Flow Control

- The order of execution of the statements in a program is known as flow of control.
- The flow of control can be implemented using control structures.
- Python supports two types of control structures—
  - **Decision making** for selection
  - **Looping** for repetition

A) Decision Making

- Decision making evaluate multiple expressions which produce TRUE or FALSE as outcome.
- It needs to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.
- Python programming language provides following types of decision-making statements.
  - **if statements**
  - **Else statements**
  - **Elif statements**
  - **nested if statements**

1] If statement

- An "if statement" is written by using the if keyword.
- If the condition is true, then the indented statement(s) are executed.
- Syntax:

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

Example:

```
age = int(input("Enter your age "))
if age >= 18:
        print("Eligible to vote")
```

Example:

```
a = 200
b = 33
c = 500
if a > b and c > a:
        print("Both conditions are True")
```

2] Else statement

- A variant of if statement called if..else statement.
- It allows us to write two alternative paths and the control condition determines which path gets executed.
- The else keyword catches anything which is not caught by the preceding conditions.
- Syntax:

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

Example:

```
age = int(input("Enter your age: "))
if age >= 18:
  print("Eligible to vote")
else:
  print("Not eligible to vote")
```

Example:

```
#Program to find larger of the two numbers
num1 = 5
num2 = 6
if num1 > num2:
  print("first number is larger")
  print("Bye")
else:
  print("second number is larger")
  print("Bye Bye")
```

Output:

```
second number is larger
Bye Bye
```

3] Elif statement

- Many a times there are situations that require multiple conditions to be checked and it may lead to
- many alternatives.
- In such cases we can chain the conditions using elif means (else..if).
- The elif keyword - "if the previous conditions were not true, then try this condition".
- **Syntax:**

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

Example

```
number = int(input("Enter a number: ")
if number > 0:
   print("Number is positive")
elif number < 0:
   print("Number is negative")
else:
  print("Number is zero")
```

B) Looping

- Looping allows to execute a set of statements in a program multiple times repetitively, based on a condition.
- The statements in a loop are executed again and again as long as particular logical condition remains true. When the condition becomes false, the loop terminates.
- Python programming language provides the following types of loops to handle looping requirements.
  - **while loop**
  - **for loop**
  - **nested loop**

1] while Loop

- It executes a set of statement as long as the given condition is TRUE.
- It tests the condition before executing the loop body.
- Syntax:

```
while test_condition:
  body of while
```

Note:

Using else statement with while loops: A while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed. The else clause (if used) is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

Syntax:

```
while condition:
    # execute these statements
else:
    # execute these statements
```

Example:

```
#Python program to illustrate combining else with while
count = 0
while (count < 3):
   count = count + 1
   print("Hello")
else:
   print("In Else Block")
```

```
Output
Hello
Hello
Hello
In Else Block
```

```
#Print first 5 natural numbers using while loop
count = 1
while count <= 5:
   print(count)
   count += 1
```

Output:

```
1
2
3
4
5
```

2] **'For' Loop**

- for loop executes a set of statements multiple times.
- A for loop is used for iterating over each value in a sequence (i.e. either a list, a tuple, a dictionary, a set, or a string).
- A control variable can be used to count the executions. A sequence holds multiple items of data, stored one after the other
- The variable takes on each successive value in the sequence, and the statements in the body of the loop are executed once for each value.
- The function range() is often used in for loops for generating a sequence of numbers.
- **Syntax:**

```
for <control-variable> in <sequence> <items in range>:
    <statements inside body of the loop>
```

OR

```
for i in range(initialValue, endValue):
  #Loop body
```

```
#Print each fruit in a fruit list:
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Output:

```
apple
banana
cherry
```

```
#Program to print the characters in the string 'PYTHON' using for loop.
#Print the characters in word PYTHON using for loop
for letter in 'PYTHON':
        print(letter)
```

Output:

```
P
Y
T
H
O
N
```

```
#Program to print even numbers in a given sequence using for loop.
numbers = [1,2,3,4,5,6,7,8,9,10]
for num in numbers:
        if (num % 2) == 0:
                print(num,'is an even Number')
```

Output:

```
2 is an even Number
4 is an even Number
6 is an even Number
8 is an even Number
10 is an even Number
```

The Range() Function

- To loop through a set of code a specified number of times range() function is used.
- The range() function returns a sequence of numbers **starting from 0 by default**, and **increments by 1 (by default),** and ends at a specified number (excluding stop value),
- Syntax of range() function is:

  range(start, stop, step)

  - ✓ **The start and step parameters are optional.**
  - ✓ If start value is not specified, by default the list starts from 0.
  - ✓ If step is also not specified, by default the value increases by 1 in each iteration.
  - ✓ **All parameters of range() function must be integers.**
  - ✓ The step parameter can be a positive or a negative integer excluding zero.

Example:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
#default step value is 1

>>> list(range(2, 10))
[2, 3, 4, 5, 6, 7, 8, 9]
#step value is 5

>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
#step value is -1. Hence, decreasing sequence is generated
```

```
>>> list (range (0, -9, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8]
```

Example:

```
n = 4
for i in range(0, n):
    print(i)
```

Output :

```
0
1
2
3
```

Example:

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
  print(x)
```

Output:

```
2
3
4
5
```

Example:

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

```
for x in range(2, 30, 3):
  print(x)
```

Output:

```
2
5
8
11
14
17
20
23
26
29
```

```
#Program to print the multiples of 10 for numbers in a given range.
for num in range(5):
        if num > 0:
                print(num * 10)
```

Output:
```
10
20
30
40
```

#Program
```
names = ['Kelly', 'Jessa', 'Emma']
# outer loop
for name in names:
    # inner while loop
    count = 0
    while count < 5:
        print(name, end=' ')
        # increment counter
        count = count + 1
    print()
```

Output:
```
Kelly Kelly Kelly Kelly Kelly
Jessa Jessa Jessa Jessa Jessa
Emma Emma Emma Emma Emma
```

Note:

- For loops can iterate over any iterable object (example: List, Set, Dictionary, Tuple or String).
- **Using else statement with for loops:** We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.

```
# Python program to illustrate combining else with for
 list = ["hello", "python", "data"]
for index in range(len(list)):
    print list[index]
else:
    print "Inside Else Block"
```

Output:
```
hello
python
data
Inside Else Block
```
Example:
```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

1) Nested Loops

- A loop may contain another loop inside it.
- A loop inside another loop is called a nested loop.
- Any type of loop (for/while) may be nested within another loop (for/while).

Example:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Output:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

```
#Program to find prime numbers between 2 to 50 using nested for loops.
num = 2
for i in range(2, 50):
        j= 2
        while ( j <= (i/2)):
                if (i % j == 0): #factor found
                        break #break out of while loop
                j += 1
        if ( j > i/j) : #no factor found

                print( i, "is a prime number")
print("Bye Bye!!")
```

Output:

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
```

```
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
Bye Bye!!
```

#Write a program to calculate the factorial of a given number.

```
#block to calculate the factorial of a given number
num = int(input("Enter a number: "))
fact = 1
# check if the number is negative, positive or zero
if num < 0:
        print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
        print("The factorial of 0 is 1")
else:
        for i in range(1, num + 1):
                fact = fact * i
        print("factorial of ", num, " is ", fact)
```

Output:

```
Enter a number: 5
Factorial of 5 is 120
```

Loop Control Statements

1) Break Statement

The break statement alters the normal flow of execution as it terminates the current loop and resumes execution of the statement following that loop.

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output =

```
1
2
3
```

```
#Program to demonstrate the use of break statement in loop
num = 0
for num in range(10):
        num = num + 1
        if num == 8:
                break
        print('Num has value ' + str(num))
print('Encountered break!! Out of loop')
```

Output:

```
Num has value 1
Num has value 2
Num has value 3
Num has value 4
Num has value 5
Num has value 6
Num has value 7
Encountered break!! Out of loop
```

2) Continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:
- When a continue statement is encountered, the control skips the execution of remaining statements inside the body of the loop for the current iteration and jumps to the beginning of the loop for the next iteration.
- If the loop's condition is still true, the loop is entered again, else the control is transferred to the statement immediately following the loop.

Example:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Output =

```
1
2
4
5
6
```