|  |  |
| --- | --- |
|  | Logo  Description automatically generated |

**ANL252**

**Python for Data Analytics**

**Tutor Marked Assignment**

**Jul 2023 Presentation**

**Submitted By:**

|  |  |  |
| --- | --- | --- |
| S.N. | Name | PI. No |
| 1. | Ng Ling Ying | E2181019 |

**Tutorial Group: TG03**

**Submission Date: 12 Sep 2023**

# QUESTION 1

**Question 1**

Plagiarism in coding does happen, but it occurs mostly in educational settings *(Carmil, 2021).*

**Reasons for plagiarism**

1. **Academic pressure & and lack of understanding**

Students might feel pressure to submit assignments quickly which will lead them to copy codes online. They might also not understand the coding concept and borrow code without understanding.

1. **Easy access online**

In schools, students are assessed not only on the assessment but also their academic integrity which means that they have to submit an original work *(Carmil, 2021).* However, when they are coding at home or work, they are free to refer to any material and might copy without providing any credits.

**Avoiding the issue**

1. **Code commenting**

Schools can encourage students to use comments in their codes to explain where they borrow the code from. This will ensure that the students will provide the source and teach them about the importance of referring back to the source.

1. **Real-world case studies**

Teachers can create assignments that encourage problem-solving and solutions instead of straightforward tasks. They might be able to insert real-world scenarios to help students to think outside of the box.

189 words

# Question 1b)

**Python Code:**

# Python program for simple calculator

# Function to add two numbers

def add(num1, num2):

return num1 + num2

# Function to subtract two numbers

def subtract(num1, num2):

return num1 - num2

# Function to multiply two numbers

def multiply(num1, num2):

return num1 \* num2

# Function to divide two numbers

def divide(num1, num2):

return num1 / num2

print("Please select operation -\n" \

"1. Add\n" \

"2. Subtract\n" \

"3. Multiply\n" \

"4. Divide\n")

# Take input from the user

select = int(input("Select operations form 1, 2, 3, 4 :"))

number\_1 = int(input("Enter first number: "))

number\_2 = int(input("Enter second number: "))

if select == 1:

print(number\_1, "+", number\_2, "=",

add(number\_1, number\_2))

elif select == 2:

print(number\_1, "-", number\_2, "=",

subtract(number\_1, number\_2))

elif select == 3:

print(number\_1, "\*", number\_2, "=",

multiply(number\_1, number\_2))

elif select == 4:

print(number\_1, "/", number\_2, "=",

divide(number\_1, number\_2))

else:

print("Invalid input")

Source:

<https://www.geeksforgeeks.org/make-simple-calculator-using-python/>

(retrieved on 10 Sep 2023)

This Python code is a simple calculator program that allows the user to perform basic math calculations on two numbers.

* **Def add(num1, num2)**: will add the two numbers
* **Def subtract(num1, num2)**: will minus the two numbers
* **Def multiply(num1, num2)**: will times the two numbers
* **Def divide(num1, num2)**: will divide the two numbers

It prints a menu for users to select the operation they want to perform. The users should select one of the operations (1, 2, 3, or 4) and prompt the user to enter two numbers.

Depending on the user's selection, the code will perform the operation using the appropriate function and display the result. It uses conditional statements (**if**, Elif, and **else**) to determine which operation to perform based on the user's input *(tutorialspoint, n.d.)*. If the user enters an invalid choice, it displays "Invalid input."

An **input()** function takes user input for selecting the operation and entering the two numbers *(w3schools).* The **int()** function is used to convert the input to integers. It displays the selected operation, two input numbers, and the result of the operation.

*179 words*

# Question 1c)

**Python Code:**

history = []

def numbers(prompt):

while True:

try:

number = float(input(prompt))

return number

except ValueError:

print("Please enter a valid number.")

def operators():

operator\_mapping = {

'+': lambda number1, number2: number1 + number2,

'-': lambda number1, number2: number1 - number2,

'\*': lambda number1, number2: number1 \* number2,

'/': lambda number1, number2: number1 / number2 if number2 != 0 else None

}

while True:

operator = input('''

Choose the operations:

+ for plus

- for minus

\* for times

/ for divide

''')

if operator in operator\_mapping:

return operator, operator\_mapping[operator]

else:

print("Please enter a valid operator.")

def calculate():

number1 = numbers('Enter the first number: ')

operation, operation\_func = operators()

number2 = numbers('Enter the second number: ')

result = operation\_func(number1, number2)

if result is None:

print("Division by zero is not allowed.")

return

history.append((number1, operation, number2, result))

print('{} {} {} = {:.2f}'.format(number1, operation, number2, result))

while True:

calculate()

while True:

choice = input("Do you wish to continue? Please enter yes/no: ")

if choice in ['yes', 'no']:

break

else:

print("Please enter 'yes' or 'no'.")

if choice == 'no':

print('Calculation history:')

for entry in history:

number1, operation, number2, result = entry

print('{} {} {} = {:.2f}'.format(number1, operation, number2, result))

break

1. **Organisation and Error Handling:**

Functions like **numbers()** and **operators()** make the code more organized and easier to maintain. I have added error handling to the functions. The **numbers()** function includes a **try...except** block to catch invalid inputs (non-numeric values) and prompt the user to enter a valid number. The **operators()** function will help to ensure that the user's choice of operation is valid by checking it against a list of valid operators. If an invalid operator is entered, the program asks the user to provide a valid one. This prevents the program from attempting undefined operations.

1. **Calculation History**

I created **calculation\_history** to store the details of each calculation. After each calculation, the program offers the user the option to continue or exit. If the user chooses to continue, another calculation is performed and added to the history. This feature allows users to keep track of their calculations during their session. When the user decides to stop calculating by entering 'no,' the program displays the calculation history. This provides a convenient summary of all calculations made during the session.

1. **Improved Readability:**

The rewritten code provides clear and user-friendly prompts to guide the user through the process. For example, it asks the user for the first and second numbers and provides a list of valid operators. It uses a **while** loop to prompt the user for calculations until they decide to stop, enhancing the user's experience by eliminating the need to rerun the program for each calculation. The code is designed to be easily expandable. If users want to add more operations in the future, users can simply expand the **operators()** function and add another calculation within the **calculate()** function. This makes it straightforward to adapt the program to handle more complex calculations without major code restructuring.

299 words

# Question 2

**Python Code:**

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

query = 'yes'

shopping\_items = []

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

while query == 'yes' :

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

if item not in products:

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

continue

try:

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

except ValueError:

print("Invalid price. Please enter a valid price.")

continue

shopping\_items.append({'item': item, 'price': price\_of\_item})

query = input("Would you like to continue? (yes/no) ")

while query not in ['yes', 'no']:

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

query = input("Would you like to continue? (yes/no) ")

if query == 'no':

break

total\_price = sum(item['price'] for item in shopping\_items)

print(f'This is our updated shopping list:')

for item in shopping\_items:

print(f'{item ["item"]}: {item["price"]:.2f} SGD')

print(f'Total price: {total\_price:.2f} SGD')

A “try...except” block was added to handle errors when entering the price of an item and the “yes” or “no” options. This ensures that when a user inputs an invalid input, it will respond by providing a clear error message and prompting the user to re-enter a valid input. This prevents the program from unintentionally continuing with undefined inputs. Having an input validation will help to enhance the **reliability** of the program by guaranteeing that only valid and anticipated inputs are entered.

A feature was added to calculate and display the total price of shopping items. This will offer users a summary of their expenses, allowing them to keep track of their spending. This will help the user to make informed purchase decisions and stay within budget. Hence, this will improve the user’s **readability** and make the program more effective for managing shopping lists.

There is a consistent indentation that will allow the code to become less error-prone. If there were to be any edits made, there should be lesser indentation-related bugs, such as incorrect nesting or mismatched block structures. This adherence to conventions reduces the risk of introducing subtle errors that can be challenging to debug.

*197 Words*

**References**

Carmil, C. (2021, July 9). *All you need to know about plagiarism in coding*. https://copyleaks.com/blog/all-you-need-to-know-about-plagiarism-in-coding

tutorialspoint. (n.d.). *Python if...ELIF...else statements*. Online Courses and eBooks Library. https://www.tutorialspoint.com/python/python\_if\_else.htm

w3schools. (n.d.). Python input() function. https://www.w3schools.com/python/ref\_func\_input.asp