Flow control:

- Flow controls are used to understand the flow of statements execution in a program
- In any programming language, statements will be executed mainly in three ways,
    - **Sequential execution:** Statements execute from top to bottom, means one by one sequentially.By using sequential statement, we can develop only simple programs.
    - **Conditional execution:** Based on conditions, statements used to execute.Conditional statements are useful to develop better and complex programs.
    - **Looping execution:** Based on conditions, statements used to execute randomly and repeatedly. Looping execution is useful to develop better and complex programs.

Sequential statements:

- Sequential statements execute from top to bottom, means one by one sequentially

```
demo.py

print("one")
print("two")
print("three")

output:

one
two
three
```

Conditional or Decision-making statements:

- if - valid combination
- if else - valid combination
- if elif else - valid combination(recommended)
- if elif elif else - valid combination(recommended)
- if elif - valid combination(not recommended)

Looping:

- while loop
- for loop

Others:

- break
- continue

- return
- pass

If Statement:

- **syntax:**

```
                if condition:
                    if block statements

                out of if block statements
```

- if statement contains an expression/condition.
- As per the syntax colon (:) is mandatory otherwise it throws syntax error.
- After if statement we need to follow indentation otherwise it throws IndentationError
- Condition gives the result as a bool type, means either True or False.
- If the condition result is True, then if block statements will be executed
- If the condition result is False, then if block statements won't execute.
- If you want to do either one thing or nothing at all, then you should go for if statement.

```
demo.py

x=1
y=1

print("x==y value is: ", (x==y))

if x == y:
    print("if block statements executed")

output

x==y value is: True

if block statements executed
```

if else statement:

- **Syntax:**

```
                if condition:
                    if block statements1
                else:
                    else block statements2
```

- if statement contains an expression/condition.
- As per the syntax colon (:) is mandatory otherwise it throws syntax error
- After if statement and else statements we need to follow indentation otherwise it throws IndentationError
- Condition gives the result as bool type, means either True or False
- If the condition result is True, then if block statements will be executed
- If the condition result is False, then else block statements will be executed.
- If you want to do either one thing or other thing, then you should go for if else statement

```
demo.py

x=1
y=1

print("x==y value is: ", (x==y))

if x == y:
     print("if block statements executed")
else:
    print("else block statements executed")

output:

x==y value is: True
if block statements executed
```

if elif else statement:

- **syntax**

```
if condition1:
    if block statements
elif condition2:
    elif block1statements
elif condition3:
    elif block2statements
else:
    else block statements
```

- if statement contains an expression/condition.
- As per the syntax colon (:) is mandatory otherwise it throws error.
- After if statement, elif statement and else statement we need to follow indentation otherwise it throws IndentationError
- Condition gives the result as bool type, means either True or False
- If the condition result is True, then any matched if or elif block statements will execute
- If all if and elif conditions results are False, then else block statements will execute.

```
demo.py

print("Please enter the values from 0 to 4")

x=int(input("Enter a number: "))

if x==0:
    print("You entered:", x)
elif x==1:
    print("You entered:", x)
elif x==2:
    print("You entered:", x)
elif x==3:
    print("You entered:", x)
elif x==4:
    print("You entered:", x)
else:
    print("Beyond the range than specified")
```

```
output:

Enter a number: 1
You entered: 1
Enter a number: 100
Beyond the range than specified
```

Looping:

- If we want to execute a group of statements in multiple times, then we should go for looping kind of execution.
    - while loop
    - for loop

While loop:

1. If we want to execute a group of statements repeatedly until the condition reaches to False, then we should go for while loop
2. **Syntax**

```
Initialization
while condition:
    while block statements
    increment/decrement
```

3. while loop contains an expression/condition
4. As per the syntax colon (:) is mandatory otherwise it throws syntax error
5. After while loop we need to follow indentation otherwise it throws IndentationError.
6. Condition gives the result as bool type, means either True or False
7. If the condition result is True, then while loop executes till the condition reaches to False.
8. If the condition result is False, then while loop execution terminates.

while loop main parts :

- **Initialization section:**Initialization section is the first part in while loop.Here, before entering the condition section, initialization part is required.
- **Condition section:** With the initialized value, the next step is checking the condition. In python, while loop will check the condition at the beginning of the loop.
- Either increment or decrement by using arithmetic operator.
- In first iteration, if the condition returns True,then it will execute the statements which having inside while loop.
- In first iteration, if the condition returns False, then while loop terminates immediately.
- Then the loop execution goes to increment or decrement section.

Increment or decrement:

- Next step is, increment or decrement section.
- In python we need to use arithmetic operator inside while loop to increment or decrements the value.
- So, based on requirement, value will be either increment or decrement.
- In second iteration, if the condition returns True, then it will execute the statements which having inside while loop.

- In second iteration, if the condition returns False, then while loop terminates immediately.

```
demo.py

x=1

while x<=5:
     print(x)
     x+=1

print("End")

output:
1
2
3
4
5
End
```

for loop:

- Basically, for loop is used to get or iterate elements one by one from sequence like string, list, tuple, etc…
- While iterating elements from sequence we can perform operations on every element.
- **Syntax:**

```
for variable in sequence:
     statements
```

```
demo.py

obj = [10, 20, 30, 'Sachin']
for x in obj:
     print(x)

output:

 10
 20
 30
 Sachin
```

Nested loops:

- A loop inside loop is called as nested loop
- It is possible to write one loop inside another loop

```
demo.py

l=[[1, 2, 3], [4, 5, 6]]

for x in l:
        print(x)

  output:

  [1,2,3]
  [4,5,6]
```

```
demo.py

l=[[1, 2, 3], [4, 5, 6]]

for x in l:
     for y in x:
          print(y)

output:

1
2
3
4
5
6
```

break:

- The break statement can be used to terminate the execution based on some condition
- If you are searching element from list object, if unable to find element then for loop won't return anything, so at that time else suit help us to tell not found the element.
- The break statement can be used inside the loops to break the execution based on some condition.
- Generally, break statement is used to terminate the for loop and while loop.

```
demo.py

x=1
while x<=10:
     print("x=", x)
     x+=1

print("out of loop")

output

x= 1
x= 2
x= 3
x= 4
x= 5
x= 6
x= 7
x= 8
```

```
x= 9
x= 10
out of loop
```

```
demo.py

x=1
while x<=10:
     print("x=", x)
     x+=1
     if x == 5:
         break
print("out of the loop")

output

x= 1
x= 2
x= 3
```

continue statement:

- We can use continue statement to skip current iteration and continue next iteration

```
demo.py

cart=[10,20,500,700,50,60]

for item in cart:
     if item==500:
         continue
     print(item)
```

pass statement:

- We can use pass statement when we need a statement syntactically, but we do not want to do any operation.

```
num = [10, 20, 30,400, 500, 600]
for i in num:
    if i<100:
       pass
    else:
       print(i)
```