2...

Python Operators and Control Flow Statements

Chapter Outcomes...

- Write simple Python program for the given arithmetic expressions.
- Use different types of operators for writing the arithmetic expressions.
- Write a 'Python' program using decision making structure for two-way branching to solve the given problem.
- Write a 'Python' program using decision making structure for multi-way branching to solve the given problem.

Learning Objectives...

- To understand Basic Operators in Python Programming
- To learn Control Flow and Conditional Statements in Python
- To study Looping in Python Programming
- To understand Loop Manipulation Statements in Python

2.0 INTRODUCTION

- Operators are the constructs which can manipulate the value of operands. Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator. The Python language provides a rich set of operators.
- 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.
- In any programming language, a program is written as a set of instructions. The instructions written in programs are termed as statements.
- 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.
- But in some situations, the programmer may need to alter the normal flow of execution of a program or to perform the same operations a number of times.
- For this purpose, Python provides a control structure which transfers the control from one part of the program to some other part of the program.
- A control structure is a statement that determines the control flow of the set of instructions i.e. a program. Control statements are the set of statements that are responsible to change the flow of execution of the program.
- There are different types of control statements supported by Python programming like decision/conditional control, loop/iteration control and jump or loop control.

2.1 OPERATORS

- An operator is a symbol which specifies a specific action. An operator is a special symbol that tells the interpreter to perform a specific operation on the operands. The operands can be literals, variables or expressions.
- An operand is a data item on which operator acts. Operators are the symbol, which can manipulate the value of operands. Some operators require two operands while others require only one.

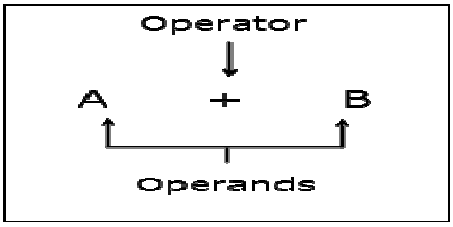


Fig. 2.1: Concept of Operator and Operands

- Consider the expression 5 + 2 = 7. Here, 5 and 2 are called the operands and + is called the operator.
- In Python, the operators can be unary operators or binary operator.

1. Unary Operators:

- Unary operators are operators with only one operand. These operators are basically used to provide sign to the operand. +, – ~ are called unary operators.

  Syntax: `operator operand`

Example:

```
>>> x=10
>>> +x
10
>>> -x
-10
>>> ~x
-11
>>>
```

- The (invert (~) operator returns the bitwise inversion of long integer arguments. Inversion of x can be computed as – (x+1).

2. Binary Operator:

- Binary operators are operators with two operands that are manipulated to get the result. They are also used to compare numeric values and string values.

  Syntax: `operand1 operator operand2`

- Binary operators are: **, *, /, %, +, -, <<, >>, &, |, ^, <, >, <=, >=, ==, !=, <>.

Expression:

- An expression is nothing but a combination of operators, variables, constants and function calls that results in a value.
- In other words, expression is a combination of literals, variables and operators that Python evaluates to produce a value.

  For examples:
  ```
  1 + 8
  (3 * 9) / 5
  a * b + c * 3
  ```

- Python operators allow programmers to manipulate data or operands. The types of operators supported by Python includes Arithmetic operators, Assignment operators, Relational or

Comparison operators, Logical operators, Bitwise operators, Identity operators and Membership operators.

2.1.1 Arithmetic Operators

- The arithmetic operators perform basic arithmetic operations like addition, subtraction, multiplication and division. All arithmetic operators are binary operators because they can perform operations on two operands. There are seven arithmetic operators provided in Python programming such as addition, subtraction, multiplication, division, modulus, floor division, and exponential operators.
- Assume variable a holds the value 10 and variable b holds the value 20.

Sr. No.	Operator Type	Operator	Description	Example
1.	+	Addition	Adds the value of the left and right operands.	`>>> a+b` `30`
2.	–	Subtraction	Subtracts the value of the right operand from the value of the left operand.	`>>> b-a` `10`
3.	*	Multiplication	Multiplies the value of the left and right operand.	`>>> a*b` `200`
4.	/	Division	Divides the value of the left operand by the right operand.	`>>> b/a` `2.0`
5.	**	Exponent	Performs exponential calculation.	`>>> a**2` `100`
6.	%	Modulus	Returns the remainder after dividing the left operand with the right operand.	`>>> a%b` `10`
7.	//	Floor Division	Division of operands where the solution is a quotient left after removing decimal numbers.	`>>> b//a` `2`

2.1.2 Assignment Operators (Augmented Assignment Operators)

- Assignment operators are used in Python programming to assign values to variables.
- The assignment operator is used to store the value on the right-hand side of the expression on the left-hand side variable in the expression.
- For example, a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.
- Following table shows assignment operators in Python programming:

Sr. No.	Operator	Description	Example
1.	=	Assigns values from right side operands to left side operand.	c = a + b assigns value of a + b into c
2.	+=	It adds right operand to the left operand and assign the result to left operand.	c += a is equivalent to c = c + a
3.	–=	It subtracts right operand from the left operand and assign the result to left operand.	c –= a is equivalent to c = c – a
4.	*=	It multiplies right operand with the left operand and assign the result to left operand.	c *= a is equivalent to c = c * a
5.	/=	It divides left operand with the right operand and assign the result to left operand.	c /= a is equivalent to c = c / a

6.	%=	It takes modulus using two operands and assign the result to left operand.	c %= a is equivalent to c = c % a
7.	**=	Performs exponential (power) calculation on operators and assign value to the left operand.	c **= a is equivalent to c = c ** a
8.	//=	Performs exponential (power) calculation on operators and assign value to the left operand.	c //= a is equivalent to c = c // a

2.1.3 Relational or Comparison Operators

- Comparison operators are binary operators and used to compare values. Relational operators either return True or False according to the condition.
- Assume variable a holds the value 10 and variable b holds the value 20

Sr. No.	Operator	Description	Example
1.	== (Equality Operator)	If the values of two operands are equal, then the condition becomes true.	`>>> (a==b)` `False`
2.	!= (Not Equality Operator)	If values of two operands are not equal, then condition becomes true.	`>>> (a!=b)` `True`
3.	> (Greater Than Operator)	If the value of left operand is greater than the value of right operand, then condition becomes true.	`>>> (a>b)` `False`
4.	< (Less Than Operator)	If the value of left operand is less than the value of right operand, then condition becomes true.	`>>> (a<b)` `True`
5.	>= (Greater Than Equal to Operator)	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	`>>> (a>=b)` `False`
6.	<= (Less Than Equal to Operator)	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	`>>> (a<=b)` `True`

2.1.4 Logical Operators

- The logical operators in Python programming used to combine one or more relational expressions that result in complex relational operations. The result of the logical operator is evaluated in the terms of True or False according to the result of the logical expression.
- Logical operators perform logical AND, logical OR and logical NOT operations. These operations are used to check two or more conditions. The resultant of this operator is always a Boolean value (True or false).
- Assume variable a holds True and variable b holds False then:

Sr. No.	Operator	Description	Example
1.	AND (Logical AND Operator)	If both the operands are true then condition becomes true.	(a and b) is False.
2.	OR (Logical OR Operator)	If any of the two operands are non-zero then condition becomes true.	(a or b) is True.
3.	NOT (Logical NOT Operator)	Used to reverse the logical state of its operand.	Not(a and b) is True.

2.1.5 Bitwise Operators

- Bitwise operators acts on bits and performs bit by bit operation. Python programming provides the bit manipulation operators to directly operate on the bits or binary numbers directly.
- When we use bitwise operators on the operands, the operands are firstly converted to bits and then the operation is performed on the bit directly.
- Bitwise operators in Python are binary operators and unary operators that can be operated on two operands or one operand.
- Following table shows bitwise operators assume a=10 (1010) and b=4 (0100).

Sr. No.	Operator	Description	Example
1.	and (Bitwise AND Operator)	This operation performs AND operation between operands. Operator copies a bit, to the result, if it exists in both operands	`a&b = 1010 & 0100 = 0000 =0`
2.	| (Bitwise OR Operator)	This operation performs OR operation between operands. It copies a bit, if it exists in either operand.	`a|b = 1010 | 0100 = 1110 = 14`
3.	^ (Bitwise XOR Operator)	This operation performs XOR operations between operands. It copies the bit, if it is set in one operand but not both.	`a^b=1010 ^ 0100 = 1110 =14`
4.	~ (Bitwise Ones Complement Operator)	It is unary operator and has the effect of 'flipping' bits i.e. opposite the bits of operand.	`~a= ~ 1010 = 0101`
5.	<< (Bitwise Left Shift Operator)	The left operand's value is moved left by the number of bits specified by the right operand.	`a<<2 = 1010 << 2 =101000 = 40`
6.	>> (Bitwise Right Shift Operator)	The left operand's value is moved right by the number of bits specified by the right operand.	`a>>2 = 1010 >> 2 =0010 = 2`

- Following table shows the outcome of each operations:

A	B	A&B	A|B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

2.1.6 Identity Operators

- Sometimes, in Python programming a need to compare the memory address of two objects; this is made possible with the help of the identity operator.
- Identity operators are used to check whether both operands are same or not. Python provides 'is' and 'is not' operators which are called identity operators and both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Sr. No.	Operator	Description	Example

1.	is	Return true, if the variables on either side of the operator point to the same object and false otherwise.	>>> a=3
			>>> b=3
			>>> print(a is b)
			True

2.	is not	Return false, if the variables on either side of the operator point to the same object and true otherwise.	>>> a=3
			>>> b=3
			>>> print(a is not b)
			False

Example 1:

```
>>> a=3
>>> b=3.5
>>> print(a is b)
False
>>> a=3
>>> b=4
>>> print(a is b)
False
>>> a=3
>>> b=3
>>> print(a is b)
True
>>>
```

Example 2:

```
>>> x=10
>>> print(type(x) is int)
True
>>>
```

Example 3:

```
>>>x2 = 'Hello'
>>>y2 = 'Hello'
>>>print(x2 is y2)
True
>>>x3 = [1,2,3]
>>>y3 = [1,2,3]
>>>print(x3 is y3)
False
>>>x4=(1,2,3)
>>> y4=(1,2,3)
>>> print(x4 is y4)
False
```

2.1.7 Membership Operators

- The membership operators in Python are used to find the existence of a particular element in the sequence, and used only with sequences like string, tuple, list, dictionary etc.

- Membership operators are used to check an item or an element that is part of a string, a list or a tuple. A membership operator reduces the effort of searching an element in the list.
- Python provides 'in' and 'not in' operators which are called membership operators and used to test whether a value or variable is in a sequence.

Sr. No.	Operator	Description	Example

1.	in	True if value is found in list or in sequence, and false it item is not in list or in sequence	>>> x="Hello World"
			>>> print('H' in x)
			True

2.	not in	True if value is not found in list or in sequence, and false it item is in list or in sequence.	>>> x="Hello World"
			>>> print("Hello" not in x)
			False

Example:

```
>>> x="Hello World"             # using string
>>> print("H" in x)
True
>>> print("Hello" not in x)
False
>>> y={1:"a",2:"b"}              # using list
>>> print(1 in y)
True
>>> print("a" in y)
False
>>> z=("one","two","three")      # using tuple
>>> print ("two" in z)
True
```

2.1.8 Python Operator Precedence and Associativity

- An expression may include some complex operations and may contain several operators. In such a scenario, the interpreter should know the order in which the operations should be solved. Operator precedence specifies the order in which the operators would be applied to the operands.
- Moreover, there may be expressions in which the operators belong to the same group, and then to resolve the operations, the associativity of the operators would be considered.
- The associativity specifies the order in which the operators of the same group will be resolved, i.e., from left to right or right to left.

1. Python Operator Precedence:

- 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.

Example 1:

10–4*2 answer is 2 because multiplication has higher precedence than subtraction.

But we can change this order using parentheses () as it has higher precedence.

(10–4)*2 answer is 12

Example 2:

10+5/5

- When given expression is evaluated left to right answer becomes 3. And when expression is evaluated right to left, the answer becomes 11.

- Therefore, in order to remove this problem, a level of precedence is associated with the operators. Precedence is the condition that specifies the importance of each operator relative to the others.

- The following table lists all operators from highest precedence to the lowest.

Sr. No.	Operator	Name/Description	Assoicativity
1.	(), []	Parentheses	Left to Right
2.	**	Exponent	Right to Left
3.	+x, –x, ~x	Unary plus, Unary minus, Bitwise NOT	Right to Left
4.	*, /, //, %	Multiplication, Division, Floor division, Modulus	Left to Right
5.	+, –	Addition, Subtraction	Left to Right
6.	<<, >>	Bitwise left and right shift operators	Left to Right
7.	&	Bitwise AND	Left to Right
8.	^	Bitwise XOR	Left to Right
9.	|	Bitwise OR	Left to Right
10.	<=, <, >, >=	Comparison Operator	Left to Right
11.	<> == !=	Equality operators	Left to Right
12.	= %= /= //= -= += *= **=	Assignment Operators	Right to Left
13.	is, is not	Identity	Left to Right
14.	in, not in	Membership operators	Left to Right
15.	Not, OR, AND	Logical Operators NOT, AND, OR	Left to Right

2. Associativity of Pythons Operators:

- When two operators have the same precedence, associativity helps to determine which the order of operations. Associativity decides the order in which the operators with same precedence are executed.
- There are two types of associativity.

  (i) Left-to-right: The operator of same precedence is executed from the left side first.

  (ii) Right-to-left: The operator of same precedence is executed from the right side first.
- Most of the operators in Python have left-to-right associativity.

Example:

```
>>> 5*2//3
3
>>> 5*(2//3)
0
```

2.2 CONTROL FLOW

- A program's control flow is the order in which the program's code executes. The control flow of a Python program is regulated by conditional statements, loops and loop manipulation (jump) statements.
- Python programming provides a control from are part of the program to some other part of program. A control structure is a statement that determines the control flow of the set of instruction.
- The control flow is refer to statement sequencing in a program together desire result. In this section, we introduce statements that allow us to change the control flow, using logic about the values of program variables.

2.3 CONDITIONAL STATEMENTS/DECISION MAKING STATEMENTS

- Decision making statements an those block of statements that executes a particular block according to the Boolean expression evaluation and the condition is checked and block as statements is executed according to the evaluated condition.
- In Python, conditional statements are used to determine if a specific condition is met by testing whether a condition is True or False. Conditional statements are used to determine how a program is executed.
- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- Decision structures evaluate multiple expressions which produce True or False as outcome. We need to determine which action to take and which statements to execute if outcome is True or False otherwise.
- Python decision making statements includes, if statements, if-else statements, nested-if statements, multi-way if-elif-else statements.

2.3.1 if Statement

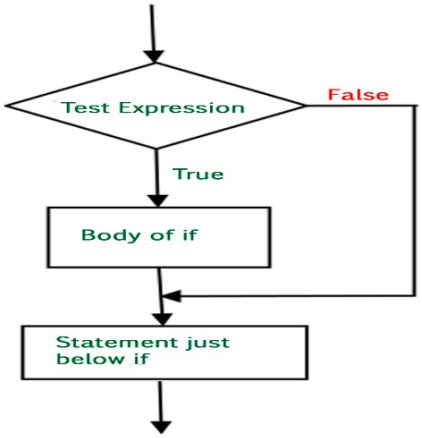
- The if statement executes a statement if a condition is true.

Syntax:

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

- Note that indentation is required for statements which are under if condition.

Control Flow Diagram of if Statement:



Example:

```
i=10
if(i<15):
print("i is less than 15")
print("This statement is not in if")
```

Output:

```
i is less than 15
This statement is not in if
```

Example 1: To find out absolute value of an input number.

```
x=int(input("Enter an integer number:"))
y=x
if (x<0):
    x=-x
print('Absolute value of',y,'=',x)
```

Output:

```
Enter an integer number:3
Absolute value of 3 = 3
Enter an integer number:-3
Absolute value of -3 = 3
```

Example 2: To find whether a number is even or odd.

```
number=int(input("Enter any number: "))
if(number%2)==0:
      print(number, " is even number")
else:
      print(number," is odd number")
```

Output:

```
Enter any number: 10
10  is even number
Enter any number: 11
11  is odd number
```

2.3.2 if-else Statement

- if statements executes when the conditions following if is true and it does nothing when the condition is false. The if-else statement takes care of a true as well as false condition.

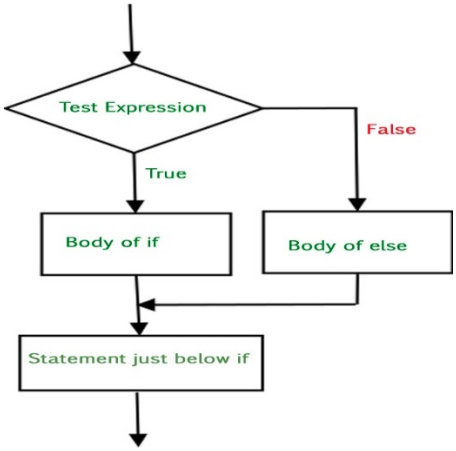
Syntax:

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

OR

```
if condition:
   if_block
else:
   else_block
```

Control flow Diagram of if else statement:



Example:

```
i=20
if(i<15):
print("i is less than 15")
else:
print("i is greater than 15")
```

Output:

```
i is greater than 15
```

Example: To check if the input year is a leap year of not.

```
year=int(input("Enter year to be checked:"))
if(year%4==0 and year%100!=0 or year%400==0):
print(year, " is a leap year!")
else:
print(year, " isn't a leap year!")
```

Output:

```
Enter year to be checked:2016
2016  is a leap year!
Enter year to be checked:2018
2018  isn't a leap year!
```

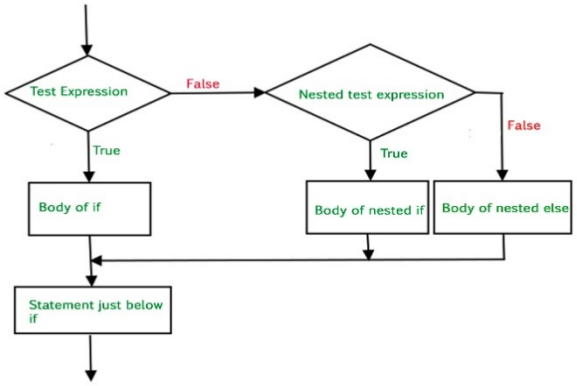
2.3.3 Nested if Statements

- When a programmer writes one if statement inside another if statement then it is called a nested if statement.

Syntax:

```
if condition1:
    if condition2:
        statement1
    else:
        statement2
    else:
        statement3
```

Control Flow diagram of Nested if Statement:



Example:

```
a=30
b=20
c=10
if (a>b):
    if (a>c):
    print("a is greater than b
    and c")
else:
    print("a is less than b
    and c")
print("End of Nested if")
```

Output:

```
a is greater than b and c
End of Nested if
```

2.3.4 Multi-way if-elif-else (Ladder) Statements

- Here, a user can decide among multiple options. The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

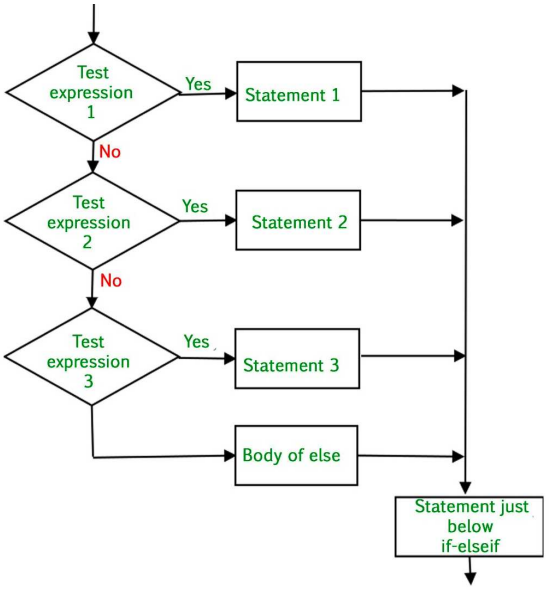
Syntax:

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

Control Flow Diagram for if-elif-if statements:

Example:

```
i = 20
if (i == 10):
    print ("i is 10")
elif (i == 15):
    print ("i is 15")
```




```
    elif (i == 20):
        print ("i is 20")
    else:
        print ("i is not present")
```

Output:

```
i is 20
```

Example: To check the largest number among the three numbers.

```
x=int(input("Enter first number: "))
y=int(input("Enter second number: "))
z=int(input("Enter third number: "))
if(x>y) and (x>z):
    l=x
elif(y>x) and (y>z):
    l=y
else:
    l=z
print("largest number is: ",l)
```

Output:

```
Enter first number: 10
Enter second number: 20
Enter third number: 30
largest number is:  30
```

2.4 LOOPING IN PYTHON

- A loop statement allows us to execute a statement or group of statements multiple times, this is called iteration.
- Looping control statements (or interactive control statements) are repeatedly execute a set of statements until a certain (stated) condition is met.
- Python programming language provides three types of loops to handle looping requirements namely, while loop, for loop and nested loops.

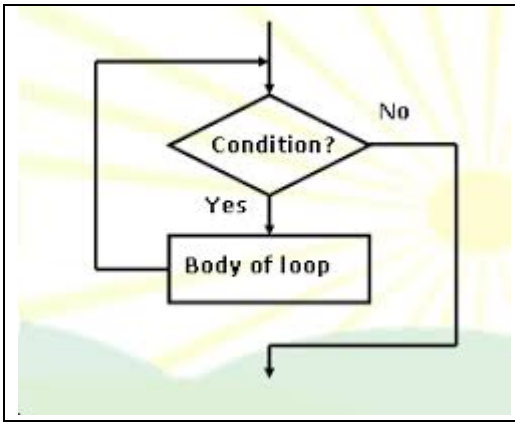
2.4.1 while Loop

- It is the most basic looping statement in Python. It executes a sequence of statements repeatedly as long as a condition is true.

Syntax:

```
while expression:
    statement(s)
```

Control Flow Diagram for while Loop:	**Example:**
	`count=0 #initialize counter`




```
while count<=3:          #test condition
print("count= ",count) #print value
count=count+1          #increment counter
```

Output:

```
count=0
count=1
count=2
count=3
```

Example: To display Fibonacci series.

```
num=int(input("Enter how many number u want to display: "))
x=0
y=1
c=2
print("Fibonacci Sequence is:")
print(x)
print(y)
while(c<num):
    z=x+y
    print(z)
    x=y
    y=z
    c+=1
```

Output:

```
Enter how many number u want to display: 10
Fibonacci Sequence is:
0
1
1
2
3
5
8
13
21
34
```

Additional Programs:

1. **Program to find out the reverse of the given number.**

```
n=int(input('Enter a number: '))
rev=0
while(n>0):
    rem=n%10
    rev=rev*10+rem
    n=int(n/10)
```

```
print('Reverse of number=',rev)
```

Output:

```
Enter a number: 123
Reverse of number= 321
```

2. Program to find sum of digit of a given number.

```
n=int(input('Enter a number:'))
sum=0
while(n>0):
    rem=n%10
    sum=sum+rem
    n=int(n/10)
print('Sum of number=',sum)
```

Output:

```
Enter a number: 123
Sum of number= 6
```

3. Program to check whether the input number is Armstrong.

```
n=int(input('Enter a number:'))
num=n
sum=0
while(n>0):
    rem=n%10
    sum=sum+rem*rem*rem
    n=int(n/10)
if num==sum:
print(num, "is armstrong")
else:
print(num, "is not armstrong")
```

Output:

```
Enter a number: 153
153 is armstrong
Enter a number: 123
123 is not armstrong
```

2.4.2 for Loop

- In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is "for in" loop which is similar to for each loop in other languages.
- The Python for loop iterates through a sequence of objects, i.e. it iterates through each value in a sequence, where the sequence of object holds multiple items of data stored one after another.

Syntax:

```
for var in sequence:
   Statements
```

Example:	

`list=[10,20,30,40,50]`
`for x in list:`
`print(x)`	

Output:

```
10
20
30
40
50
```

range() Function:

- 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.

  Syntax: `range(begin, end, step)`

  where,

  - **Start:** An integer number specifying at which position to start. Default is 0.
  - **End:** An integer number specifying at which position to end. i.e. the last number in the sequence.
  - **Step:** An integer number specifying the increment. Default is 1.

Example: For range() function.

```
>>> list(range(1,6))
[1, 2, 3, 4, 5]
```

- More examples of range() function:

Example	Output
`list(range(5))`	`[0, 1, 2, 3, 4]`
`list(range(1,5))`	`[1, 2, 3, 4]`
`list(range(1,10,2))`	`[1, 3, 5, 7, 9]`
`list(range(5,0,-1))`	`[5, 4, 3, 2, 1]`
`list(range(10,0,-2))`	`[10, 8, 6, 4, 2]`
`list(range(-4,4))`	`[-4, -3, -2, -1, 0, 1, 2, 3]`
`list(range(-4,4,2))`	`[-4, -2, 0, 2]`
`list(range(0,1))`	`[0]`
`list(range(1,1))`	`[]` Empty
`list(range(0))`	`[]` Empty

Example:

```
for i in range(1,11):
    print (i, end=' ')
```

Output:

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

- The print() function has end=' ' which appends a space instead of default newline. Hence, the numbers will appear in one row.

Additional Programs:

1. Program to print prime numbers in between a range.

```
start=int(input("Enter starting number: "))
end=int(input("Enter ending number: "))
for n in range(start,end + 1):
    if (n>1):
        for i in range(2,n):
```

```
            if(n%i)== 0:
                break
        else:
            print(n)
```

Output:

```
Enter starting number: 1
Enter ending number: 20
2
3
5
7
11
13
17
19
```

---

2. Program to check whether the entered number is prime or not.

```
n=int(input("Enter a number:"))
for i in range(2,n+1):
      if n%i==0:
          break
if i==n:
print(n," is prime number")
else:
print(n," is not a prime number")
```

Output:

```
Enter a number:5
5  is prime number
Enter a number:4
4  is not a prime number
```

---

3. Program to print and sum of all even numbers between 1 to 20.

```
sum=0
for i in range(0,21,2):
    print(i)
    sum=sum+i
print("Sum of Even numbers= ",sum)
```

Output:

```
0
2
4
6
8
10
12
14
16
18
```

```
20
Sum of Even numbers=  110
```

2.4.3 Nested for and while Loops

- Loops within the loops or when one loop is inserted completely within another loop, then it is called nested loop. Both for and while loop statement can be nested.

Syntax:

```
for var in sequence:                AND          while expression:
    for var in sequence:                             while expression:
        statements(s)                                    statement(s)
    statements(s)                                    statement(s)
```

Example: For nested loop (for and while).

```
for i in range(1,5):
    for j in range(1,(i+1)):
        print (j, end=' ')
    print()
```

Output:

```
1
1 2
1 2 3
1 2 3 4
```

```
i = 1
while i < 5:
    j = 1
    while j < (i+1):
        print(j, end=' ')
        j = j + 1
    i = i + 1
    print()
```

Output:

```
1
1 2
1 2 3
1 2 3 4
```

- Last print() will be executed at the end of inner for loop.

Additional Programs:

1. Program to print following pyramid:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
for i in range(1,6):
    for j in range(1,i+1):
        print(j,end=' ')
    print()
```

2. Program to print following pyramid:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
for i in range(1,6):
    for j in range(1,i+1):
        print(i,end=' ')
```

```
    print()
```

3. Program to print following pyramid:

```
1
2 3
4 5 6
7 8 9 1
2 3 4 5 6
count=1
for i in range(1,6):
   for j in range(1,i+1):
      print(count,end=' ')
      count=count+1
      if count>9:
         count=1
   print()
```

4. Program to print following pyramid:

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
count=1
for i in range(1,6):
    for j in range(1,i+1):
      print(count,end=' ')
      count=count+1
   print()
```

5. Program to print pyramid:

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
for row in range(1,6):
    for sp in range(1,6-row):
print(' ',end=' ')
    for col in range(1,row+1):
print(col, end=' ')
    for erow in range(col-1,0,-1):
print(erow,end=' ')
    print()
```

2.5 LOOP MANIPULATION/LOOP CONTROL STATEMENTS

- In Python, loop statements give us a way execute the block of code repeatedly. But sometimes, we may want to exit a loop completely or skip specific part of loop when it meets a specified condition. It can be done using loop control mechanism.

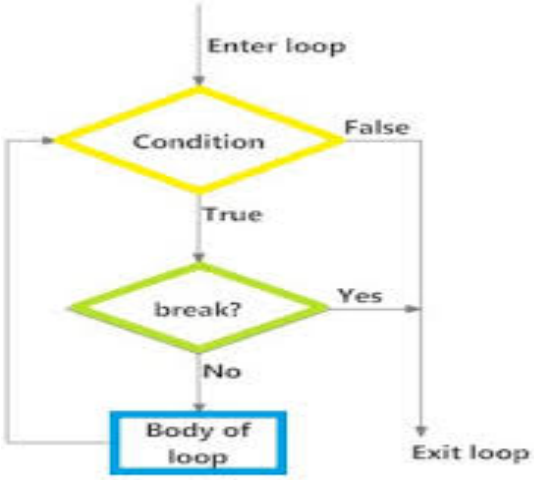
- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Loop control statement in Python is basically used to terminate a loop or skip the particular code in the block or it can also be used to escape the execution of the program.
- The loop control statements in Python programming includes break statement, continue statement and pass statement.

2.5.1 break Statement

- The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

Syntax: `break`

Control Flow Diagram for break Statement:



Example: For break statement.

```
i=0
while i<10:
    i=i+1
    if i==5:
        break
    print("i= ",i)
```

Output:

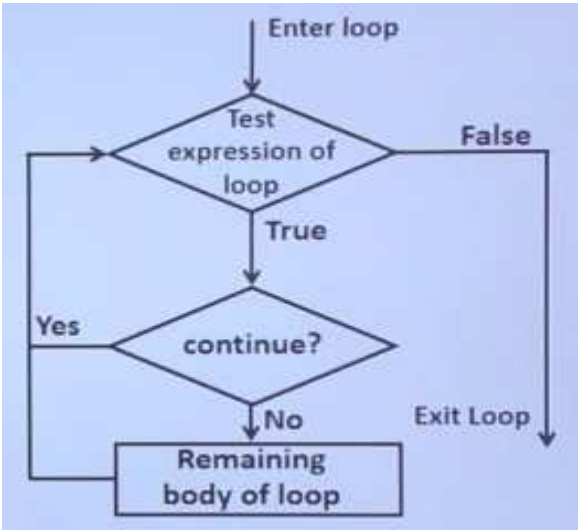
```
i=1
i=2
i=3
i=4
```

2.5.2 continue Statement

- The continue statement in Python returns the control to the beginning of the while loop.
- The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Syntax: `continue`

Control Flow Diagram for break Statement.



Example: For continue statement.

```
i=0
while i<10:
    i=i+1
    if i==5:
        continue
    print("i= ",i)
```

Output:

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

2.5.3 Pass Statement

- It is used when a statement is required syntactically but we do not want any command or code to execute. A pass statement in Python also refers to as a will statement.
- The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet

Syntax: `pass`

Example: For pass statement.

```
for i in range(1,11):
    if i%2==0:         # check if the number is even
        pass           # (No operation)
    else:
        print("Odd Numbers: ",i)
```

Output:

```
Odd Numbers:  1
Odd Numbers:  3
Odd Numbers:  5
Odd Numbers:  9
Odd Numbers:  7
```

Additional Programs:

1. Program to find factorial of a given number.

```
num=int(input("Enter Number:"))
fact=1
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("The factorial of ",num," is ",fact)
```

Output:

```
Enter Number: 5
The factorial of  5  is  120
```

2. Program to print multiplication table of the given number.

```
n=int(input('Enter a number: '))
for i in range (1,11):
print(n,' * ',i,' = ',n*i)
```

Output:

```
Enter a number: 3
3  *  1  =  3
3  *  2  =  6
3  *  3  =  9
3  *  4  =  12
3  *  5  =  15
3  *  6  =  18
3  *  7  =  21
```

```
3  *  8  =  24
3  *  9  =  27
3  *  10  =  30
```

3. Program to find out whether the input number is perfect number or not.

```
n=int(input("Enter number"))
sum=0
for i in range(1,n):
      if n%i==0:
          sum=sum+i
if sum==n:
print(n,' is perfect number')
else:
print(n,' is not perfect number')
```

Output:

```
Enter number 20
20  is not perfect number
Enter number 28
28  is perfect number
```

4. Program to generate Student Result. Accept marks of five subject and display result according to following conditions:

Percentage	Division
>=75	First class with Distinction
>=60 and <75	First Class
>=45 and <60	Second Class
>=40 and <45	Pass
<40	Fail

```
m1=int(input("Enter marks of Subject-1:"))
m2=int(input("Enter marks of Subject-2:"))
m3=int(input("Enter marks of Subject-3:"))
total=m1+m2+m3
per=total/3
print("Total Marks=",total)
print("Percentage=",per)
if per >= 75:
       print("Distinction")
elif per >=60 and per<75:
print("First class")
elif per >=45 and per<60:
print("Second class")
elif per >=40 and per<45:
    print("Pass")
else:
    print("Fail")
```

Output:

```
Enter marks of Subject-1:60
```

```
Enter marks of Subject-1:70
Enter marks of Subject-1:80
Total Marks= 210
Percentage= 70.0
First class
```

5. Python program to perform Addition Subtraction Multiplication and Division of two numbers.

```
num1 = int(input("Enter First Number: "))
num2 = int(input("Enter Second Number: "))
print("Enter which operation would you like to perform?")
ch = input("Enter any of these char for specific operation +,-,*,/: ")
result = 0
if ch == '+':
    result = num1 + num2
elifch == '-':
    result = num1 - num2
elifch == '*':
    result = num1 * num2
elifch == '/':
    result = num1 / num2
else:
print("Input character is not recognized!")
print(num1, ch , num2, ":", result)
```

Output:

```
Enter First Number: 20
Enter Second Number: 10
Enter which operation would you like to perform?
Enter any of these char for specific operation +,-,*,/: *
20 * 10 : 200
```

6. Program to check whether a string is a palindrome or not.

```
string=input("Enter string:")
if(string==string[::-1]):
print("The string is a palindrome")
else:
print("The string isn't a palindrome")
```

Output:

```
Enter string: abc
The string isn't a palindrome
Enter string: madam
The string is a palindrome
```

7. Program to check whether a number is a palindrome or not.

```
num = int(input("enter a number: "))
temp = num
rev = 0
while temp != 0:
   rev = (rev * 10) + (temp % 10)
   temp = temp // 10
```

```
if num == rev:
   print("number is palindrome")
else:
   print("number is not palindrome")
```

Output:

```
enter a number: 121
number is palindrome
enter a number: 123
number is not palindrome
```

8. Program to return prime numbers from a list.

```
list=[3,2,9,10,43,7,20,23]
print("list=",list)
l=[]
print("Prime numbers from the list are:")
for ain list:
    prime=True
    for i in range(2,a):
        if (a%i==0):
            prime=False
            break
    if prime:
l.append(a)
print(l)
```

Output:

```
list= [3, 2, 9, 10, 43, 7, 20, 23]
Prime numbers from the list are:
[3, 2, 43, 7, 23]
```

9. Program to add, subtract, multiply and division of two complex numbers.

```
print("Addition of two complex numbers : ",(4+3j)+(3-7j))
print("Subtraction of two complex numbers : ",(4+3j)-(3-7j))
print("Multiplication of two complex numbers : ",(4+3j)*(3-7j))
print("Division of two complex numbers : ",(4+3j)/(3-7j))
```

Output:

```
Addition of two complex numbers :  (7-4j)
Subtraction of two complex numbers :  (1+10j)
Multiplication of two complex numbers :  (33-19j)
Division of two complex numbers :  (-0.1551724137931034 8+0.6379310344827587j)
```

10. Program to find the best of two test average marks out of three test's marks accepted from the user.

```
n1=int(input('enter a number'))
n2=int(input('enter 2nd number'))
n3=int(input('enter the 3rd number'))
avg1=(n1+n2)/2
avg2=(n2+n3)/2
avg3=(n3+n1)/2
maxm=max(avg1, avg2, avg3)
```

```
print(maxm)
```

11. Print patterns of (*) using loop.

(i)
```
for i in range(0, 5):
    for j in range(0, i+1):
        print("* ",end="")
    print("\r")
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

(ii)
```
for i in range(0, 5):
    num = 1
    for j in range(0, i+1):
        print(num, end=" ")
    num = num + 1
    print("\r")
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Practice Questions

1. What is operator? Which operators used in Python?
2. What is meant by control flow of a program?
3. Define the terms: (i) Loop, (ii) Program, (iii) Operator, (iv) Control flow.
4. What are the different loops available in Python?
5. What happens if a semicolon (;) is placed at the end of a Python statement?
6. Explain about different logical operators in Python with appropriate examples.
7. Explain about different relational operators in Python with examples.
8. Explain about membership operators in Python.
9. Explain about Identity operators in Python with appropriate examples.
10. Explain about arithmetic operators in Python.
11. List different conditional statements in Python.
12. What are the different nested loops available in Python?
13. What are the different loop control (manipulation) statements available in Python? Explain with suitable examples.
14. Explain if-else statement with an example.
15. Explain continue statement with an example.
16. Explain use of break statement in a loop with example.
17. Predict output and justify your answer: (i)–11%9 (ii) 7.7//7 (iii) (200-70)*10/5 (iv) 5*1**2.
18. What the difference is between == and is operator in Python?
19. List different operators in Python, in the order of their precedence.
20. Write a Python program to print factorial of a number. Take input from user.

21. Write a Python program to calculate area of triangle and circle and print the result.
22. Write a Python program to check whether a string is palindrome.
23. Write a Python program to print Fibonacci series up to n terms.
24. Write a Python program to print all prime numbers less than 256.
25. Write a Python program to find the best of two test average marks out of three test's marks accepted from the user.

■■■