UNIT NO. 2 Marks: 10

PYTHON OPERATORS & CONTROL FLOW STATEMENT

Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a specific programming language. Python provides a variety of operators, which are described as follows.

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

1] Arithmetic Operators

- Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.
- Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10

/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
***(Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to the second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

2] Comparison operator

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

3] Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

4] Bitwise Operators

The bitwise operators perform bit by bit operation on the values of the two operands. Consider the following example.

if a = 7

b = 6

then, binary (a) = 0111

binary (b) = 0110

hence, a & b = 0011

a | b = 0111

a ^ b = 0100

~ a = 1000

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different; otherwise, the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

5] Logical Operators:

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
And	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
Or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
Not	If an expression **a** is true, then not (a) will be false and vice versa.

6] Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
In	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

7] Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
Is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both sides do not point to the same object.

8] Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Conditional 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.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	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.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

1) The if statement

i) The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.
ii) The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

iii) Syntax :

```
if expression:
    statement
```

Example 1

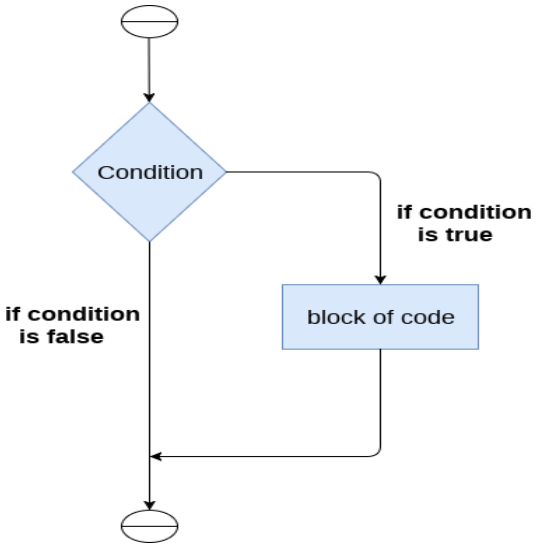
```
num = int(input("enter the number:"))
```

```
if num%2 == 0:
    print("Number is even")
```

Output:
enter the number:10
Number is even

Example 2 : Program to print the largest of the three numbers.

```
a = int(input("Enter a "));
b = int(input("Enter b "));
c = int(input("Enter c "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

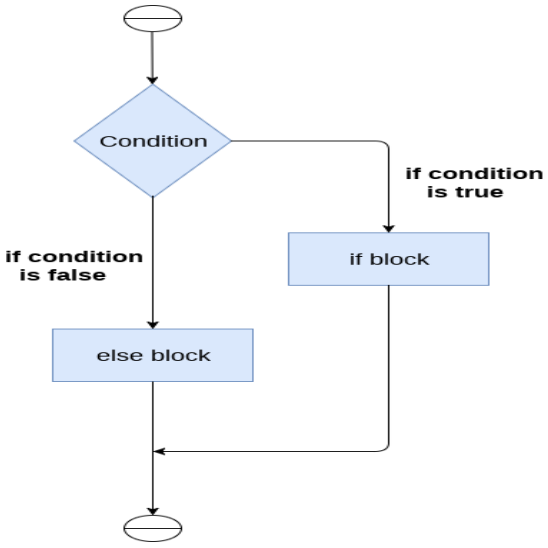


2) The if-else statement

i) 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.

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

ii) If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
   #block of statements
else:
   #another block of statements (else-block)
```

Ex 1 : Prog to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age "))

    if age>=18:
       print("You are eligible to vote !!");
    else:
       print("Sorry! you have to wait !!");
```

Ex 2: Program to check whether a number is even or not.

```
num = int(input("enter the number: "))

if num%2 == 0:
   print("Number is even...")
else:
   print("Number is odd...")
```

3) The elif statement

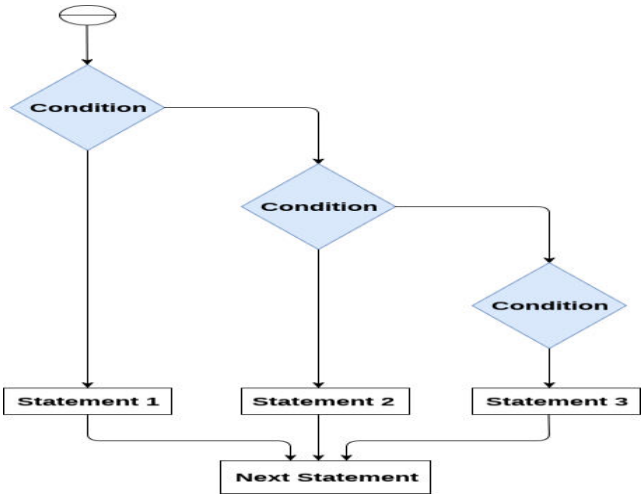
1) The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

2) We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

3) The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
   # block of statements

elif expression 2:
   # block of statements

elif expression 3:
   # block of statements

else:
   # block of statements
```



Example 1

```
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");
```

Example 2

```
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 ")
```

4) Nested if Statement:

i) Nested if statements enable us to use if ? else statement inside an outer if statement.

ii) There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct.

iii) In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

iv) Syntax

```
if expression1:
statement(s)
if expression2:
statement(s)
elif expression3:
statement(s)
elif expression4:
statement(s)
else:
statement(s)
else:
statement(s)
```

Example:

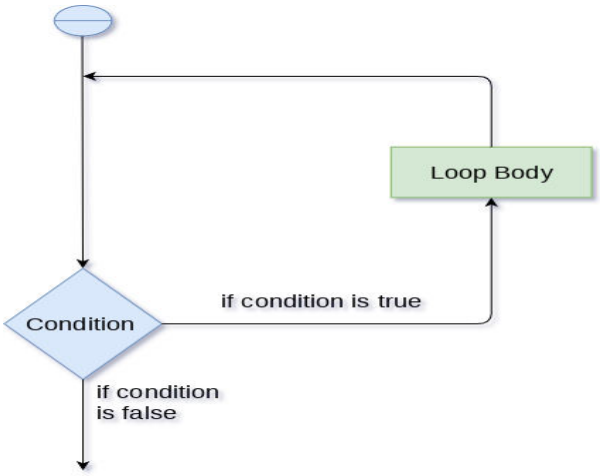
```
number = 7

if (number >= 0):

    if number == 0:
      print('Number is 0')


    else:
        print('Number is positive')


else:
    print('Number is negative')

# Output: Number is positive
```

Python Loops

1) 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.

2) The execution of a specific code may need to be repeated several numbers of times.

3) For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.



Why we use loops in python?

1) The looping simplifies the complex problems into the easy ones.

2) It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.

3) e.g. 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.

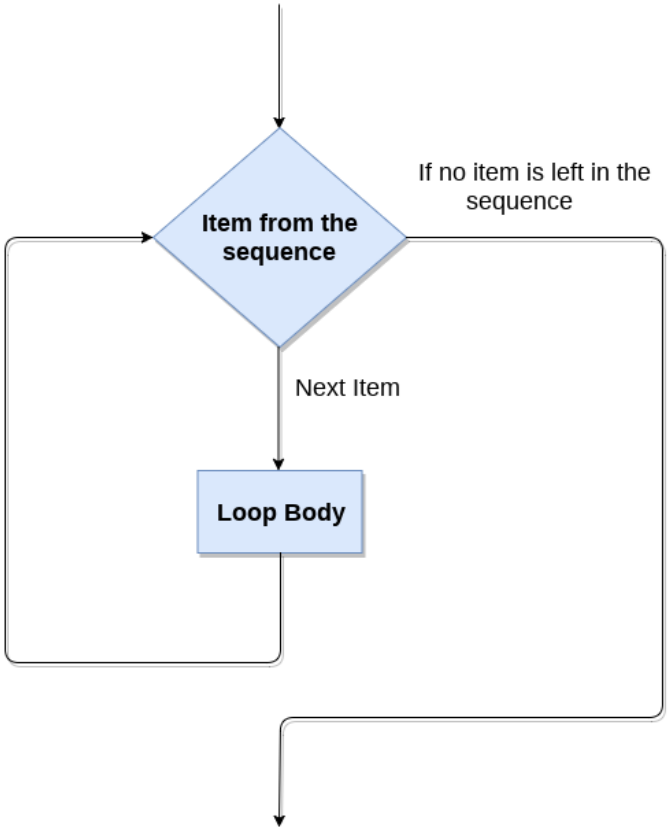
Advantages of loops

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

1) The for loop



For loop Using Sequence

Example-1: Iterating string using for loop

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

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

```
list = [1,2,3,4,5,6,7,8,9,10]
n = 5
for i in list:
   c = n*i
   print(c)
```

For loop Using range() function

The 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 syntax of the range() function is given below.

Syntax:

1. range(start,stop,step size)
   - 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.

Consider the following examples:

Example-1: Program to print numbers in sequence.

```
for i in range(10):
   print(i,end = ' ')
```

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

```
n = int(input("Enter the number "))
for i in range(1,11):
    c = n*i
    print(n,"*",i,"=",c)
```

We can also use the **range()** function with sequence of numbers. The **len()** function is combined with range() function which iterate through a sequence using indexing. Consider the following example.

```
list = ['Pratyush','Neer','Jay','Ram']
for i in range(len(list)):
    print("Hello",list[i])
```

2) 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. The syntax is given below.

Syntax

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

Example- 1: Nested for loop

```
    rows = int(input("Enter the rows:"))        # User input for number of rows
for i in range(0,rows+1):                       # Outer loop will print number of rows
    for j in range(i):
     print("*",end = '')                        # Inner loop will print number of Astrisk
     print()
```

Example-2: Program to number pyramid.

```
rows = int(input("Enter the rows"))
for i in range(0,rows+1):
    for j in range(i):
        print(i,end = '')
    print()
```

Using else statement with for loop

Unlike other languages like C, C++, or Java, Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example 1

```
for i in range(0,5):
    print(i)
else:
    print("for loop completely exhausted, since there is no break.")
```

Example 2

```
for i in range(0,5):
    print(i)
    break;
else:
print("for loop is exhausted");
print("The loop is broken due to break statement...came out of the loop")
```

In the above example, the loop is broken due to the break statement; therefore, the else statement will not be executed. The statement present immediate next to else block will be executed.

Python While loop

1) 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.

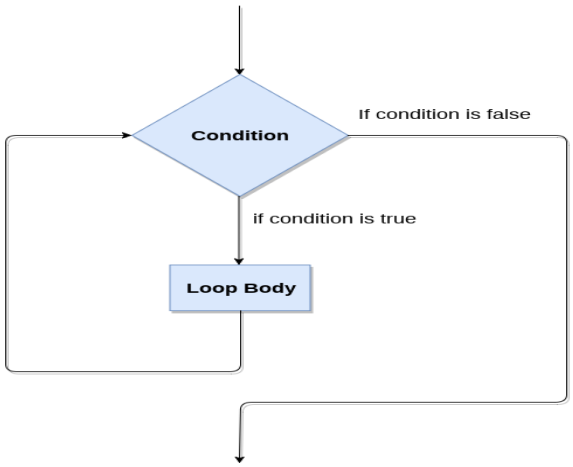
2) It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

```
while expression:
    statements
```

Here, the statements can be a single statement or a group of statements. The expression should be any valid Python expression resulting in true or false. The true is any non-zero value & false is 0.

While loop Flowchart



Loop Control Statements

We can change the normal sequence of **while** loop's execution using the loop control statement. When the while loop's execution is completed, all automatic objects defined in that scope are demolished. Python offers the following control statement to use within the while loop.

1] Continue Statement –

i) When the continue statement is encountered, the control transfer to the beginning of the loop.

ii) The continue keyword is **used to end the current iteration in a for loop (or a while loop), and continues to the next iteration**.

Example:

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

2] Break Statement –

1) When the break statement is encountered, it brings control out of the loop.
2) Break' in Python is **a loop control statement**. It is used to control the sequence of the loop.

3) Suppose you want to terminate a loop and skip to the next code after the loop; break will help you do that.
4) A typical scenario of using the Break in Python is when an external condition triggers the loop's termination.

Example:

```
 i = 1

while i <= 10:
   print('6 * ',(i), '=',6 * i)

   if i >= 5:
      break

   i = i + 1
```

3] Pass Statement –

I) The pass statement is used to declare the empty loop. It is also used to define empty class, function, and control statement.

ii) The pass statement is **used as a placeholder for future code**.

iii) When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

iv) Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

Let's understand the following example.

Example - with pass statement

```
n = 10
# use pass inside if statement
if n > 10:
   pass
print('Hello')
#Without pass statement

n = 10

if n > 10:
   # write code later

print('Hello')
```

Example-1: Program to print 1 to 10 using while loop

```
i=1
#The while loop will iterate until condition becomes false.
While(i<=10):
    print(i)
    i=i+1
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Infinite while loop

If the condition is given in the while loop never becomes false, then the while loop will never terminate, and it turns into the **infinite while loop.**

Any **non-zero** value in the while loop indicates an **always-true** condition, whereas zero indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

Ex 1

```
while (1):
    print("Hi! we are inside the infinite while loop")
```

Ex 2

```
    var = 1
while(var != 2):
   i = int(input("Enter the number:"))
   print("Entered value is %d"%(i))
```

Using else with while loop

Python allows us to use the else statement with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed, and the statement present after else block will be executed. The else statement is optional to use with the while loop. Consider the following example.

Ex 1

```
i=1
while(i<=5):
    print(i)
    i=i+1
else:
    print("The while loop exhausted")
```

Ex 2

```
i=1
while(i<=5):
    print(i)
    i=i+1
    if(i==3):
        break
else:
    print("The while loop exhausted")
```

In the above code, when the break statement encountered, then while loop stopped its execution and skipped the else statement.

Ex-3 Program to print Fibonacci numbers to given limit

```
terms = int(input("Enter the terms "))
# first two intial terms
a = 0
b = 1
count = 0

# check if the number of terms is Zero or negative
if (terms <= 0):
```

```
    print("Please enter a valid integer")
elif (terms == 1):
    print("Fibonacci sequence upto",limit,":")
    print(a)
else:
    print("Fibonacci sequence:")
    while (count < terms) :
        print(a, end = ' ')
        c = a + b
        # updateing values
        a = b
        b = c
        count += 1
```