PYTHON STATEMENTS

Input/Output **(I/O)** statements

- Python provides **two built-in methods** to read the data from the keyboard. These methods are given below.

    1. **input(prompt)**

    2. **raw_input(prompt)** [used in Python's older version]

```
name = input("Enter your name: ")  # String Input
age = int(input("Enter your age: ")) # Integer Input
marks = float(input("Enter your marks: ")) # Float Input

print("The name is:", name)
print("The age is:", age)
print("The marks is:", marks)
```

```
Enter your name: Johnson
Enter your age: 21
Enter your marks: 89
The name is: Johnson
The age is 21
The marks is: 89.0
```

```
name = raw_input("Enter your name : ")
```

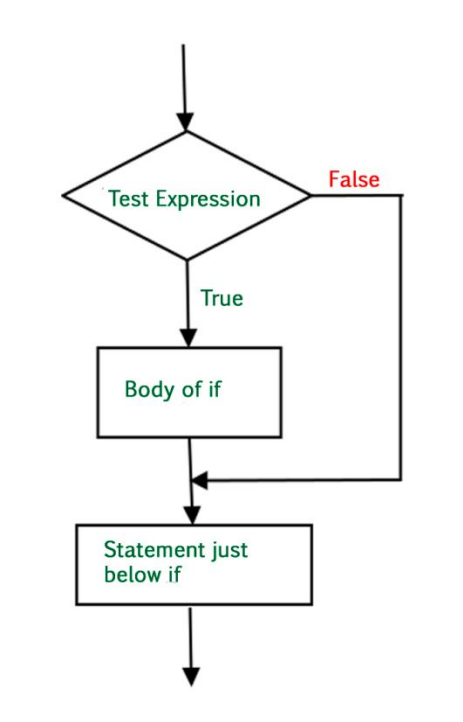
Control Statements
(Decision Making statement and Looping statement)

- Decision making is the most important aspect of almost all the programming languages.
- As the name implies, decision making allows us to run a particular block of code for a particular decision.
- Here, the decisions are made on the validity of the particular conditions.
- Condition checking is the backbone of decision making.
- Some decision making statements are:
  - if statement
  - If-else statements
  - Nested if Statement
  - if-elif-else ladder

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation.

if statement

- if statement is the most simple decision-making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.



Syntax:

```
if condition:
   # Statements to execute if
   # condition is true
    statement1
statement2
```

I am Not in if

Sample Program using if statement

```
i = 10

if (i > 15):
    print(“Welcome”)
print("I am Not in if")
```

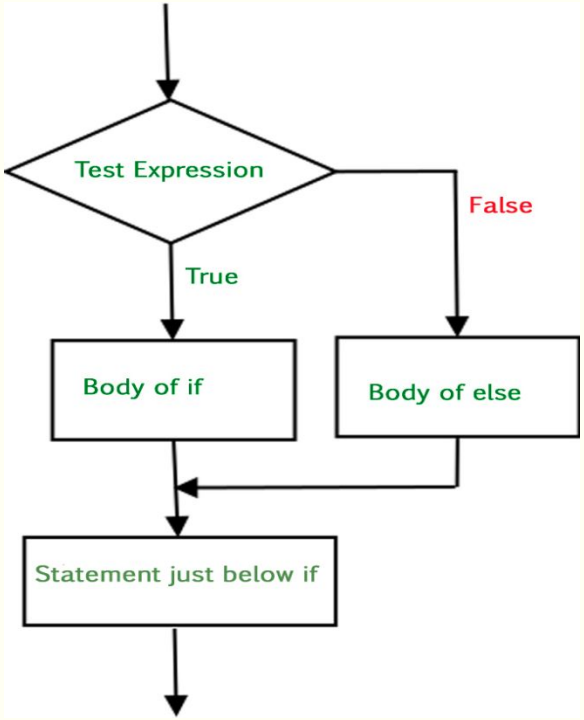
I am Not in if

```
num = int(input("enter the number?"))
if (num % 2)== 0:
    print("Number is even")
```

```
enter the number?4
Number is even
enter the number?5
```

If-else statements

- The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked.
- If the condition provided in the if statement is **false**, then the **else** statement will be executed.



Syntax:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Sample Program using if-else statement

```
i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")
```

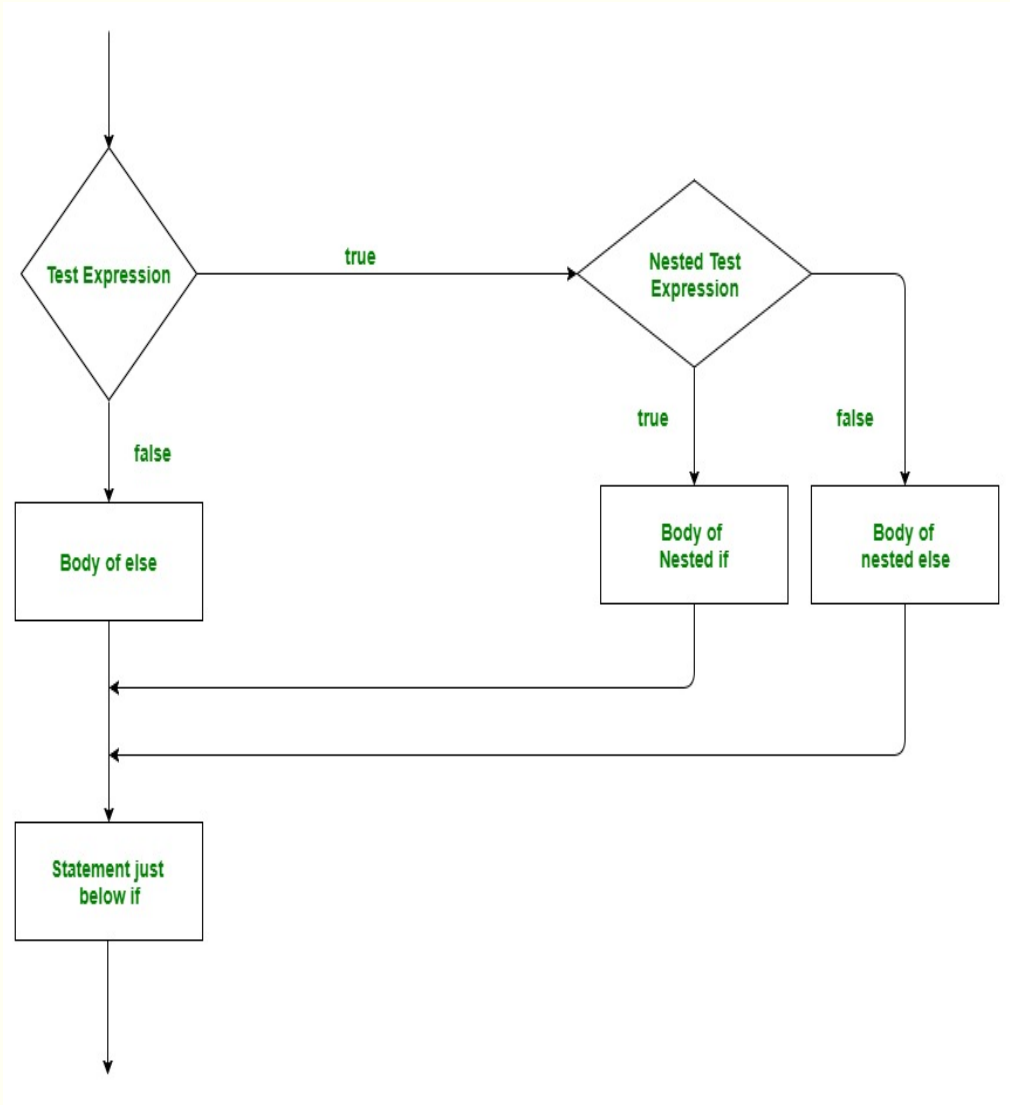
```
enter the number?10
Number is even
enter the number?7
Number is odd
```

Nested if statements

- **Nested if** statements enable us to use if ? else statement inside an outer if statement.

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```



Sample Program using Nested if statements

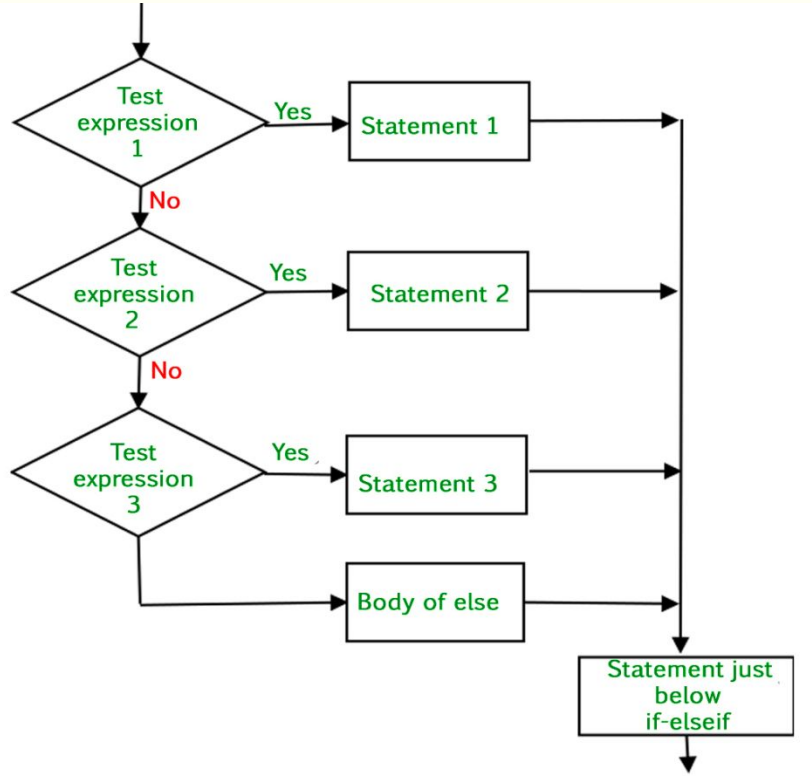
```
i = 10
if (i == 10):

    # First if statement
    if (i < 15):
        print("i is smaller than 15")

    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")
```

```
i = 10
if (i == 10):
{
  if (i < 15):
  { … }
  if (i < 12):
  { … }
  else:
  { … }
}
```

```
i is smaller than 15
i is smaller than 12 too
```

if-elif-else ladder

- The **elif statement** enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.



Syntax:

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Enter the number?15 number is not equal to 10, 50 or 100

Sample Program using if-elif-else ladder

```
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or
100");
```

```
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
   print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
   print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
   print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
   print("You scored grade C ...")
else:
   print("Sorry you are fail ?")
```

```
Enter the number?15
number is not equal to 10, 50 or 100
```

```
Enter the marks? 30
Sorry you are fail ?
Enter the marks? 85
You scored grade B + ...
```

Single Statement suites
(Short Hand statement)

- Whenever there is only a single statement to be executed inside the if block then shorthand if can be used. The statement can be put on the same line as the if statement.

Short Hand if statement

Syntax:
if condition: statement

```
if i < 15:print(i, "is less than 15")
```

Short Hand if-else statement

Syntax:
statement_when_True if condition else statement_when_False

```
i = 10
print(True) if i < 15 else print(False)
```

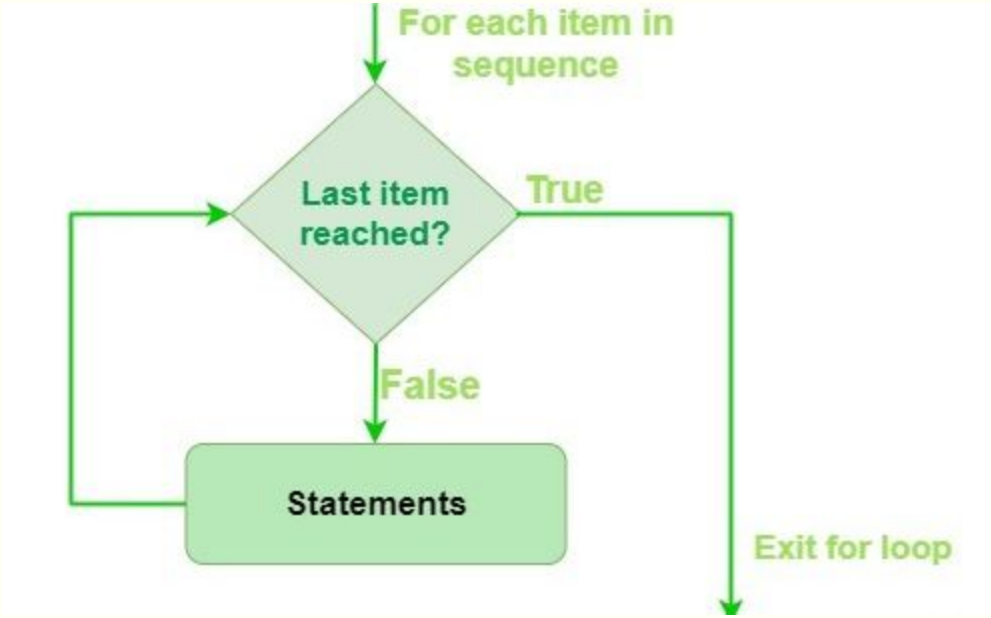
Looping statements

- The flow of the programs written in any programming language is sequential by default.
- Sometimes we may need to alter the flow of the program.
- The execution of a specific code may need to be repeated several numbers of times.
- For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.
- Loop Statements are:
  - For loop
  - While loop
  - Do-while loop (Unsupported)

for var in iterable: # statements

For Loops

- The **for loop in Python** is used to iterate the statements or a part of the program several times.
- It is frequently used to traverse the data structures like list, tuple, or dictionary.




```
for var in iterable:
    # statements
```

For loop Using Sequence

Example-1: Iterating string using for loop

```
str = "Python"
for i in str:
    print(i)
```

```
P
y
t
h
o
n
```

Example- 2: Program to print the table of the given number .

```
list = [1,2,3,4,5]
n = 5
for i in list:
    c = n*i
```

```
5
10
15
20
25
```

For loop Using range() function

- The **range()** function is used to generate the sequence of the numbers.
- If we pass the range(10), it will generate the numbers from 0 to 9.
- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1.
  The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Syntax:
range(start, stop, step size)

```
n = int(input("Enter the number "))
for i in range(2,n,2):
    print(i)
```

```
Enter the number 10
2
4
6
8
```

Nested for loop in python

- Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop.

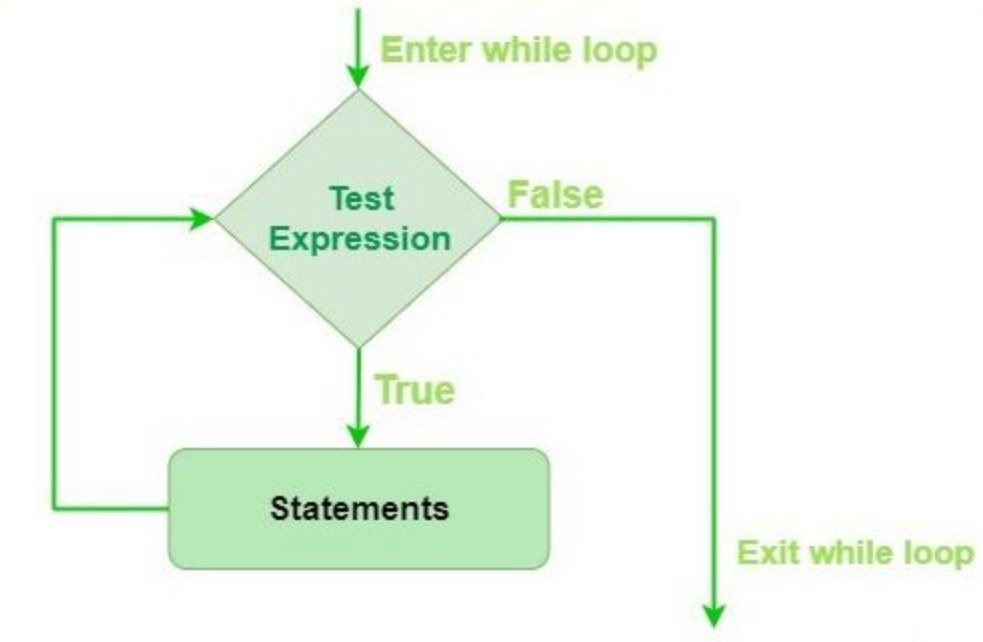
```
for iterating_var1 in sequence:  #outer loop
    for iterating_var2 in sequence:  #inner loop
        #block of statements
#Other statements
```

```
#Program to number pyramid.
rows = int(input("Enter the rows"))
for i in range(1,rows+1):
    for j in range(i):
        print(i,end = '')
    print()
```

```
1
22
333
4444
55555
```

While Loops

- The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a **pre-tested loop.**
- It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.



Syntax:
while expression:
    statements

Single statement while block

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello")
```

```
Hello
Hello
Hello
```

```
count = 0
while (count < 3): count += 1; print("Hello ")
```

Infinite while loop

```
while (1):
    print("Hi! ")
```

```
Hi!
Hi!
Hi!
Hi!
```

Loop executes indefinitely until keyboard interrupt.
Use ctrl+c in Python IDLE to end the loop

Sentinel Controlled Statement

- A sentinel value is a value that is used to terminate a loop whenever a user enters it, generally, the sentinel value is -1.

```
a = int(input('Enter a number (-1 to quit): '))

while a != -1:
    a = int(input('Enter a number (-1 to quit): '))
```

```
Enter a number (-1 to quit): 6
Enter a number (-1 to quit): 7
Enter a number (-1 to quit): 8
Enter a number (-1 to quit): 9
Enter a number (-1 to quit): 10
Enter a number (-1 to quit): -1
```

Do-While Loops

- Python does not have built-in functionality to explicitly create a do while loop like other languages.
- The do while loop is used to check condition after executing the statement. It is like while loop but it is executed at least once.

```
do {
    //statement
} while (condition);
```

```
i = 1

while True:
    print(i)
    i = i + 1
    if(i > 5):
        break
```

To create a do while loop in Python, you need to modify the while loop a bit in order to get similar behavior to a do while loop in other languages.

Loop Control Statements

- Python offers the following control statement to use within the looping statements.
- They are:
  - Break statement
  - Continue statement
  - Pass statement

0 1 2 3 4 5 6 7 8 9 came out of while loop

Python break statement

- The break is a keyword in python which is used to bring the program control out of the loop.
- The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.
- In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

```
p
y
t
h
```

Python continue Statement

- The continue statement in Python is used to bring the program control to the beginning of the loop.
- The continue statement skips the remaining lines of code inside the loop and start with the next iteration.
- It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

```
i = 0
while(i < 10):

   i = i+1
   if(i == 5):
      continue
   print(i)
```

```
1
2
3
4
6
7
8
9
10
```

Pass Statement

- The pass statement is a null operation since nothing happens when it is executed.
- It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.
- The Python pass statement to write empty loops.
- Pass is also used for empty control statements, functions, and classes.

```
for i in [1,2,3,4,5]:
    if(i==4):
        pass
        print("This is pass block",i)
    print(i)
```

```
1
2
3
This is pass block 4
4
5
```