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:

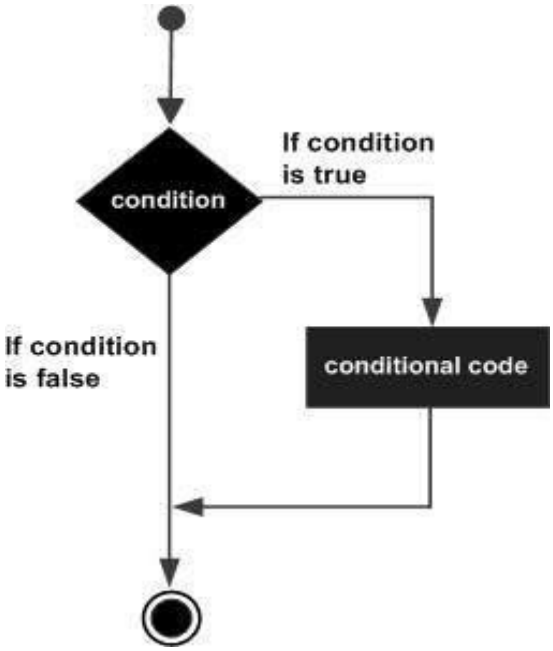
Decision Control Statements

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:

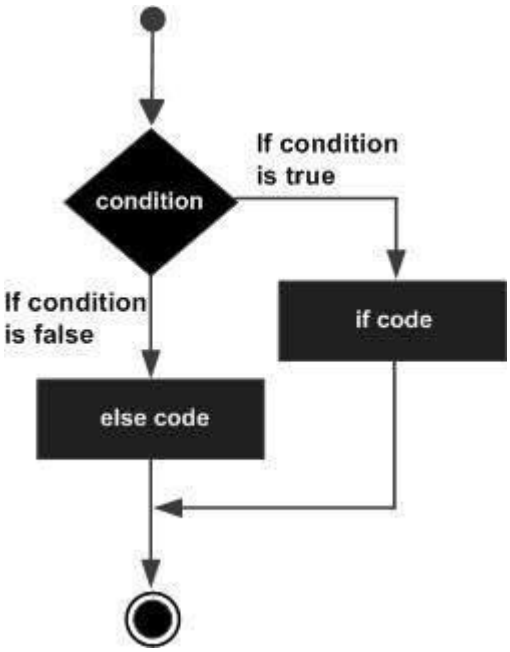
An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to 0 or a FALSE value.

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.**

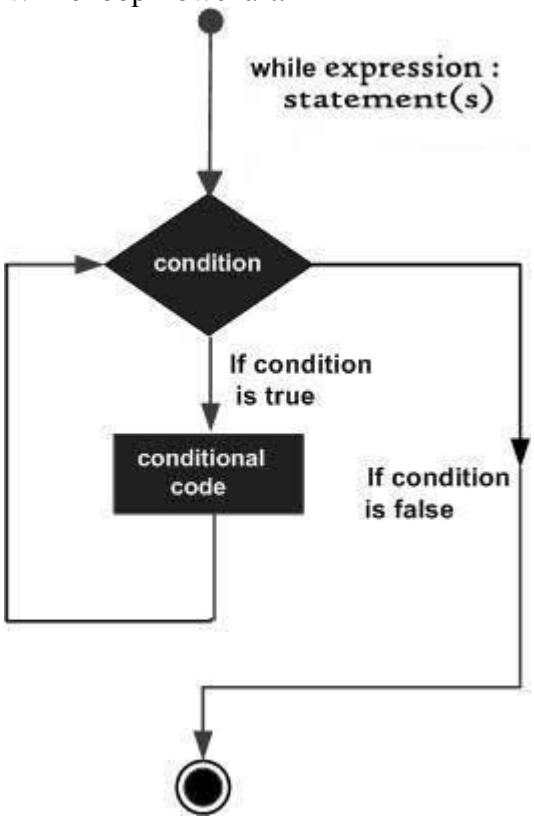
Syntax:

```
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:

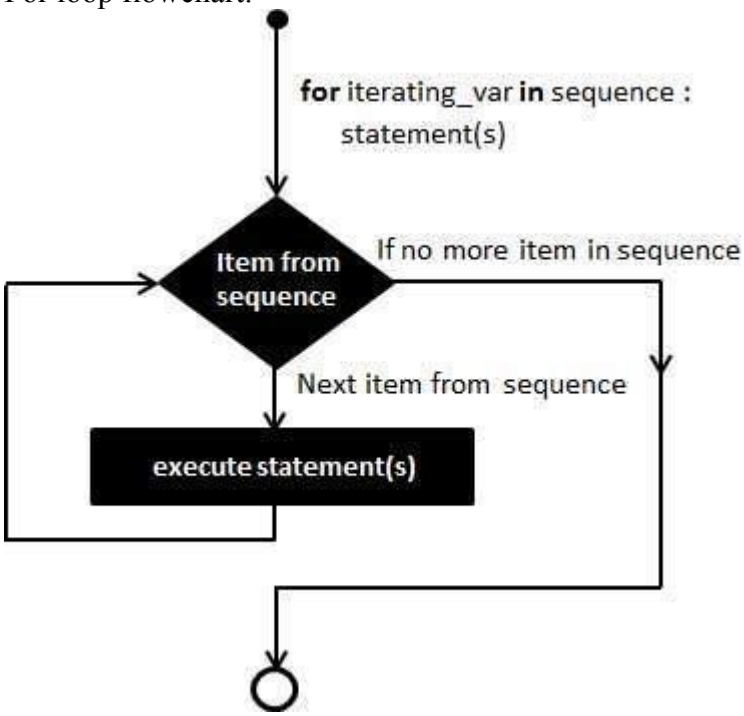


For Loop:
The **for** statement in Python differs a bit from what you may be used to in C. Rather than giving the user the ability to define both the iteration step and halting condition (as C), Python's **for** statement iterates over the items of any sequence (a list or a string), in the order that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

1st Code:

```
def prime_check(num):
    count=0
    for i in range(1,num+1):
        if num%i==0:
            count+=1

    if count<=2:
        return True
    else:
        return False
print("Enter numbers : \n(Exit with -1)")
num=int(input())
prime=0
composite=0

while num!=-1:
    if prime_check(num)==True:
        prime+=1
    else:
        composite+=1
    num=int(input("Enter numbers :"))

print("No .of Prime numbers entered are:" + str(prime))
print("No .of Composite numbers entered are:" + str(composite))
```

2nd Code:

```
num = int(input("Enter the Number to check if Armstrong: \n "))
power = 3
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** power
    temp //= 10
if num == sum:
    print(num,"is an Armstrong number")
else:
    print(num,"is not an Armstrong number")
```

Output(s):

1st Output:

```
Enter numbers :
(Exit with -1)
3
Enter numbers :2
Enter numbers :7
Enter numbers :6
Enter numbers :5
Enter numbers :9
Enter numbers :8
Enter numbers :11
Enter numbers :111
Enter numbers :1435
Enter numbers :5776
Enter numbers :-1
No .of Prime numbers entered are:5
No .of Composite numbers entered are:6
```

2nd Output:

```
Enter the Number to check if Armstrong:
 34
34 is not an Armstrong number
PS C:\Users\Pratham> python -u "c:\Users\Pratham\Dropbox\PC\Documents\py code\exp 3 -2.py"
Enter the Number to check if Armstrong:
 153
153 is an Armstrong number
```

Conclusion:

1) We learnt using conditional statements and loops to find if number are prime or armstrong .
2) We also learnt separating digits from a number to use.

Post Lab Questions:

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

-We should use nested if statements when we want the answer of a statement to depend on the previous statement such as while determining if a number is greater than one number or both the numbers.

```
n = 5
if (n == 5):
    if (i < 7):
        print ("n is smaller than 7")
    if (i < 3):
        print ("n is smaller than 3 too")
    else:
        print ("n is greater than 3 and smaller than 7")
```

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

-Break statement is used to skip out of the whole loop whenever a condition is satisfied. An example of its utility is to limit the loop to a certain extent, like in the following code-

print("This is a program to print multiples of any number to any extent you want!")

```
x = int(input("Enter the number you want multiples of-"))

y = int(input("Enter number of multiples-"))

i = 1

while True:

  print(x, "*", i, "=", x*i)

  i += 1

  if i == y+1:

    break
```

Output-

```
This is a program to print multiples of any number to any extent you want!
Enter the number you want multiples of-4
Enter number of multiples-6
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
```

A continue statement, on the other hand skips the satisfied iteration of the loop. For example-

```
x = int(input("Enter a number to print the odd numbers till it is reached-"))

i = 1

for i in range(0, x):

    if i % 2 == 0:

        continue

    print(i)
```

Output-

```
Enter a number to print the odd numbers till it is reached-10
1
3
5
7
9
```

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

-Code

```
    string    input("Input a string: \n")
Digit=Letter=0
for char in string:
    if char.isdigit():
        Digit+=1
    elif char.isalpha():
        Letter+=1
```

```
    else:
        pass
print("Letters= \n", Letter)
print("Digits=", Digit)
```

Output-

```
Input a string:
Hello % 4 92929 world
Letters=
 10
Digits= 6
```

Date: _____ **Signature of faculty in-charge**