

K. J. Somaiya College of Engineering, Mumbai-77

Batch: C2-2 **Roll No.: 047**
Experiment / assignment / tutorial No.
Grade: AA / AB / BB / BC / CC / CD / DD
Signature of the Staff In-charge with date

TITLE: Write a program to demonstrate the use of decision-making statements in Python.

AIM: 1) Write a program to read the numbers until -1 is encountered. Also, count the number of prime and composite numbers entered by the user.
2) Write a program to check whether a given number is Armstrong.

Outcome: Students will be able to

CO1: Formulate a problem statement and develop the logic (algorithm/flowchart) for its solution.

CO3: Use different decision-making statements and functions in Python.

Use the input-output function. And different decision-making statements in Python.

Resource Needed: Python IDE

Books/journals/websites referred:

1. Reema Thareja, *Python Programming: Using Problem-Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A Modular Approach*, Pearson India, Second Edition 2018, India
3. <https://www.geeksforgeeks.org/python-strings/?ref=lbp>

Theory:

Decision Control Statements

1) Selection/Conditional branching statements

- a) if statement
- b) if-else statement
- c) if-elif-else statement

2) Basic loop structures/iterative statements

- a) while loop
- b) for loop

If statement:

In Python, the **if** statement is used for decision-making operations. It contains a body of code that runs only when the condition given in the **if** statement is true.

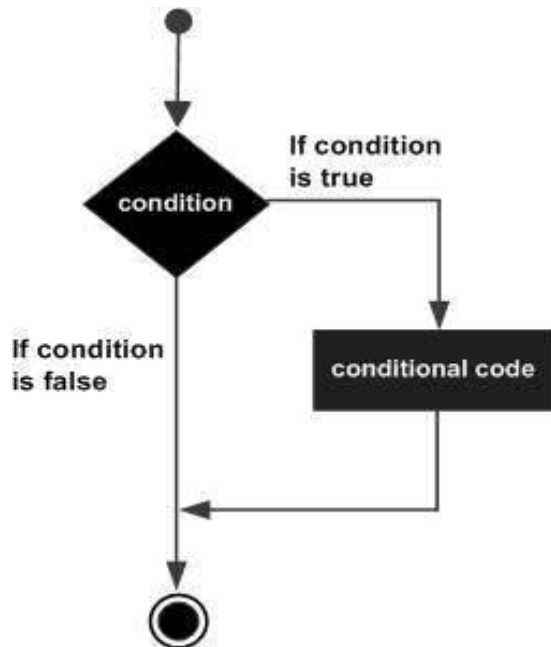
Department of Department of Science and Humanities

K. J. Somaiya College of Engineering, Mumbai-77

Syntax:

```
if condition:
    statement(s)
```

If flowchart:



If-else Statement:

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to 0 or a FALSE value.

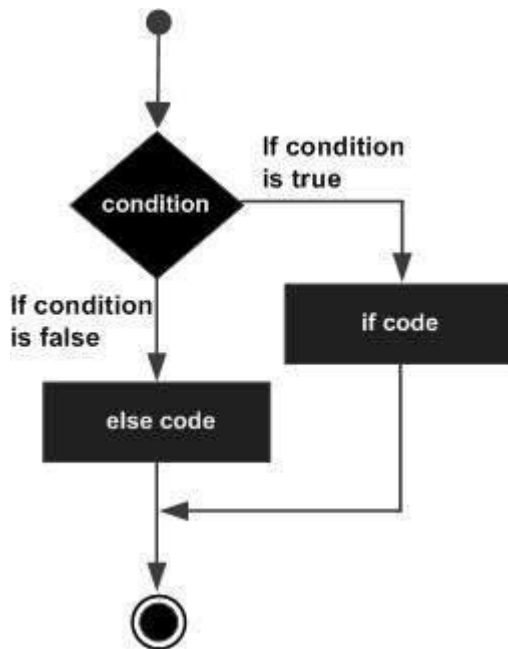
The **else** statement is an optional statement, and there could be at most only one **else** statement following the **if**.

Syntax:

```
if expression:
    statement(s)
else:
    statement(s)
```

K. J. Somaiya College of Engineering, Mumbai-77

If-else flowchart:



If-elif-else Statement:

The **elif** statement allows you to check multiple expressions for TRUE and execute a code block as soon as one of the conditions evaluates to TRUE.

Similar to the else statement, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

Syntax:

```

if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
  
```

While loop:

A **while** loop statement in the Python programming language repeatedly executes a target statement as long as a given condition is true.

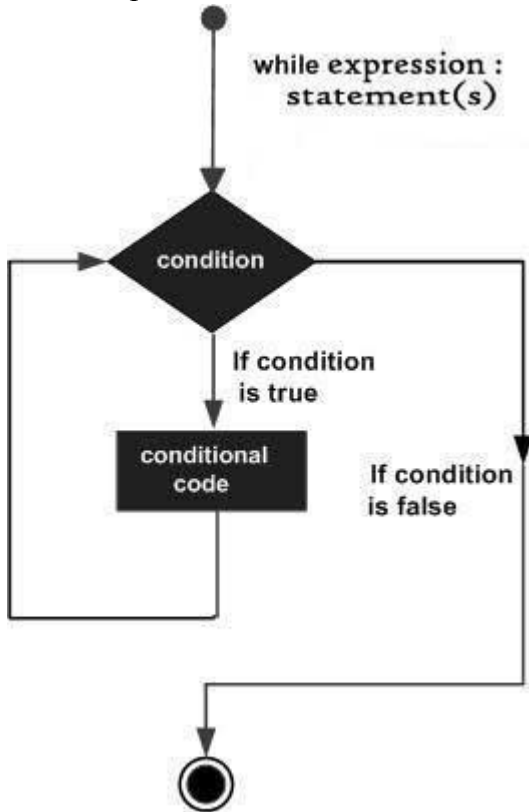
Syntax:

```

while expression:
    statement(s)
  
```

K. J. Somaiya College of Engineering, Mumbai-77

While loop flowchart:



For Loop:

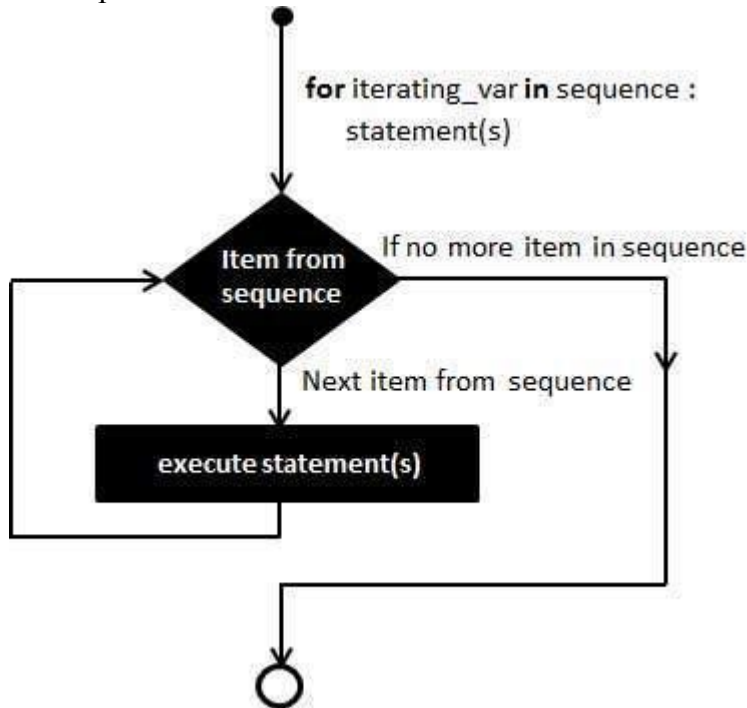
The **for** statement in Python differs a bit from what you may be used to in C. Rather than giving the user the ability to define both the iteration step and halting condition (as C), Python's **for** statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

Syntax:

```
for iterating_var in sequence:
    statements(s)
```

K. J. Somaiya College of Engineering, Mumbai-77

For loop flowchart:



Problem Definition:

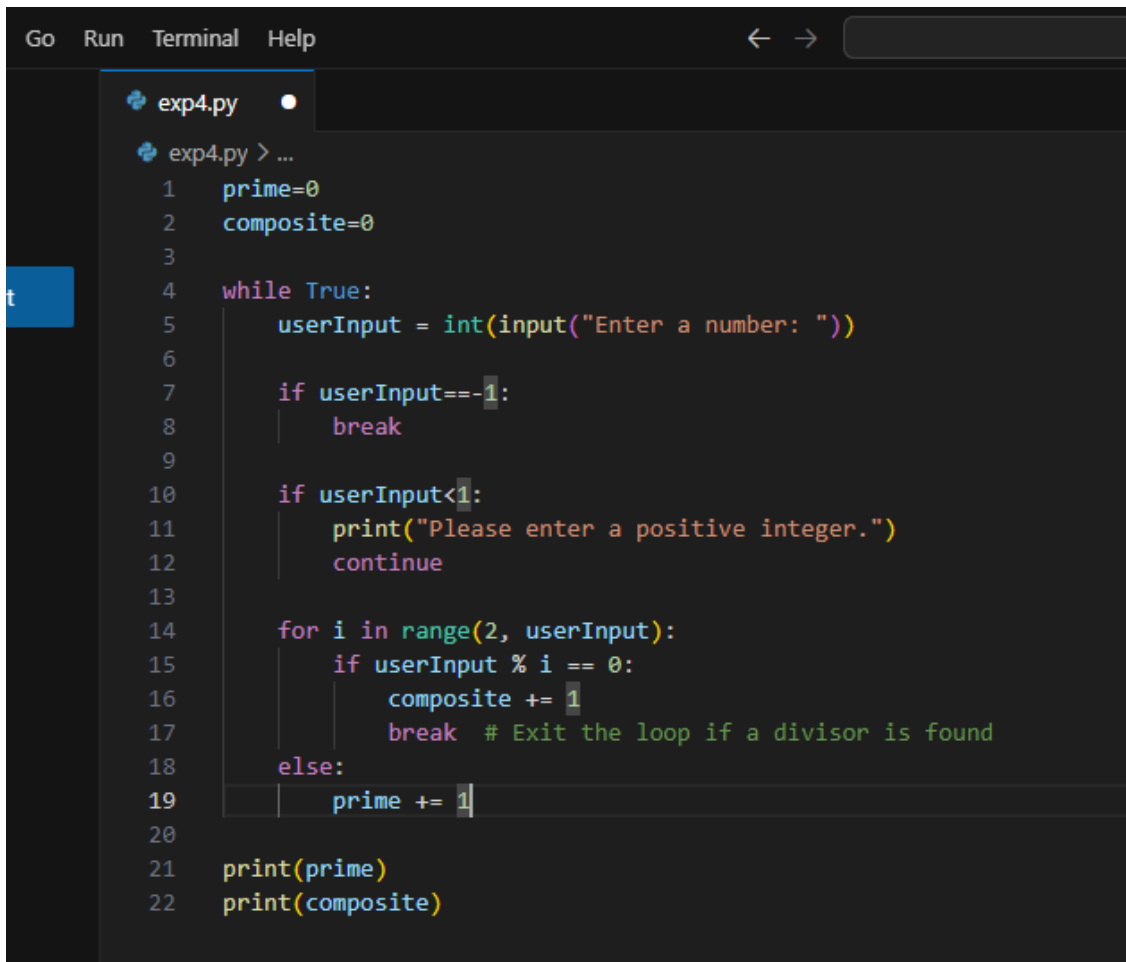
- 1) Write a program to read the numbers until -1 is encountered. Also, count the number of prime and composite numbers entered by the user.
- 2) Write a program to check whether a number is Armstrong or not.
(An Armstrong number is a number equal to the sum of cubes of its digits; for example, $153 = 1^3 + 5^3 + 3^3$.)

Implementation details:

Problem 1:

Code:

K. J. Somaiya College of Engineering, Mumbai-77



```

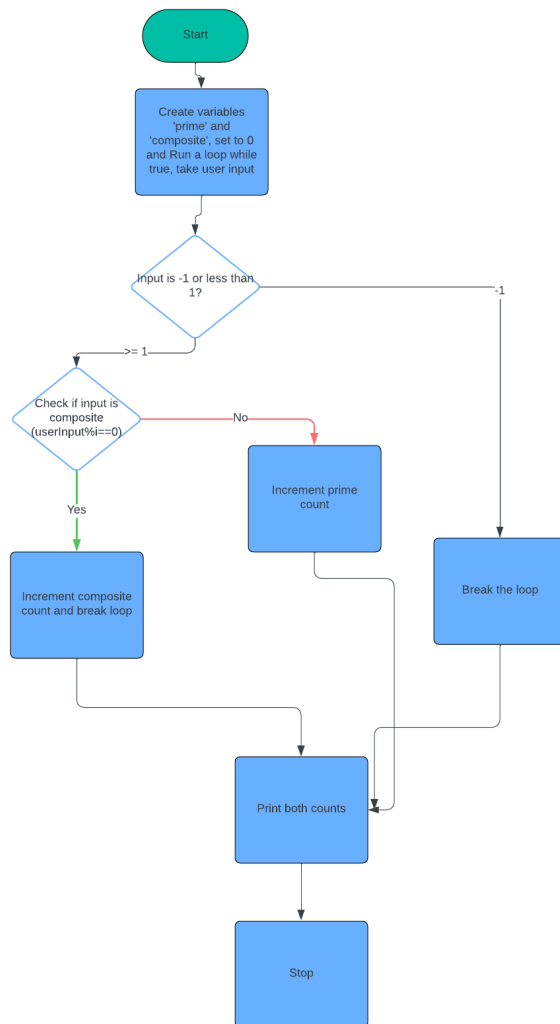
Go Run Terminal Help
exp4.py
exp4.py > ...
1 prime=0
2 composite=0
3
4 while True:
5     userInput = int(input("Enter a number: "))
6
7     if userInput== -1:
8         break
9
10    if userInput<1:
11        print("Please enter a positive integer.")
12        continue
13
14    for i in range(2, userInput):
15        if userInput % i == 0:
16            composite += 1
17            break # Exit the loop if a divisor is found
18    else:
19        prime += 1
20
21 print(prime)
22 print(composite)
  
```

Algorithm:

1. Start
2. Create a variable for prime and composite and set it to 0.
3. Run a loop while true
4. Take inputs from user.
5. If the input is -1, break out of the loop
6. If the input is less than 1, prompt the user.
7. Run a loop from 2 to the input and see if it is a composite number using $\text{userInput} \% i == 0$ logic. If yes, increment composite count and break.
8. If not, then increment the prime count.
9. Print both counts.
10. Stop.

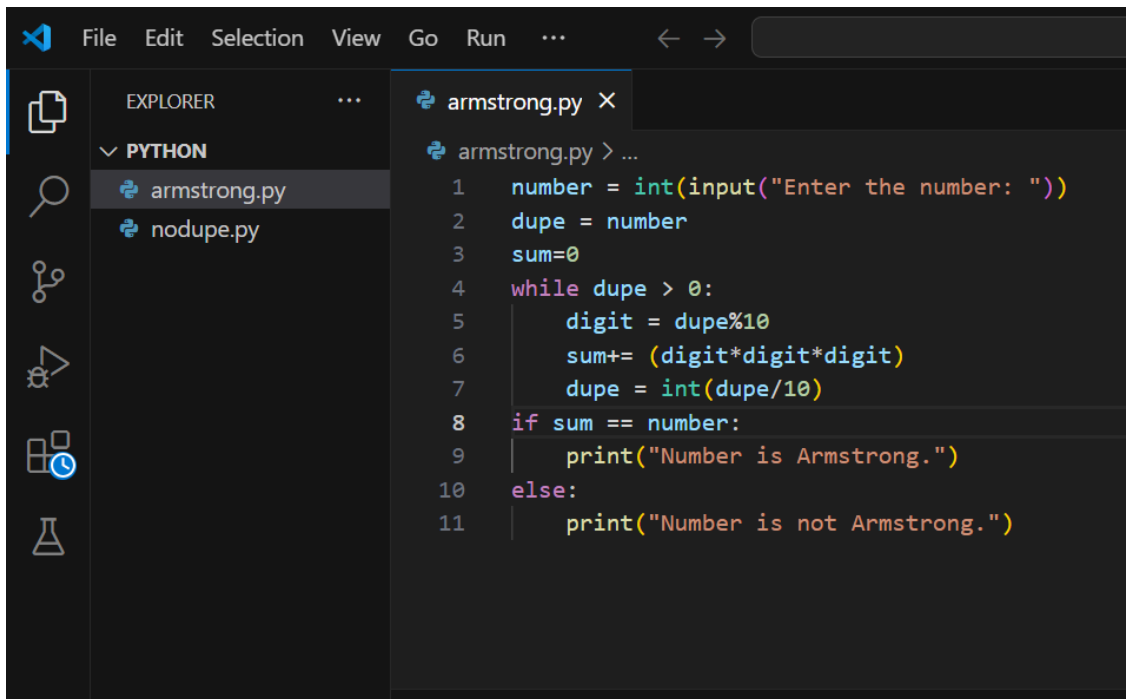
Flowchart:

K. J. Somaiya College of Engineering, Mumbai-77



Problem 2:
Code:

K. J. Somaiya College of Engineering, Mumbai-77



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project named 'PYTHON' containing two files: 'armstrong.py' and 'nodupe.py'. The 'armstrong.py' file is open in the editor, showing the following code:

```

1  number = int(input("Enter the number: "))
2  dupe = number
3  sum=0
4  while dupe > 0:
5      digit = dupe%10
6      sum+= (digit*digit*digit)
7      dupe = int(dupe/10)
8  if sum == number:
9      print("Number is Armstrong.")
10 else:
11     print("Number is not Armstrong.")

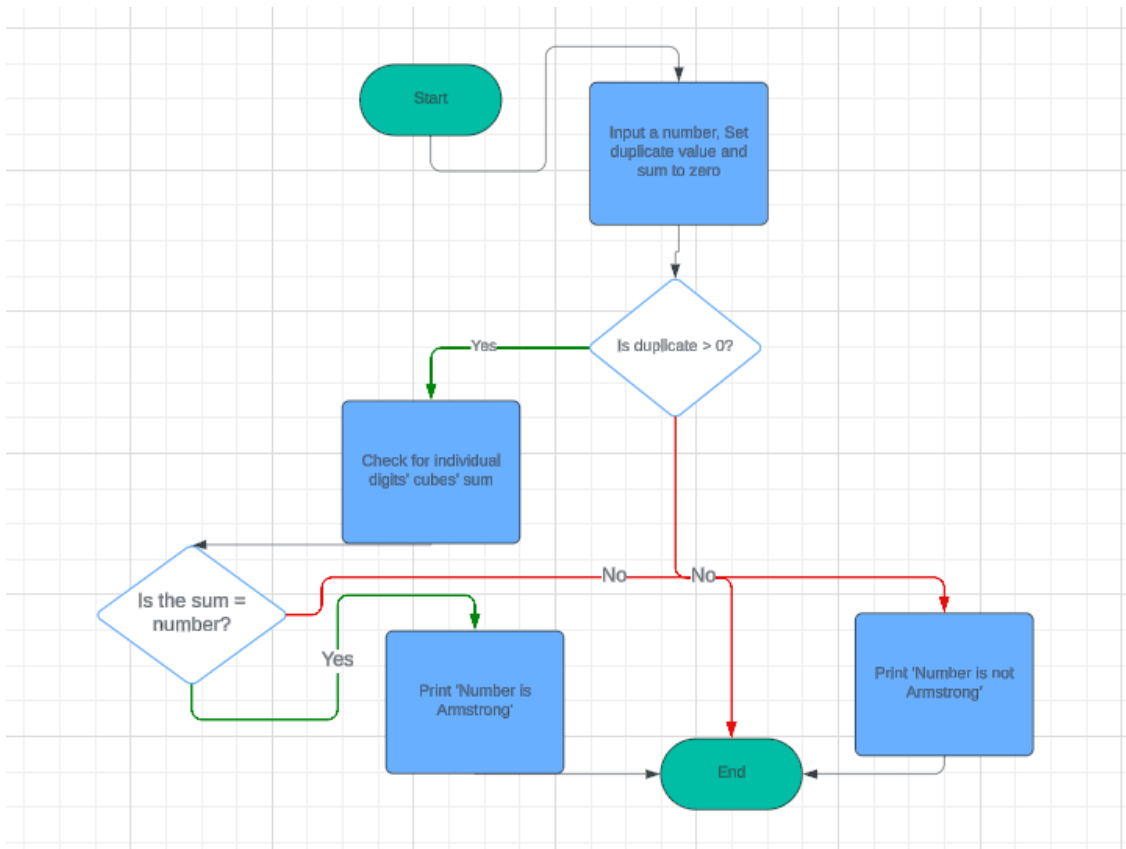
```

Algorithm:

1. Start
2. Input a number.
3. Set it to a duplicate value.
4. Set sum to zero.
5. Run a loop till the duplicate is more than zero.
6. Inside the loop, check for individual digits' cubes' sum.
7. At last, compare the sum to the original number.
8. If the sum and number are same, print "Number is Armstrong", else print "Number is not Armstrong."
9. Stop.

Flowchart:

K. J. Somaiya College of Engineering, Mumbai-77



PS: The flowchart has been spread out awkwardly to make sure it's legible. In the traditional format, the text was blurry when attaching screenshots.

Output(s):

```

C:\Users\Student\Documents\c3-58> debugpy\launcher 58542 -- c:\u
Enter a number: 5
Enter a number: 2
Enter a number: 6
Enter a number: 9
Enter a number: -1
2
2
PS C:\Users\Student\Documents\c3-58> ^C
PS C:\Users\Student\Documents\c3-58>
PS C:\Users\Student\Documents\c3-58> c:: c

```

K. J. Somaiya College of Engineering, Mumbai-77

```

1\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\
bs\debugpy\adapter\..\..\debugpy\launcher' '50810
Enter the number: 153
Number is Armstrong.
PS C:\Users\syeda\OneDrive\Desktop\Python>
PS C:\Users\syeda\OneDrive\Desktop\Python> ^C
PS C:\Users\syeda\OneDrive\Desktop\Python>
PS C:\Users\syeda\OneDrive\Desktop\Python> c::; cd
1\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\
bs\debugpy\adapter\..\..\debugpy\launcher' '50824
Enter the number: 150
Number is not Armstrong.
PS C:\Users\syeda\OneDrive\Desktop\Python>
: Ready

```

Conclusion:

Loops in python help us iterate over values for some number of times to reach desired results. In the armstrong problem, we went over the input again and again to be able to extract digits and cube them to sum together. For the prime and composite numbers, we went through the loop again and again to make sure we are reading all numbers till we encounter a -1.

Post Lab Questions:

1) When should we use nested if statements? Illustrate your answer with an example.

Nested if statements are used when more than one conditions need to be checked within each other. This is when a bigger criteria can do elimination first and not require the compiler to go through all ifs unnecessarily. For example, only positive numbers can be prime. So to check if a number is prime, we could foremost check if it's positive or negative, and only if it's positive do we continue to check if it's prime by counting its factors. This way, the code becomes more efficient. Or when we are checking if a character is a vowel or a consonant, it is helpful to first see if it's an alphabet and only then move on and see if it's a vowel.

2) Explain the utility of break and continue statements with the help of an example.

The break statement in loops and if statements help us to break prematurely out of a code/block if a certain condition is met. For example, we could be looking for a specific character in a string. If we find it at the i^{th} index, we need not to look further and can terminate the code/block or “**break**” out of it. Here, we can use break.

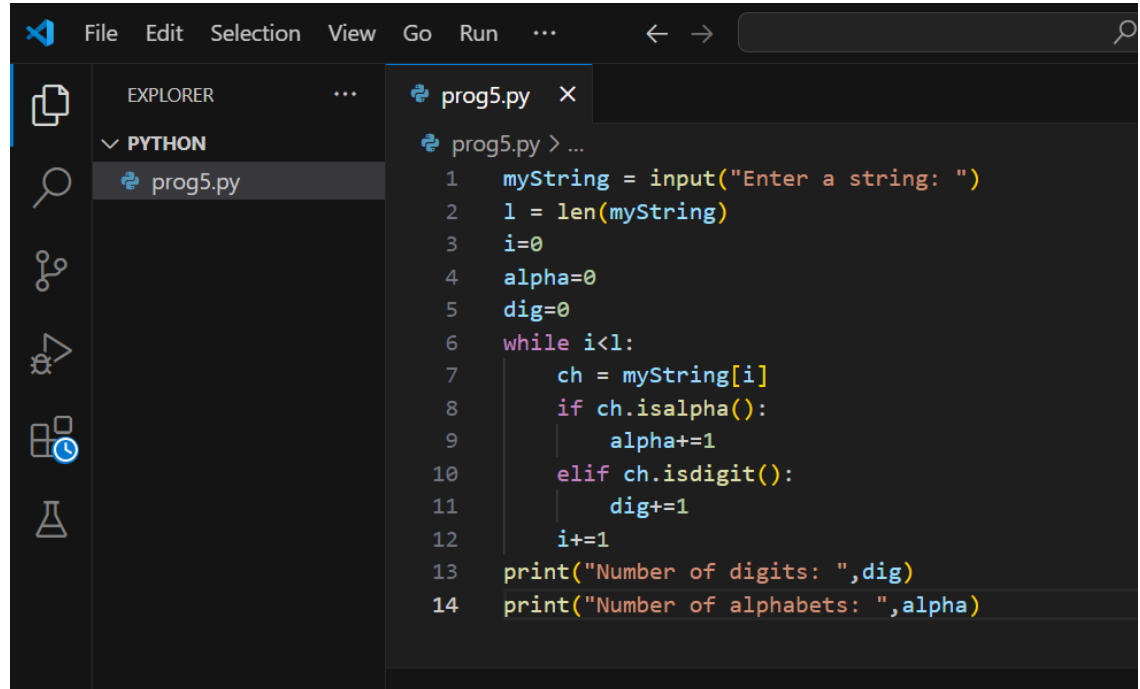
The continue statement is helpful in not letting the program break prematurely incase of unexpected inputs. For example, if a user enters unexpected value, instead of terminating the code altogether, we pass them a warning and “**continue**” the code, so that they can input again and the code goes on.

Department of Department of Science and Humanities

K. J. Somaiya College of Engineering, Mumbai-77

3) Write a program that accepts a string from the user and calculates the number of digits and letters in the string.

Code:

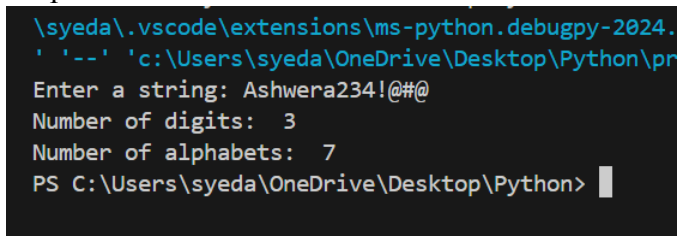


```

1  myString = input("Enter a string: ")
2  l = len(myString)
3  i=0
4  alpha=0
5  dig=0
6  while i<l:
7      ch = myString[i]
8      if ch.isalpha():
9          alpha+=1
10         elif ch.isdigit():
11             dig+=1
12         i+=1
13     print("Number of digits: ",dig)
14     print("Number of alphabets: ",alpha)

```

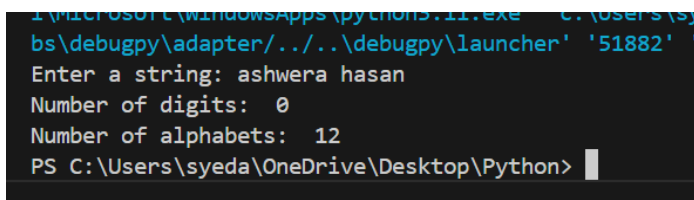
Output:



```

\syeda\.vscode\extensions\ms-python.debugpy-2024.
' '--' 'c:\Users\syeda\OneDrive\Desktop\Python\pr
Enter a string: Ashwera234!@##@
Number of digits: 3
Number of alphabets: 7
PS C:\Users\syeda\OneDrive\Desktop\Python>

```



```

I:\Microsoft\WindowsApps\python3.11.exe -c: \Users\sy
bs\debugpy\adapter\..\..\debugpy\launcher' '51882' '
Enter a string: ashwera hasan
Number of digits: 0
Number of alphabets: 12
PS C:\Users\syeda\OneDrive\Desktop\Python>

```

K. J. Somaiya College of Engineering, Mumbai-77

```
PS C:\Users\syeda\OneDrive\Desktop\Python> c::;  
1\Microsoft\WindowsApps\python3.11.exe' 'c:\User  
bs\debugpy\adapter\..\..\debugpy\launcher' '5189  
Enter a string: 34567890  
Number of digits: 8  
Number of alphabets: 0  
PS C:\Users\syeda\OneDrive\Desktop\Python> |
```