Peer Reviews: Encourage regular code assessments to catch and rectify potential overlaps.
* Educate: Emphasize and promote academic and professional integrity throughout coding education. (Puri & Mulay, 2017).
* Use Plagiarism Checkers: Tools such as MOSS can be invaluable in detecting similarities (Szymutko & Seda, 2023)

[Reference]

Bazaluk, B., Silva, F. S., Holanda, M., & Da Silva, D. (2023). Source Code Plagiarism in Computer Science Courses: Facts and Impressions. Anais do XXXI Workshop sobre Educação em Computação (WEI 2023). https://doi.org/10.5753/wei.2023.230066

Joy, M. S., Sinclair, J. E., Boyatt, R., Yau, J. Y-K., & Cosma, G. (2013). Student perspectives on source-code plagiarism. International Journal for Educational Integrity, 9(1), 3–19.

Puri, K., & Mulay, P. (2017). Knowledge Management in Academic Community: Code and Content-Based Plagiarism Prevention MARG. In Enhancing Academic Research With Knowledge Management Principles (pp.1-56). DOI: 10.4018/978-1-5225-2489-2.CH006

Szymutko, M., & Seda, P. (2023). Plagiarism Detection in Software Projects Using Abstract Syntax Trees. In Proceedings II of the 29st Conference STUDENT EEICT 2023: Selected papers (pp. 45-48). <https://doi.org/10.13164/eeict.2023.45>

Question 1B)

def fibonacci\_sequence(n, reverse\_order=False):

fib\_sequence = [0, 1]

while len(fib\_sequence) < n:

fib\_sequence.append(fib\_sequence[-1] + fib\_sequence[-2])

return fib\_sequence[::-1] if reverse\_order else fib\_sequence

if \_\_name\_\_ == "\_\_main\_\_":

num = int(input("Enter the number of Fibonacci numbers you want: "))

order\_preference = input("Do you want the sequence in reverse? (yes/no): ").lower()

if num < 2:

print("Please enter a number greater than 1.")

else:

reverse\_flag = True if order\_preference == 'yes' else False

sequence = fibonacci\_sequence(num, reverse\_flag)

order\_text = "reversed " if reverse\_flag else ""

print(f"The first {num} {order\_text}Fibonacci numbers are: {sequence}")

**Explanation**: The Python script defines a function, fibonacci\_sequence, to compute the Fibonacci sequence up to a given term, n. The Fibonacci sequence starts with 0 and 1, with each subsequent term being the sum of the two preceding ones. The function employs a loop for sequence generation and has an additional reverse\_order parameter. When true, the sequence is returned in reverse.

(Janíčko, 2018)

The script uses the conditional if \_\_name\_\_ == "\_\_main\_\_": to check if it is being run directly. When executed, it prompts the user for the sequence's length and order preference: regular or reversed. The input is validated to ensure the sequence length is greater than one. Based on the user's choices, the function is called, and the resultant sequence is displayed.

This enhanced Fibonacci generator exemplifies the importance of user-centric design in programming (Chammas, Quaresma, & Mont’Alvão, 2015)

[References]

Janíčko, O. (2018, December 10). New fundamental discovery of the reverse Fibonacci sequence.

Chammas, A., Quaresma, M., & Mont’Alvão, C. (2015). A Closer Look on the User Centred Design. Procedia Manufacturing, 3, 5397–5404. https://doi.org/10.1016/j.promfg.2015.07.656

Question 1C)

def generate\_fib(n, reversed\_seq=False):

fib\_list = [0, 1]

for \_ in range(2, n):

fib\_list.append(fib\_list[-1] + fib\_list[-2])

return fib\_list if not reversed\_seq else fib\_list[::-1]

if \_\_name\_\_ == "\_\_main\_\_":

count = int(input("How many Fibonacci terms do you need? "))

order\_choice = input("Would you like the terms in descending order? (yes/no): ").strip().lower()

if count < 2:

print("Choose a number greater than 1.")

else:

is\_reversed = order\_choice == 'yes'

fib\_terms = generate\_fib(count, is\_reversed)

direction = "descending" if is\_reversed else "ascending"

print(f"The {count} Fibonacci terms in {direction} order are: {fib\_terms}")

[Outputs]

How many Fibonacci terms do you need? 5

Would you like the terms in descending order? (yes/no): yes

The 5 Fibonacci terms in descending order are: [3, 2, 1, 1, 0]

Rationale for Code Rewrite:

**Modular Approach:** Renamed the function to generate\_fib for better clarity and adopted a for-loop instead of a while-loop to make the iteration process more intuitive (Wirsing & Broy, n.d.)

**Improved User Prompts:** Changed the user prompts to make them more straightforward and avoid potential confusion.

**Simplified Logic:** Simplified the logic for determining sequence order by directly evaluating the user's input as a condition for the is\_reversed variable. Additionally, renamed variables for clarity and better understanding (Tarau, 2021)

[Bibliography]

Wirsing, M., & Broy, M. (n.d.). A modular framework for specification and implementation. Fakultät für Mathematik und Informatik, Universität Passau. Retrieved from https://link.springer.com/content/pdf/10.1007/3-540-50939-9\_124.pdf.

Tarau, P. (2021). Natlog: a Lightweight Logic Programming Language with a Neuro-symbolic Touch. In A. Formisano, Y.A Liu, et al. (Eds.), International Conference on Logic Programming (Technical Communications) 2021 (ICLP 2021) (pp. 141-154). EPTCS 345. <https://doi.org/10.4204/EPTCS.345.27>

Q2)

products = ['laptop', 'mouse', 'webcam', 'keyboard', 'speaker']

def display\_products():

print(f"We have the following products available: {', '.join(products)}.")

def add\_to\_cart():

cart = []

while True:

item = input("What item would you like to purchase? ").lower()

if item not in products:

print("Sorry, that product is not available. Please choose a valid product.")

continue

price = float(input(f"How much is the {item} (in SGD)? "))

cart.append((item, price))

query = input("Would you like to add another item? (yes/no) ").lower()

if query != 'yes':

break

return cart

if \_\_name\_\_ == "\_\_main\_\_":

display\_products()

updated\_items = add\_to\_cart()

print(f"Updated shopping list: {updated\_items}")

**Function Decomposition:** The code has been divided into functions (display\_products and add\_to\_cart) for better readability and modularity. It improves maintainability by allowing changes to specific functionalities without affecting the entire script.

**Improved User Interaction:** The user is provided with a list of available products and is continually prompted until a valid product is chosen. This enhancement aids in preventing user errors.

**Variable Naming:** Renamed variables like entered\_input to more descriptive names, ensuring clarity. Consistent naming conventions and clear variable names improve the readability of the code, making it easier for developers to understand and maintain.