Python Programming 05101155

Prof. Bhavika Vaghela, Assistant Professor
Parul Institute of Computer Application - BCA

Parul® University

DIGITAL LEARNING CONTENT

CHAPTER-2

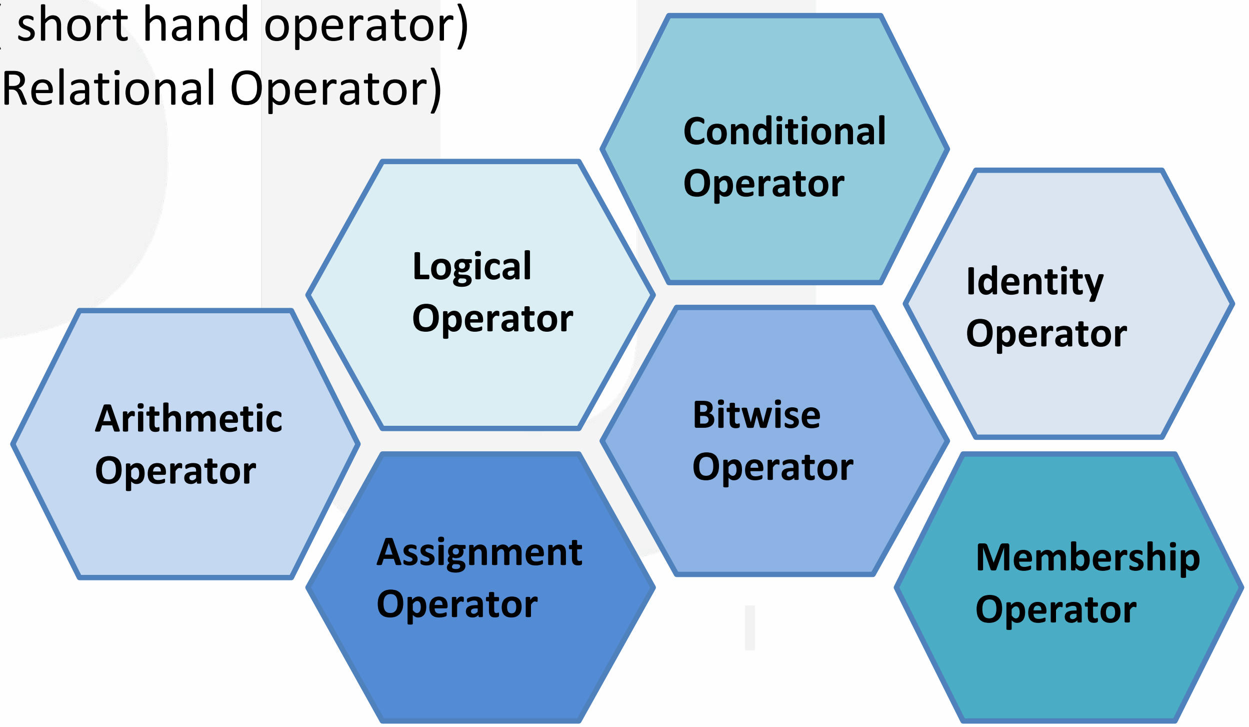
Operator, Conditional Statements and Looping in Python

Before starting operators in python lets brief it.

- Python operator is a Special symbol that performs an operation on one or more operands.
- An operand is a variable or a value on which we perform the operation.
- 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.
- Value on which operator operates is called the operand. As a value it can be any variable or data.
- Operators are the stakes of a program on which the logic is built in a particular programming language or we can in any software or application.

Python Support Seven types of Operators

- Arithmetic Operator
- Logical Operator
- Assignment Operator (short hand operator)
- Conditional Operator (Relational Operator)
- Bitwise Operator
- Identity Operator
- Membership Operator



Arithmetic Operator

- **Addition (+)** : use to add two or more operands. Value can be any integer or float. **if value is string or character than it perform concatenate operation.**
- **Subtraction(-)** : It is used to subtract the second operand from the first operand.
  Subtracts the value on the **right from the one on the left.**
- **Division(/)** : It returns the quotient after dividing the first operand by the second operand. **Notice that division results in a floating-point value.**
- **Multiplication(*)** : It is used to multiply one operand with the other.
- **Exponentiation(**)**: calculates the first operand power to second operand.
- **Modulo (%)** : Divides and returns the value of the remainder.
- **Floor Division(//)** : Divides and returns the integer value of the quotient.

Parul® University

DIGITAL LEARNING CONTENT

Example of Arithmetic Operator

```
>>> number1=10
>>> number2=2
>>> number1 + number2  (addition)
12
>>> number1-number2
(subtraction)
8
>>> number1*number2
(multiplication)
20
>>> number1/number2 (division)
5.0
>>> number1//number2  (floor
division)
5
>>> number1%number2  (modulo)
0
>>> number2**6   (exponent)
64
```

Relational (Conditional) Operator

- **Less than(<)** : checks if the value on the left of the operator is lesser than the one on the right.
- **Greater than(>)** : It checks if the value on the left of the operator is greater than the one on the right.
- **Less than or equal to(<=)** : It checks if the value on the left of the operator is lesser than or equal to the one on the right.
- **Greater than or equal to(>=)** : It checks if the value on the left of the operator is greater than or equal to the one on the right.
- **Equal to(= =)** : This operator checks if the value on the left of the operator is equal to the one on the right.
- **Not equal to(!=)**: It checks if the value on the left of the operator is not equal to the one on the right.
- **Relational Operator return Boolean value.**

Parul® University

Example of Relational (Conditional) Operator

```
number1=10
number2=2
>>> number1>number2   (Greater than)
True
>>> number1<number2    (less than)
False
>>> number1!=number2    (not equal to)
True
>>> number1<=number2   (less than equal to)
False
>>> number1>=number2  (greater than equal to)
True
>>> number1==number2   (equal to)
False
```

Assignment Operator

- **Assign(=)** : Assigns a value to the expression on the left.
- **Add and Assign(+=)** : Adds the values on either side and assigns it to the expression on the left.
- **Subtract and Assign(-=)** : Subtracts the value on the right from the value on the left. Then it assigns it to the expression on the left.
- **Divide and Assign(/=)** : Divides the value on the left by the one on the right. Then it assigns it to the expression on the left.
- **Multiply and Assign(*=)** : Multiplies the values on either sides. Then it assigns it to the expression on the left.
- **Modulus and Assign(%=)** Performs modulus on the values on either side. Then it assigns it to the expression on the left.

Assignment Operator Cont..

- **Exponent and Assign(**=)** : Performs exponentiation on the values on either side. Then assigns it to the expression on the left.
- **Floor-Divide and Assign(//**=) Performs floor-division on the values on either side. Then assigns it to the expression on the left.

Parul® University

DIGITAL LEARNING CONTENT

Example of Assignment Operator

```
>>> number1=10
>>> number1+=1
>>> print(number1)
11
>>> number1-=2
>>> print(number1)
9
>>> number1/=3
>>> print(number1)
3.0
```

```
>>> number2=9
>>> number2//=3
>>> print(number2)
3
>>> number2=10
>>> number2%=3
>>> print(number2)
1
```

```
>>> number1*=3
>>> print(number1)
9.0
>>> number1//=3
>>> print(number1)
3.0
>>> number3=4
>>> number3**=3
>>> print(number3)
64
```

Logical Operator

- The logical operators are used to make a decision.
- These are work as conjunctions that you can use to combine more than one condition. We have three Python logical operator – and, or, and not.
- **Returns Boolean value True or False.**

- **and** : If the conditions on both the sides of the operator are true, then the expression as a whole is true.
- **or** : The expression is false only if both the statements around the operator are false. Otherwise, it is true.
- **not** : This inverts the Boolean value of an expression. It converts True to False, and False to True.

Truth Table for Logical Operator And & or

Condition 1	Condition 2	and operator	or operator
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Parul® University

DIGITAL LEARNING CONTENT

Truth Table for Logical Operator not

Condition	not
True	False
True	True

Example of Logical operator
** it will check condition both the side than gives output.

```
>>> number1=10
>>> number2=2
>>> number1>5 and number2<3
True
>>> number1>11 or number2<3
True
```

```
>>> number1>5 and number2<1
False
>>>number1>20 or number2<1
False
>>> not(number1>11)
True
```

Identity Operator (is and is not)

- These operators test if the two operands share an identity (Memory). We have two identity operators- **‘is’ and ‘is not’.**
- They are used to check if two values (or variables) are located on the same part of the memory. It returns Boolean value True or False
- **Example**

```
>>> str1="apple"
>>> str2="apple"
>>> str1 is str2
True
>>> str1 is not str2
False
```

```
>>> l1=[1,2,3]
>>> l2=[1,2,3]
>>> l1 is l2
False
>>> l1 is not l2
True
```

```
>>> l3=[10,20,30]
>>> l4=l3
>>> l3 is l4
True
>>> l3 is not l4
False
```

Membership Operator (in and not in)

- membership operators are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
- We have two membership operator in and not in. It will return Boolean value.
- **Example**

```
>>> tuple1=(1,2,"xyz",10.2,50)
>>> 4 in tuple1
False
>>> 4 not in tuple1
True
>>> "xyz" in tuple1
True
```

```
>>> "xyz" not in tuple1
False
```

Like tuple you can use membership Operator on any data types like String, list, set, dictionary, string

Bitwise Operator

- The bitwise operators perform bit by bit operation on the values of the two operands.
- If data is not in binary than it convert it first in binary and than perform operation.
- It returns decimal as a result.

- **Binary AND(&)** : it perform bit by bit and operation and if both bit at same place are 1 than put 1 else it will put 0.
- **Binary OR(|)** : it perform bit by bit or operation and if both bit at same place is 0 than put 0 else it will put 1.
- **Binary XOR(^)** : it perform bit by bit XOR (exclusive or) operation. It will put 0 if both bit at the same place is same else put 1

Bitwise Operator cont...

- **One's Complement(~) / binary negative** : It flips the bits. It will return 1 if bit is 0 and return 0 if bit is 1.
- **Left shift (<<)** : The left operand value is moved left by the number of bits present in the right operand. E.g. 10<<n (n is number of bits and 10 is value)
- **Right shift (>>)** : The left operand value is moved right by the number of bits present in the right operand. E.g 10>> n (n is number of bits and 10 is value)

- **Truth Table of Once Complement**

Bit Value	Once Complement (~)
1	0
0	1

Bitwise Operator cont...

Truth Table of and, or and xor

Bit value	Bit Value	and (&)	or (|)	Xor (^)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Example of Bitwise Operator

```
>>>bin(a)          here a=7 & b=3
0b0111
>>>bin(b)
0b0011
>>>print(a & b)
6        i.e. 0011
>>>print(a | b)
7 i.e. 0111
>>>print(~a)
8        i.e. 1000
>>>print(a^b)
4 i.e. 0100
**0b indicate binary value
```

```
>>> a=10
>>> bin(a)
'0b1010'
>>> a<<2    (left shift)
40
>>> bin(a<<2)
'0b101000'
>>> a>>2    (right shift)
2
>>> bin(a>>2)
'0b10‘
>>> a<<3   (left shift with 3 bit)
80
```

Conditional Execution

- For writing any useful programs in which if someone wants to check condition and perform operations based on that condition than we have to check condition.
- For check the condition we have statement named as "if" it will return Boolean value based on condition.
- If the condition become true than inside block is executed else not.
- We can also declare multiple if statement in single program.

```
x=10
if x > 0:
  print 'x is positive'
```

here for this code it will check value of x than if condition become true than inner block will be executed

Example of Conditional statement cont..

- As we can also declare more than one “if” within program

```
#program to check number type
Number1=10
if(Number1>0):
      print(“positive number”)
if(Number1<0):
      print(“negative Number”)
if(Number1==0):
      print(“number is zero”)
```

Program of Conditional if statement

```
#program to find number belongs to which range
number1=int(input(“enter any number between 1 to 100 : “)
if number1<21:
     print(“number between 1 to 20”)
if number1>20 and number1<41:
     print(“number between 21to 40”)
if number1>40 and number1<61:
     print(“number between 41 to 60”)
if number1>60 and number1<81:
     print(“number between 61 to 80”)
if number1>80 and number1<101:
     print(“number between 81 to 100”)
```

Alternative Execution

- A second way to conditional execution using if statement is alternative execution, here one can check two condition or one can execute alternate inner block based on the condition.
- For that we have combination of **if…else** statement.
- So that alternate is called **branch** as it is a branch in flow of execution based on condition.
- So in if…else block which is **declare using else statement is known as branch.**
- As like conditional statement we can also **declare multiple if..else block within single program.**

Example of alternative execution

```
Number1=int(input(“enter any number : “))
if (number1%2==0):
      print(“number is even”)
else:
      print(“number is odd”)
```

Here in above program first we check that number is divide by 2 if condition become TRUE than inner block will be executed else the alternate branch will be going to executed.

Example with multiple if else block

```
Number1=int(input(“enter any number : “))
if(Number1%2==0):
    print(“number is even”)
else:
    print(“number is odd”)
if(Number1%3==0):
    print(“number is odd and its divided by 3”)
else:
    print(“number is not divided by 3”)

#like wise you can check any number of conditions
```

Chained Condition

- Sometimes if we want to check multiple condition than its better to used chained condition rather to use alternate condition or conditional execution.
- For this we can use **if else..if else..if else..if else** block.
- This **else..if** stand as **elif** in python so there is no need to write full **else..if** rather than that we can write **if elif elif elif else**.
- **Each condition in chained condition executed in sequence if first condition become true than it not check further all listed condition.**
- This all condition again known as branch.
- It is not mandatory to write or declare else block but if we want to execute something if condition is not fulfill than its mandatory.

Example of Chained Condition

```
per = int(input(“enter percentage : “))
if(per>90 and per=<100):
    print(“you passed with higher distinction class”)
elif(per>75 and per<90):
    print(“you passed with distinction class”)
elif(per>60 and per<=75):
    print(“you passed with first class”)
elif(per>50 and per<61):
    print(“you passed with second class”)
elif(per>=35 and per<51):
    print(“you passed with pass class”)
else:
    print(“you are fail”)
```

Nested Condition

- Nested Condition mean condition within one condition.
- It mean it executed always in hierarchy. Mean if condition 1 is true than it going to check inner most condition and so on.
- But it become difficult to read so its better to not use.

```
if x == y:
    print 'x and y are equal'
else:
    if x < y:
        print 'x is less than y'
    else:
        print 'x is greater than y'
```

Loops in Python

- Why loop? Loop is basically use to iteration purpose.
- When someone wants to print "**Wel Come To Parul University**" than it is easy to write the same using loops rather to write print statement 100 times.
- By using loop one can reduce the length of code lines.
- It become easy to execute and debugging.
- Python mainly support two types of loop 1) **for loop** and 2) **while loop**.
- It not support do while loop as like c or any other programming language.
- But for loop is most preferred loop.

For loop

- for loop iterate using two variable one is iterrable variable that may any list, tuple, dictionary, string or one can also use integer value in **range()** function.
- Second variable of for loop which stores the consecutive value from iterrable variable or sequence.

Syntax

```
for vari in sequece:
    print(vari) or statement(vari)
```

About range() function

- range() function returns a sequence of number.
- Starting with **0 by default** if start value not mention.
- **Increment by 1 by default** if stop value not mention.
- And stop execution at **stop value – 1**.
- Syntax for range() is **range (start-value, stop-value, step-value)**

**** in range() function start-value and step value is optional but stop-value is compulsory (required).**

Example : for I in range(11):

```
for I in range(11):
        print(I)
```

**** it print 0 to 10**

Suppose X = range(5) here X store the sequence of 0 to 4

Example of for loop

```
#program to find sum of natural numbers up to given range
Num = int(input(“enter any number : “))
Sum=0
for vari in range(Num+1):
      Sum = Sum + vari
print(“sum of natural number up to given range is : “, Sum)

#program to print character of string
Message=input(“enter any string : “)

for vari in Message:
      print(vari)
```

Program to print floyed triangle using *

```
# number of rows
rows = 5
for i in range(0, rows):
    # nested loop for each column
    for j in range(0, i + 1):
        # print star
        print("*", end=' ')
    # new line after each row
    print("\n")
    #print("\r")  \r is for carriage return
```

Implement below listed Program using loop

1) WAP to Print Even number between 1 to 20 using for loop.
2) WAP to Print Odd numbers between given range. Take range from user.
3) WAP to Print numbers divisible by 3 from given range. Take range from user.
4) WAP to print floyed triangle of “*” using for loop.
5) WAP to find addition of only Even numbers between given range.

While loop

- Unlike for loop while loop is always depends on some condition to complete the execution.
- It will execute up to the given condition not become false.
- while loop is entry control loop.
- Some times programmer don't know the stage where to stop the execution at this stage it become infinite loop.

Syntax

```
while condition (or expression):
        block of code or
        execution statements
```

Example of While loop

```
#program to do summation up to number become 10
number=1
sum=0

while (number<=10):
      sum = sum + number
      number = number + 1
print("summation of numbers is : ", sum)
```

Example of While loop

```
#program to do check number is odd or even
#up to number become 10
number=1

while (number<=10):
      if(number%2==0):
            print(“number is even : “,number)
      else:
            print(“number is odd : “,number)
       number = number + 1
```

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in