Batch: C1_2	**Roll No.: 38**
Experiment / assignment / tutorial No. -3	

TITLE: Decision Making Statements

AIM: 1) Write a program to count the number of prime numbers and composite numbers entered by the user.
2) Write a program to check whether a given number is Armstrong or not.

Expected OUTCOME of Experiment: Use different Decision Making statements in Python.

Resource Needed: Python IDE

Theory:

Decision Control Statements

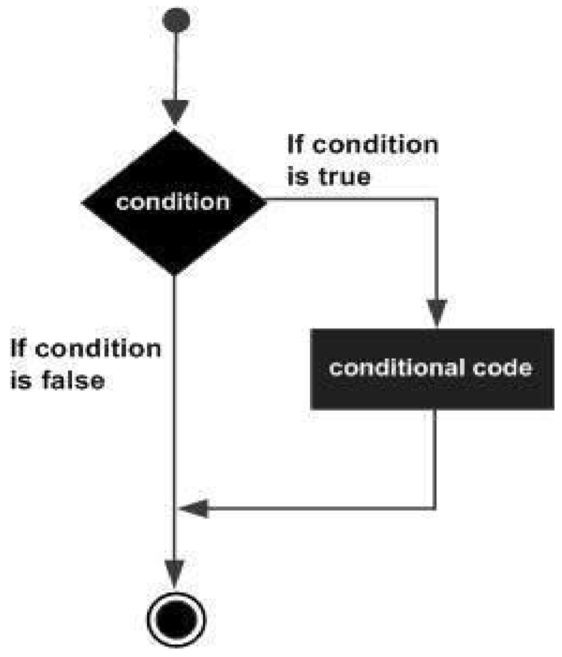
1) Selection/Conditional branching statements
- a) if statement
- b) if-else statement
- c) if-elif-else statement

2)Basic loop Structures/Iterative statement
- a) while loop
- b) for loop

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

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

If flowchart:



If-else Statement:

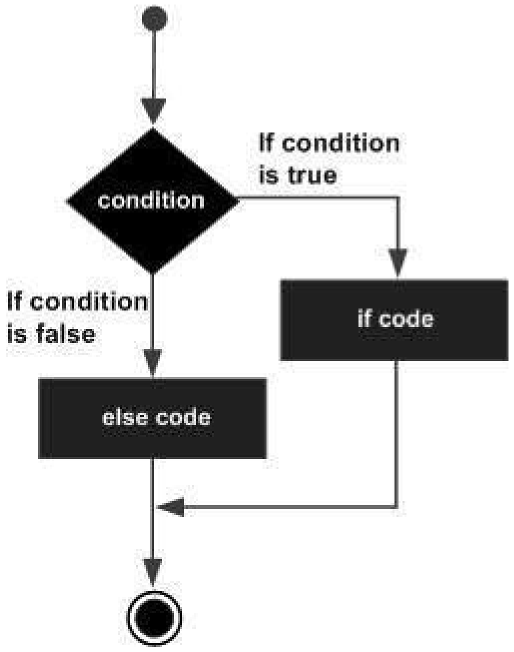
An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to 0 or a FALSE value.

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

Syntax:

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

If-else flowchart:



If-elif-else Statement:

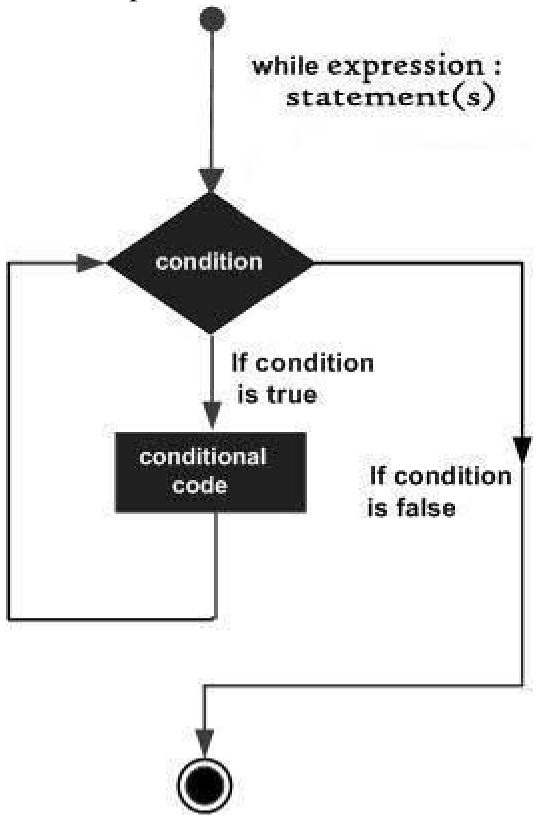
The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
Similar to the else, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary 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 Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:
```
while expression:
   statement(s)
```

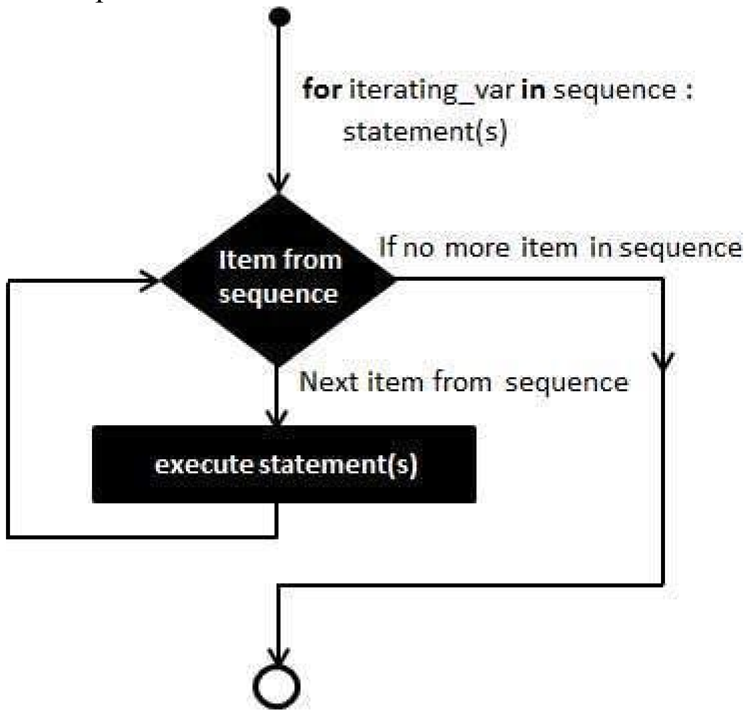
While loop flowchart:



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)
```

For loop flowchart:



Problem Definition:

1) Write a program to read the numbers until -1 is encountered. Also, count the number of prime numbers and composite numbers entered by the user

2) Write a program to check whether a number is Armstrong or not.
(Armstrong number is a number that is equal to the sum of cubes of its digits for example: 153 = 1^3 + 5^3 + 3^3.)

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://docs.python.org/3/tutorial/controlflow.html#for-statements

Implementation details:

1)

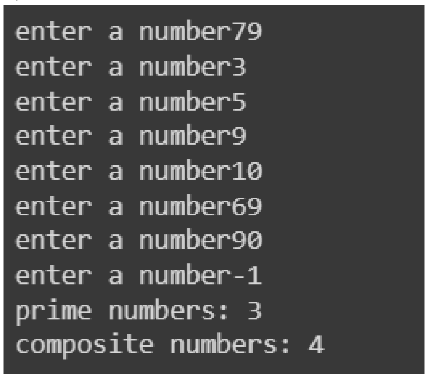
```
from sympy import *
c=0
c_p=0
c_c=0
n=0
while n!=1:
  n=int(input("enter a number"))
  if n==-1:
    break
  if isprime(n)==True:
    c_p+=1
  else:
    c_c+=1

print("prime numbers:",c_p)
print("composite numbers:",c_c)
```

2)

```
def is_armstrong_number(number):
    # Convert the number to a string to iterate through its digits
    num_str = str(number)

    # Calculate the number of digits in the number
    num_digits = len(num_str)

    # Initialize a variable to store the sum of cubes of digits
    armstrong_sum = 0

    # Iterate through each digit and add its cube to the sum
    for digit in num_str:
        armstrong_sum += int(digit) ** num_digits

    # Check if the sum is equal to the original number
```

```python
    if armstrong_sum == number:
        return True
    else:
        return False

# Input from the user
num = int(input("Enter a number: "))

# Check if the number is an Armstrong number
if is_armstrong_number(num):
    print(num, "is an Armstrong number.")
else:
    print(num, "is not an Armstrong number.")
```

Output(s):
1)



```
enter a number79
enter a number3
enter a number5
enter a number9
enter a number10
enter a number69
enter a number90
enter a number-1
prime numbers: 3
composite numbers: 4
```

Conclusion:
Learnt the count of prime and composite numbers through while loops and also understood the use of nested if else,for loops,break and continue statement

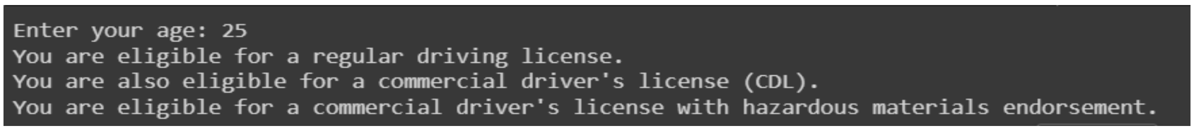
Post Lab Questions:

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

Ans-Nested if statements should be used when you need to create more complex conditional logic in your code, where one condition depends on the outcome of another condition. They allow you to check multiple conditions sequentially, and the inner if statements are only evaluated if the outer if statements are true. Nested if statements are useful for handling situations with multiple levels of decision-making.

Here's an example to illustrate when to use nested if statements. Suppose you want to build a program that checks a person's eligibility for different types of driving licenses based on their age.

```
age = int(input("Enter your age: "))

if age >= 18:
    print("You are eligible for a regular driving license.")

    if age >= 21:
        print("You are also eligible for a commercial driver's
license (CDL).")

        if age >= 25:
            print("You are eligible for a commercial driver's
license with hazardous materials endorsement.")
    else:
        print("You are not eligible for a CDL yet.")
else:
    print("You are not eligible for any type of driving license.")
```



```
Enter your age: 25
You are eligible for a regular driving license.
You are also eligible for a commercial driver's license (CDL).
You are eligible for a commercial driver's license with hazardous materials endorsement.
```

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

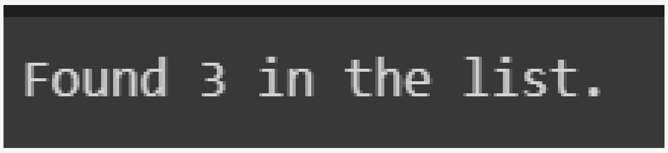
Ans: The break and continue statements are control flow statements in Python that are used to alter the flow of a loop. They serve different purposes and can be quite useful in specific scenarios.

The break statement:

- The break statement is used to exit a loop prematurely, before its normal termination condition is met.
- It is typically used when you want to stop the loop as soon as a certain condition is satisfied.
- It is often used to terminate a loop early when searching for a specific item or when a certain condition is met.

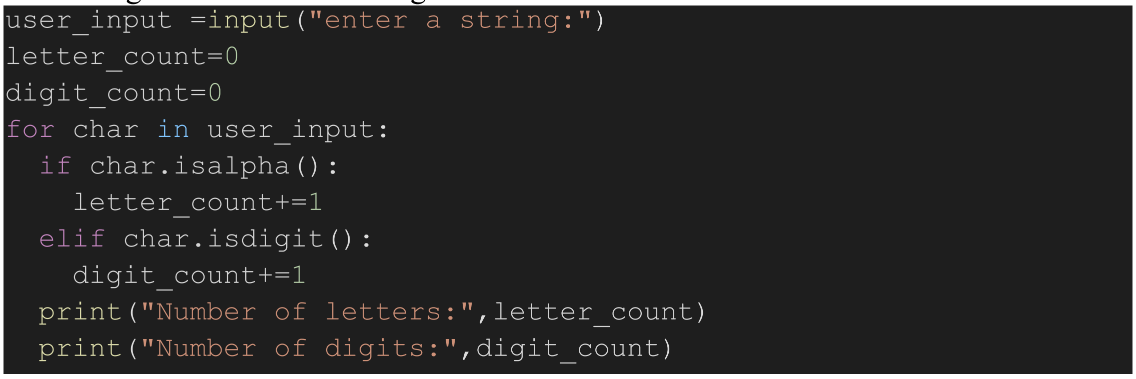
Here's an example illustrating the use of the break statement:

```python
# Search for a specific number in a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_number = 3

for number in numbers:
    if number == search_number:
        print(f"Found {search_number} in the list.")
        break
else:
    print(f"{search_number} not found in the list.")
```

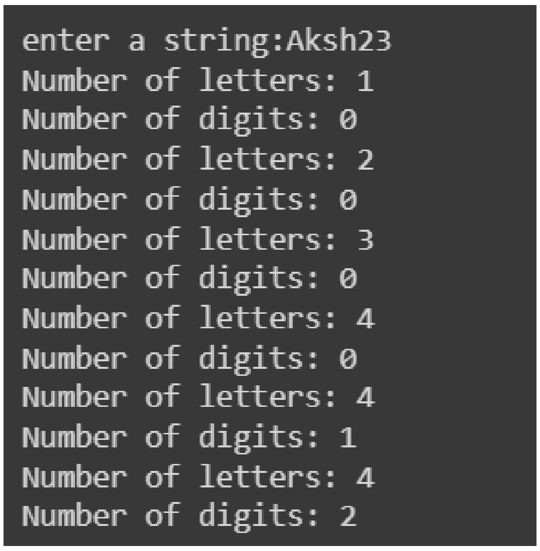


```
Found 3 in the list.
```

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



```
user_input =input("enter a string:")
letter_count=0
digit_count=0
for char in user_input:
  if char.isalpha():
    letter_count+=1
  elif char.isdigit():
    digit_count+=1
  print("Number of letters:",letter_count)
  print("Number of digits:",digit_count)
```



```
enter a string:Aksh23
Number of letters: 1
Number of digits: 0
Number of letters: 2
Number of digits: 0
Number of letters: 3
Number of digits: 0
Number of letters: 4
Number of digits: 0
Number of letters: 4
Number of digits: 1
Number of letters: 4
Number of digits: 2
```