Course Code: ANL252

Title: TMA01

PI No.: H2310678

Name: Muhammad Bin Osman

Date of submission: 15/09/23

1a) Plagiarism in coding entails crediting one’s self for work that is done by others. An unethical practice, it is unfortunate that plagiarism is widespread in many settings such as academics. We will be discussing the reasons behind plagiarism and ways to avoid it below.

Why plagiarise?

* Anxiety in meeting deadlines could result in plagiarism. This is exacerbated when assignments are difficult. Students that are relatively weaker would struggle in handling tougher assignments, resorting to plagiarism to not face penalties for late submission (Bidgood & Merrill, 2017).
* Extremely time-efficient. Plagiarising saves time and energy in completing assignments as one would only need to copy-and-paste from multiple sources to finish their work (Chen, 2018).

How to avoid?

* Improving one’s craft would remove the need for plagiarising. Being exposed to more practices builds confidence in one’s skill, making them less reliant on unethical practices.
* Education on coding ethics would instill proper moral values, reducing plagiarism. By understanding the rationale of ethics, students learn to respect the work of others and give credit where it’s due.
* Learning how to cite resources properly protects one from being accused of plagiarism, while also properly crediting the owner of said work. This shows respect to intellectual property.

(199 words)

b) Code:

#area calculator

def areacalculator():

\_input\_ = input("Enter the shape you want to calculate area of: ")

area = 0

pie = 3.14

if \_input\_ == "Square":

side = int(input("Enter the value of side: "))

area = area + (side \*\* 2)

elif \_input\_ == "Circle":

radius = int(input("Enter the value of radius: "))

area = area + (2 \* pie \* radius)

elif \_input\_ == "Rectangle":

length = int(input("Enter the value of length: "))

width = int(input("Enter the value of length: "))

area = area + (length \* width)

elif \_input\_ == "Triangle":

base = int(input("Enter the value of base: "))

height = int(input("Enter the value of height: "))

area = area +(0.5 \* base \* height)

else:

print ("Select a valid shape")

print ("%.2f" % area)

areacalculator()

Output:

Enter the shape you want to calculate area of: Square

Enter the value of side: 2

4.00

Enter the shape you want to calculate area of: Circle

Enter the value of radius: 3

18.84

Enter the shape you want to calculate area of: Rectangle

Enter the value of length: 4

Enter the value of length: 2

8.00

Enter the shape you want to calculate area of: Triangle

Enter the value of base: 6

Enter the value of height: 3

9.00

(codecademy, 2018)

(22 lines)

This code is useful if the user intends to calculate the areas for various shapes. By assigning the function areacalculator(), the user can call upon the calculator anytime by inputting the function. Within the function, it prompts the user to enter their desired shape. The values of area and pi are already pre-assigned to be used in calculations later. Running on if-elif-else conditions, the code would respond differently to the shape that was inputted. The user is then prompted to input either side,radius,length/width or base/height. The script then computes the area via the assigned arithmetic statements and is stored in the variable “area”. The area is then printed, rounded off to 2 decimal places. If the user inputs an invalid input, the “else” condition is triggered and a prompt is shown to inform the user of their error. (142 words)

c) Rewritten code:

#Header

print("~Area Calculator~")

#Defining function with tuple to ensure user inputs specific shapes

def calc\_area():

valid\_shapes=("Square","Rectangle","Triangle","Circle")

while True:

input\_shape=str(input(f"Please enter desired shape {valid\_shapes}: "))

if input\_shape in valid\_shapes:

break

else:

print("You have entered an invalid shape. Try again.")

#incorporating if-else-elif with loops to prevent termination of code during wrong input

if input\_shape == "Square":

true\_input = False

while true\_input == False:

try:

length=float(input("What is the length? "))

except ValueError:

print("This is an invalid input. Try again. ")

else:

true\_input = True

area=length\*\*2

print(f"Area: {area}")

elif input\_shape == "Rectangle":

true\_input = False

while true\_input == False:

try:

length=float(input("What is the length? "))

breadth=float(input("What is the breadth? "))

except ValueError:

print("This is an invalid input. Try again.")

else:

true\_input = True

area=length\*breadth

print(f"Area: {area}")

elif input\_shape == "Triangle":

true\_input = False

while true\_input == False:

try:

base=float(input("What is the base? "))

height=float(input("What is the height? "))

except ValueError:

print("This is an invalid input. Try again. ")

else:

true\_input = True

area = 0.5 \* base \* height

print(f"Area: {area}")

elif input\_shape == "Circle":

true\_input = False

while true\_input == False:

try:

radius=float(input("What is the radius? "))

except ValueError:

print("This is an invalid input. Try again. ")

else:

import numpy as np

true\_input = True

area = np.pi \* radius\*\*2

print(f"Area: {area}")

#recall function

calc\_area()

Output:

~Area Calculator~

Please enter desired shape ('Square', 'Rectangle', 'Triangle', 'Circle'): Square

What is the length? 4

Area: 16.0

~Area Calculator~

Please enter desired shape ('Square', 'Rectangle', 'Triangle', 'Circle'): Rectangle

What is the length? 5

What is the breadth? 2

Area: 10.0

~Area Calculator~

Please enter desired shape ('Square', 'Rectangle', 'Triangle', 'Circle'): Triangle

What is the base? 4

What is the height? 6

Area: 12.0

~Area Calculator~

Please enter desired shape ('Square', 'Rectangle', 'Triangle', 'Circle'): Circle

What is the radius? 5

Area: 78.53981633974483

Why rewrite code?:

* We rewrite the code in order to make it more user-friendly. The revised version provides a tuple and displays it to the user to prevent any confusion by showing what shapes are acceptable in the function. It also implements while and try-loops to prevent the script from terminating with a wrong input, giving the user a chance to re-type their input without having to call the function again.
* We also made it more organised. Splitting the script into multiple parts with relevant headers/labels makes it easy to edit or debug when necessary. Having sufficient labels also makes the script more readable and easier to maintain.
* Another reason is to improve the accuracy of answers. Using numpy instead of an approximation of 3.14 makes answers more accurate and more in line with the function of a calculator. Removing the string formatting at the end of the original script gives the user more freedom in choosing their answers as they would not be limited to only answers with 2 decimal places. While it can be more efficient depending on the needs of the user, leaving the answer in full would be better for providing flexibility.
* Lastly, it can also provide opportunity to test and evaluate one’s coding ability. By rewriting the script, it exposes the user to more problem-solving and critical thinking to come up with a work-arounds or alternatives to improve the original script and make it their own. By personalising the script, it also gives the user greater control over the script as they would know the ins and outs of it better. It also highlights the gaps in one’s ability which gives vision to what needs to be practiced or improve on specifically. (286 words)

2) There are several points that can be improved in appendix 1. They will be discussed below:

* Defining a **function** to the script would enhance **maintainability** and **reliability** as it limits the operation to be done within and only when the function is called upon. Functions make the script more concise while also preventing interference within bigger scripts.

def shopping\_list(): would be at the top of the code and everything else is inputed with an indent below to keep it within the function.

* Removal of unnecessary f-string in print(f"Wrong product! Please try again.") in the first while loop improves **readability** as there is no reason for it to be there. Removal prevents confusion among users and programmers.
* Replacing break with continue in that same loop makes **quality** of the code better as well since it removes the need for users to re-run the code during invalid inputs
* Adding try loops with a float condition for the price would also improve the code. As mentioned, loops prevents complete termination due to invalid inputs. With a float condition, inputs are more accurate as the code only accepts a specific type of input as seen here:

while True: #added while loop for price of item + float condition

try:

price\_of\_item=float(input("How much is it (in SGD)? "))

break

except ValueError:

print("Wrong input! Please try again.")

* Headers would enhance maintainability and readability of the code. Headers/comments makes it more organised and debugging would be easier as it would not be a hassle to locate and pinpoint the cause of errors.

References:

Chen, K. (2018, March 22). Why we still can’t stop plagiarism in undergraduate computer science. *kevinchen*. September 12, 2023, https://kevinchen.co/blog/cant-stop-plagiarism-in-computer-science/

Bidgood, J., & Merrill, J. B. (2017, May 29). As computer coding classes swell, so does cheating. *New York Times*. Retrieved September 12, 2023, from https://www.nytimes.com/2017/05/29/us/computer-science-cheating.html?smid=url-share.

<https://discuss.codecademy.com/t/area-calculator/365169> (Retrieved September 12,2023)