CHAPTER 2 [10M]

INTRODUCTION TO SYNTAX OF PYTHON PROGRAMMING

TOPICS TO LEARN :-

2.1 Basic Operators: Arithmetic, Comparison/Relational, Assignment,Logical, Bitwise, Membership, Identity operators, Python Operator Precedence

2.2 Control Flow

2.3 Conditional Statements (if, if ... else, nested if)

2.4 Looping in python (while loop, for loop,nested loops)

2.5 loop manipulation using continue, pass, break, else.

Operands are special symbols that are used in computations. For example +,-, * and / are used for performing arithmetic operations. The values that operator uses for performing computation are called operands.

Ex.
Consider the expression 4+5-=9. Here, 4 and 5 are called operands and + is called operator.

The operator and operand when combined to perform a certain operation, it becomes an expression.
For example, in expression x + y, x and y are the variables (operands) and the plus (+) sign is the operator that specifies the type of operation performed on the variables.

REMEMBER

In Python, statements in a program are executed one after another in the order in which they are written. This is called sequential execution of the program.

2.1 BASIC OPERATOR :

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

1. Arithmetic Operators :

These operators are used for performing arithmetic operations.

OPERATOR	DESCRIPTION
+	Addition is used operator performing addition of two numbers.
-	Subtraction operator is used for performing subtraction.
*	Multiplication operator is used for performing multiplication.
/	Division operator is used for performing division of two numbers.
%	Mod operator returns the remainder value.
**	This is an exponentiation operator(power).
//	This is floor division operator. In this operation the result is the quotient in which the digits after the decimal point are removed.

Ex.

```
print(100 + 20)
print(100 - 20)
print(10 * 10)
print(10 / 2)
print(10 % 2)
print(2 ** 3)
print(10 // 3)
```

2. Comparison / Relational Operators :

The operators compare the values and establish the relationship among them.

OPERATOR	DESCRIPTION
==	If two values are equal then condition becomes true
!=	If two operands are not equal then the condition becomes true
<	This is less than operator. If left operand is less than the right operator then the return value is true
>	This is greater than operator. If left operand is greater than the right operator then the return value is true
<=	This is less than equal to operator. If left operand is less than the right operator or equal to the right operator then the return value is true
>=	This is greater than equal to operator. If left operand is less than the right operator or equal to the right operator then the return value is true

Ex.

```
x = 5
y = 3
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
```

```
print(x <= y)
```

3. Assignment Operators :

The assignment operator is used to assign the values to variables.

OPERATOR	DESCRIPTION
=	This is an operator using which values is assigned to a variable
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign

Ex.

```
x = 5
print(x)
x += 3
print(x)
y = 10
y -= 3
print(y)
z = 15
z *= 3
print(z)
a = 30
a /= 3
print(a)
```

4. Logical Operators :

Logical operators are used to combine conditional statements.

OPERATOR	DESCRIPTION
and	If both the operands are true then the entire expression is true.
or	If either first or second operand is true.
not	If the operand is false then the entire expression is true.

Ex.

```
x = 5

print(x > 3 and x < 10)

print(x > 3 or x < 10)

print(not(x > 3 and x < 10))
```

5. Bitwise Operators :

Bitwise operators work on the bits of the given value. These bits are binary numbers i.e. 0 or 1

OPERATOR	DESCRIPTION
&	This AND operation operation performs | between operands. Operator copies a bit, to the result, exists in both operands
|	This operation if it performs OR operation between operands. It copies a bit, if it exists in either operand.
~	This operations performs XOR operations between operands. It copies the bit, if it is set in one operand but not both.
^	It is unary operator and has the effect of 'flipping' bits i.e. opposite the bits of operand.
<<	The left operand's value is moved left by the number of bits specified by the left operand.
>>	The right operand's value is moved right by the number of bits specified by the right operand.

Ex.

```
print(6 & 3)
print(6 | 3)
print(6 ~ 3)
print(6 ^ 3)
print(6 << 3)
print(6 >> 3)
```

6. Membership Operators :

Membership operators are used to find out whether a value is a member of a sequence such as string or list.

OPERATOR	DESCRIPTION
in	It returns True if a sequence with specified value is present in the object.
not in	It returns true if a sequence with specified value is not present in the object.

Ex.

```
x = ["apple", "banana"]

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list

print("pineapple" not in x)

# returns True because a sequence with the value "pineapple" is not in the list
```

7. Identity Operators :

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

OPERATOR	DESCRIPTION
is	Returns True if both variables are the same object
is not	Returns True if both variables are not the same object

Ex.

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)
# returns True because z is the same object as x

print(x is y)
# returns False because x is not the same object as y, even if they have the same content

print(x is not z)
# returns False because z is the same object as x

print(x is not y)
# returns True because x is not the same object as y, even if they have the same content
```

Python Operator Precedence :

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.

The acronym **PEMDAS** is a useful way to remember the order of operations:

1. **P**: Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 1 * (10-5) is 5
2. **E**: Exponentiation has the next highest precedence, so 2**3 is 8.
3. **MDAS**: Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence. So 2+3*4 yields 14 rather than 20.
4. Operators with the same precedence are evaluated from left to right. So in the expression 3-2+1 will result 2. As subtraction is performed first and then addition will be performed.

Ex.

```
print((6 + 3) - (6 + 3))
```

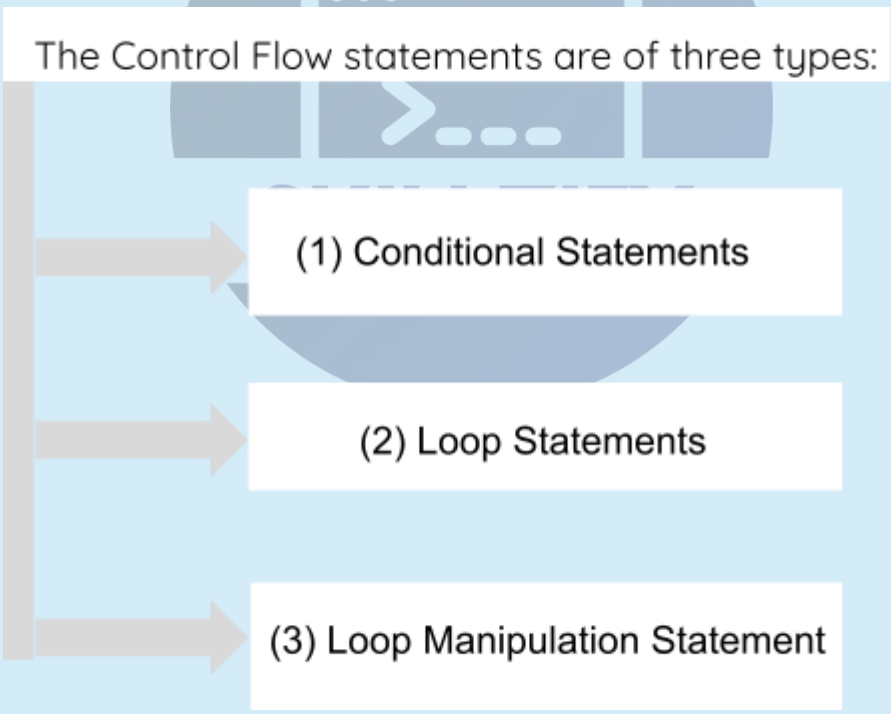
REMEMBER

When an expression has two or more operators, we need to identity the correct sequence to evaluate these operators. This is because the final answer changes depending on the sequence thus chosen.

2.2 CONTROL FLOW :

The program control flow is the order is the order in which program code executes. Python programming provides a control structure which transfers the control from one part of program to some other part of program.

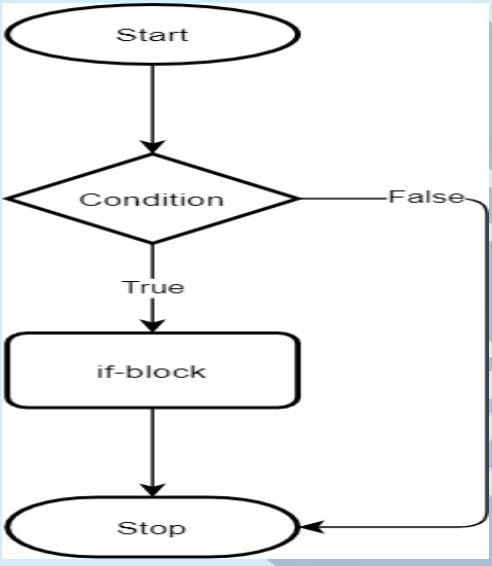
The Control Flow is refer to statement sequencing in a program to get desire result.



2.3 CONDITIONAL STATEMENTS

1.IF STATEMENT -

The if statement is used to test particular condition. If the condition is true then it executes the block of statements which is called as if block.



SYNTAX -

```
if condition :
//statement
```

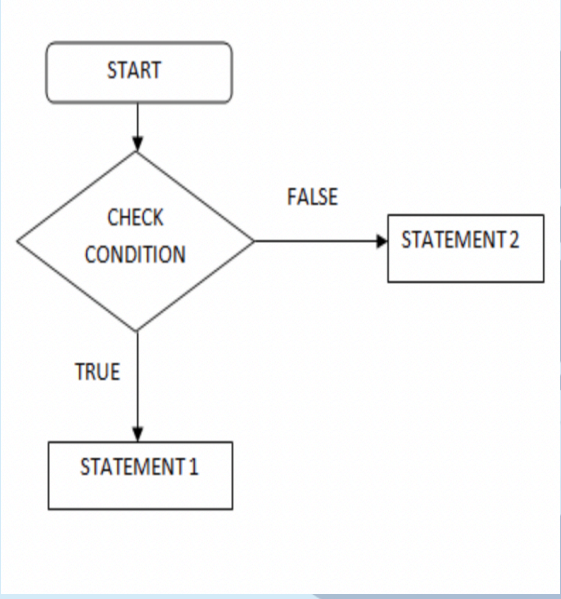
Ex.

```
a = 33
b = 200

if b > a:
  print("b is greater than a")
```

2.IF-ELSE STATEMENT -

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.



SYNTAX -

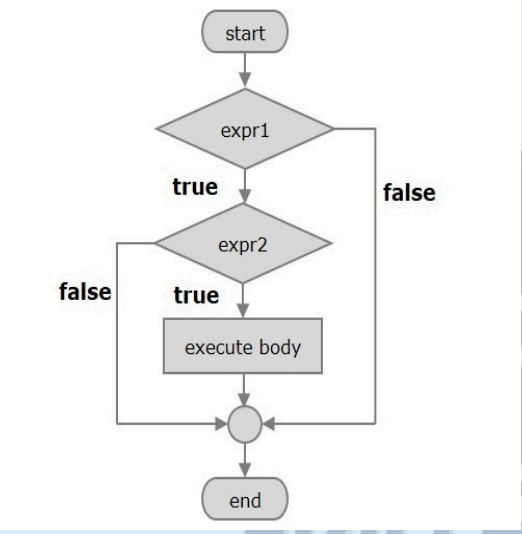
```
if condition :
//statement 1
else
//statement 2
```

Ex.

```
print("Enter value of n")
n=int(input())
if n%2= =0:
print('Even Number')
else:
print("Odd Number")
```

3.NESTED IF STATEMENT -

When one if condition is present inside another if then it is called nested if conditions. Any number of these statements can be nested inside one another.



SYNTAX -

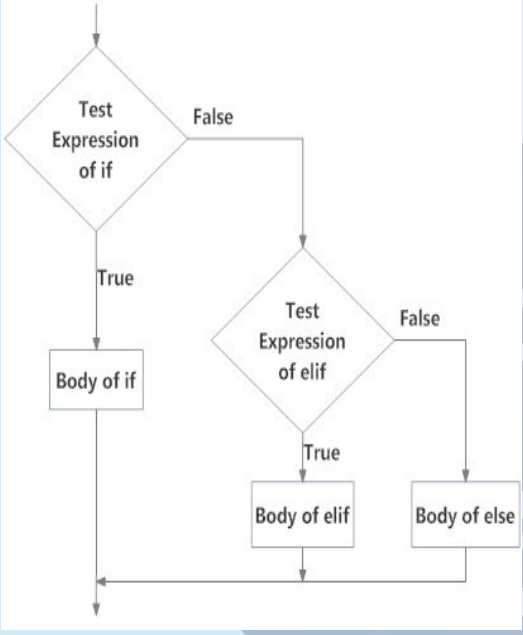
```
if condition 1 :
  if condition 2 :
   //statement 1
  if condition 3 :
   //statement 2
else :
//statement 3
```


Ex.

```
print("Enter value of a")
a=int(input())
 print("Enter value of b")
 b=int(input())
 if a==b:
 print("Both the numbers are equal')
 else:
 else:
 ifa<b:
 print("a is less than b")
 if a>b:
 print("a is greater than b")
```

4. IF-ELIF-ELSE STATEMENT -

Sometimes there are more than two possibilities. These possibilities can be expressed using chained conditions.



Syntax -

```
if(condition 1) :
  //statements
elif(condition 2) :
  //statements
  .
  .
  .
elif(condition n) :
  //statements
    else:
  //statements
```

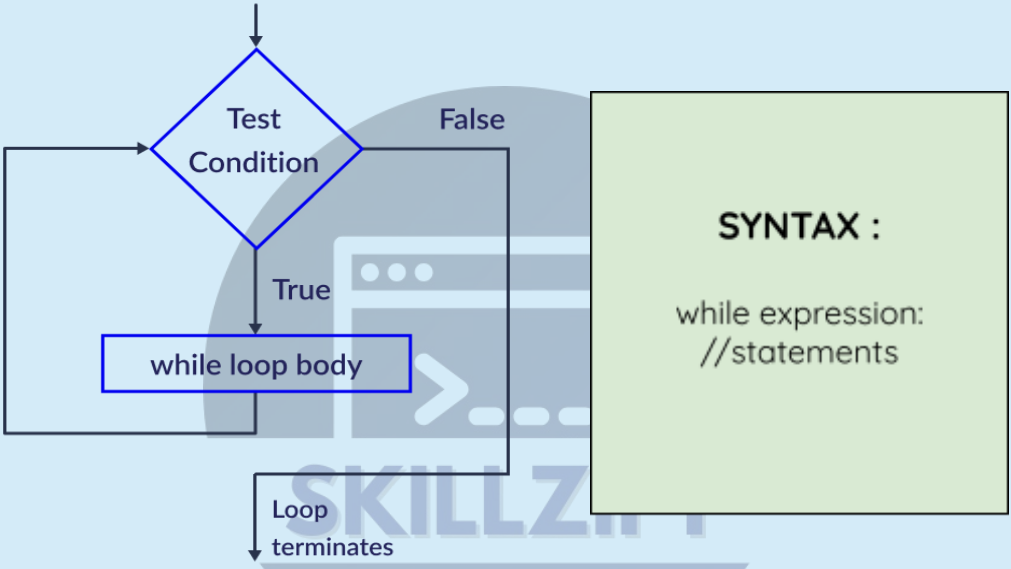
Ex.

```
i=20
if (i==10):
   print("i is 10")
elif (i==20):
   print("i is 20")
elif (i==30):
   print("i is 30")
else:
    print("i is null")
```

2.4 LOOPING IN PYTHON

1. WHILE LOOP -

It is the most basic looping treatment in Python Programming it execute. A sequence of statement repeatedly as long as condition is true



Ex.

```
i = 1
while i < 6
print(i)
i += 1
```

NOTE ME

WHILE . . . ELSE :

There is another version of while statement in which else is used.

Syntax :

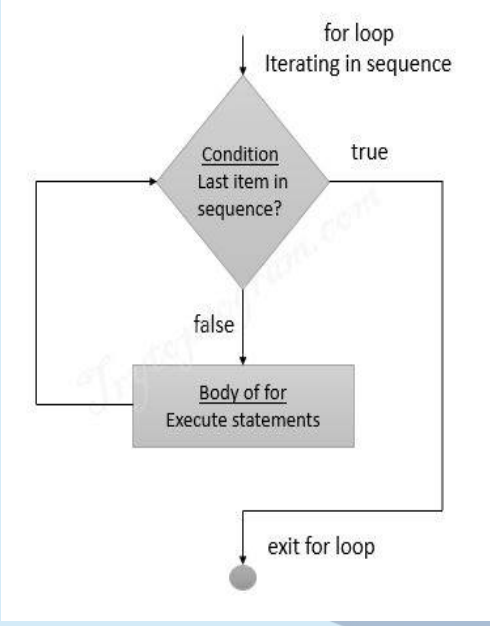
```
while test_condition:
body of while
else:
statement
```

Example :

```
while i<=10:
i=i+1
else:
print("Invalid value of i")
```

2. FOR LOOP -

The Python for loop it red through a sequence of object that is it rates through each value in a sequence where the sequence of object holes multiple items of data stored one after another



SYNTAX -

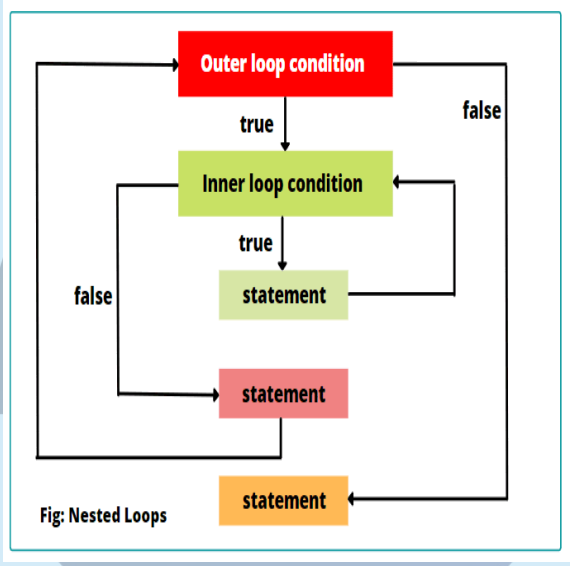
for var in sequence :
//statements

Ex.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
print(x)
```

3. NESTED LOOP -

Nested loops are the loop structure in which one loop is present inside the other loop. There can be nested for loops. That means one for loop is present inside the other for loop.



Ex.

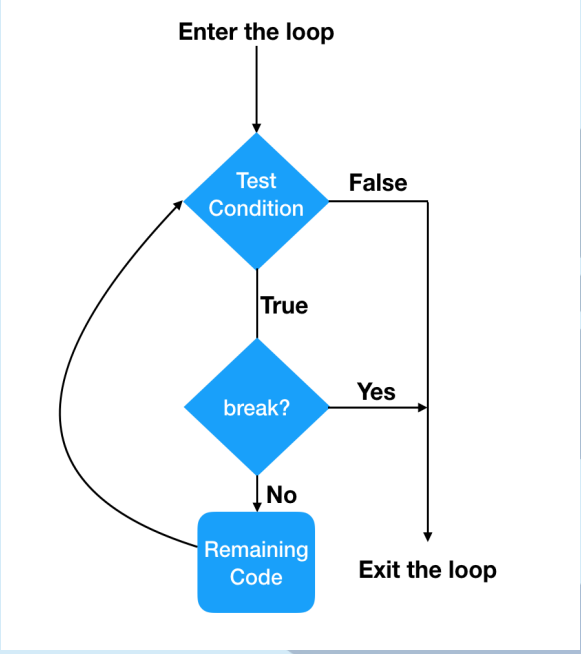
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

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

2.5 LOOP MANIPULATION -

1. BREAK -

The break statement is used to transfer the control to the end of the loop.



SYNTAX -

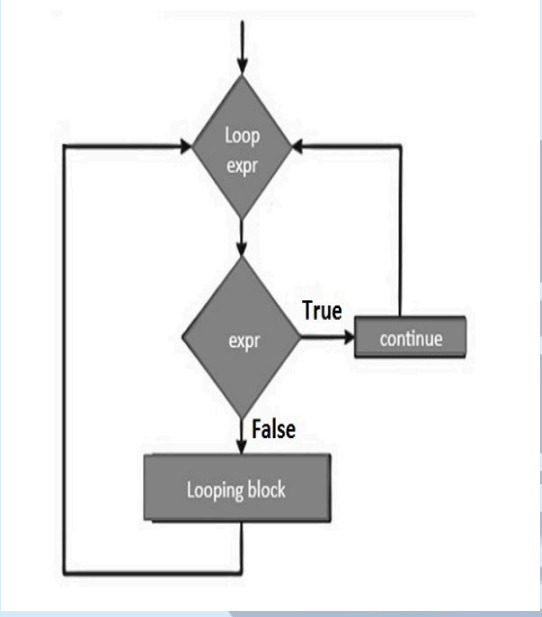
break

Ex.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
print(x)
if x == "banana":
break
```

2. CONTINUE -

The continue statement is used to skip some statements inside the loop.



SYNTAX -

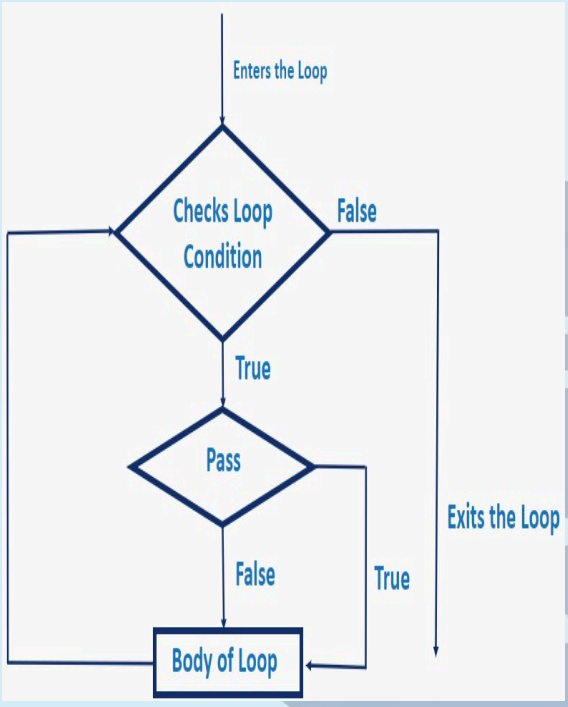
continue

Ex.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
 if x == "banana":
   continue
 print(x)
```

3. PASS -

The pass statement is used when we do not want to execute any statement. Thus the desired statements can be bypassed.



Syntax -
Pass

Ex.

```
for in range(1,5):
if i==3:
pass
print("Reached at pass statement")
print("The current number is ",i)
```

4. ELSE -

i) Else statement with for loop :

Else keyword in a for loop specifies a block of code to be executed when loop is finished

SYNTAX :

```
for variable in sequence:
Body of for loop
else:
Statement
```

Ex.

```
for variable in sequence:
 Body of for loop
 else:
 Statement
```

ii) Else statement with while loop :

Else keyword in a while loop specifies a block of code to be executed until the condition is true.

SYNTAX :

```
while condition :
    statement
else:
    statement
```

Ex.

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```