Day-1:
21/06/2018

Workshop on "Introduction to Python"

Department Of
Computer Science
Engineering

SKFGI, Mankundu

1.History

Python was developed by Guido van Rossum in the early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. C, C++, ABC, Modula-3, Algol-68, SmallTalk, Unix shell and other programming and scripting languages influenced the development of Python. Python is copyrighted. Python source code is now available under the GNU General Public License (GPL).

2.Features

- **Simple to Understand**– Python offers has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below-

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

3.Basic Syntax of Python

Modes of Programming

- **Interactive Mode Programming**

  In this mode, interpreter is invoked without passing a script file as a parameter. The codes are written in the prompt without creating any file. Codes are written one after another in the prompt and by pressing enter key the written code is interpreted and if no error detected, then output is shown in the prompt; otherwise error message is displayed in the prompt.

  Examples:

  1.

  Type the following text at the Python prompt–

  >>> print("Hi Everyone")

  Now, press the Enter to see the following output-

  Hi Everyone

  2.

  Type the following text at the Python prompt–

  >>> 8+3

  Now, press the Enter to see the following output-

  11

  3.

  Type the following text at the Python prompt–

>>> 8-3

Now, press the Enter to see the following output-

5

4.

Type the following text at the Python prompt−

>>> 8*3

Now, press the Enter to see the following output-

24

5.

Type the following text at the Python prompt−

>>> 8/3

Now, press the Enter to see the following output-

2.6666

6.

Type the following text at the Python prompt−

>>> 8//3

Now, press the Enter to see the following output-

2

7.

Type the following text at the Python prompt−

>>> 9%2

Now, press the Enter to see the following output-

1

8.

Type the following text at the Python prompt−

>>> 7**2

Now, press the Enter to see the following output-

49

9.

Type the following text at the Python prompt−

>>> base=10

Press Enter

>>> height=20

Press Enter

>>> 0.5*base*height

Now, press the Enter to see the following output-

100.0

10.

Type the following text at the Python prompt−

>>> 5+2*3

Now, press the Enter to see the following output-

11

11.

Type the following text at the Python prompt−

>>> (5+2)*3

Now, press the Enter to see the following output-

21

- **Script Mode Programming**

  In this mode, users can write codes in files. Files are to be saved with .py extension. This mode allows the user to run the program by clicking the Run button. After clicking on the run button the interpreter is invoked and if the program is error free, then the output will be displayed, otherwise; error message will be conveyed to the user. This mode facilitates storing of programs in a file permanently so that ,it could be used in future also.

4.Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

5.Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the

Python keywords contain lowercase letters only. Following table shows the list of keywords available in Python-

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

6.Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Thus, in Python all the continuous lines indented with same number of spaces would form a block.

7.Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

Example:

```
total = x + \
        y + \
        z
```

8.Quotation in Python

Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines.

Example:

```
w = 'Hi'
s= "Hello Everyone Present Here"
p= """All the students who are interested in writing programs must attend this class of Python programming. This will increase your confidence and embed new technology in yourselves."""
```

9.Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Multi line comments is done by triple single quotes.

Example:

```
a=10
'''This is a python program
for the beginners
'''
#hello student
print(a)
```

Output:

```
10
```

10.Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block.

Example:

a= "hi hello"; b=5; print (a); print (b)

11.Python Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

12.Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Example:

```
pincode=712125          # An integer assignment

marks=87.50       # A floating point assignment

name="Sachin"         # A string assignment
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously.

Example:

a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

Example:

x,y,z=10,12.98,”Sourav”

Here, One integer object with values 10 and one float object with value 12.98 and one string object with value “Sourav”

13.Data Types of Python

The data stored in memory can be of many types. For example, a person's salary is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python offers the following five data types–

- Numbers
- Strings
- Lists
- Tuples
- Dictionaries

14.1Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

Example:

x = 100

y = 10.67

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

14.2Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number

explicitly from one type to another to satisfy the requirements of an operator or function parameter.

- Type int(x) to convert x to a plain integer.
- Type long(x) to convert x to a long integer.
- Type float(x) to convert x to a floating-point number.
- Type complex(x) to convert x to a complex number with real part x and imaginary part zero.
- Type complex(x, y) to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions
- Example

```
dec = int(input("Enter a decimal number: "))
print(bin(dec),"in binary.")
print(oct(dec),"in octal.")
print(hex(dec),"in hexadecimal." )
print("The ASCII value of '" + c + "' is",ord(c))
print(float(dec),"in float." )
print(str(5),"in String" )
print(chr(65),"in character " )
```

Address of a variable

Example:

```
a=10
print("Address of a in integer form is:",int(id(a)))
print("Address of a in hexadecimal form is:",hex(id(a)))
```

Output:

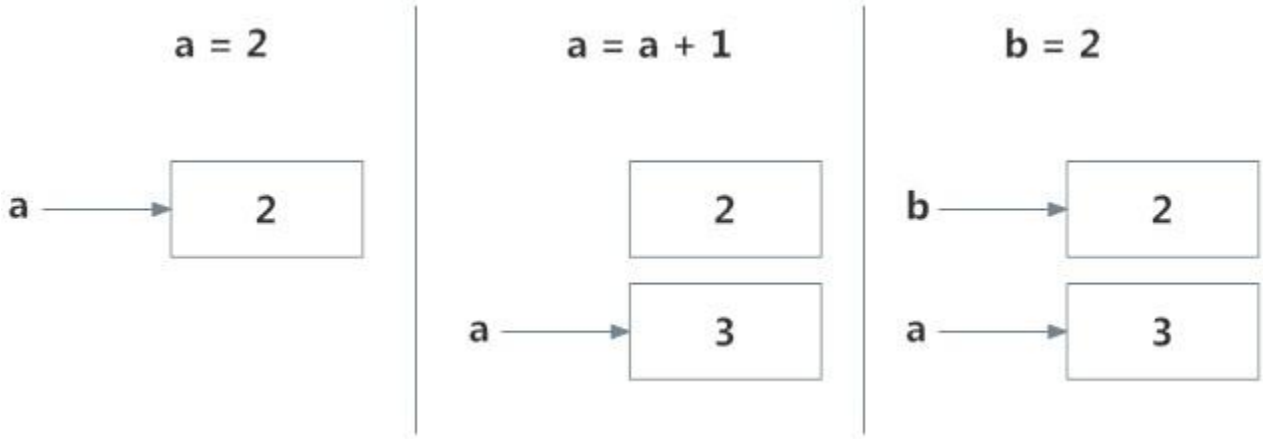
```
Address of a in integer form is: 1787159456
Address of a in hexadecimal form is: 0x6a85e3a0
```

Example2:

```
a = 2
print('id(a) =', id(a))
a = a+1
print('id(a) =', id(a))
print('id(3) =', id(3))
b = 2
```

```
print('id(2) =', id(2))
```

A diagram will help us explain this.



14.3Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

Example:

```
s="Hello Everyone"
print("String s is:")
print(s)
```

Output:

```
String s is:
Hello Everyone
```

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Example:

```
s1="India is"
s2="My Country"
print("Repeated String s1 is:")
print(s1*2)
print("Concatenated String is:")
print(s1+s2)
```

Output:

```
Repeated String s1 is:
```

India isIndia is
Concatenated String is:
India isMy Country

14.4 Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Example:

1.

```
list_new=[]   #Empty list. list_new is the name of the user defined list
print(list_new)
```

Output:

```
[]
```

2.

```
list_new=[100,23,45] # List with same types of elements
print(list_new)
```

Output:

```
[100, 23, 45]
```

3.

```
list_new=[2,5.37,'Python'] # List with different types of elements
print(list_new)
```

Output:

```
[2, 5.37, 'Python']
```

4.

```
list_new=[200,15.37,'Hello']
```

```
print("list_new[0] is =",list_new[0]) # Printing element at index 0 of the list
print("list_new[1] is =",list_new[1]) # Printing element at index 1 of the list
print("list_new[2] is =",list_new[2]) # Printing element at index 2 of the list
```

Output:

```
list_new[0] is = 200
list_new[1] is = 15.37
list_new[2] is = Hello
```

5.

```
list_new=[200,15.37,'Hello',8000]
print(list_new[0:3]) # Print elements starting from index 0 to 2
print(list_new[1:3]) # Print elements starting from index 1 to 2
print(list_new[1:]) # Print elements starting from index 1 to last index
```

Output:

```
[200, 15.37, 'Hello']
[15.37, 'Hello']
[15.37, 'Hello', 8000]
```

6.

```
list_new=[200,15.37,'Hello',8000]
print(list_new[-1]) # Print the 1th element from right; there is no concept of 0th element from right
```

Output:

```
8000
```

7.

```
list_new=[200,15.37,'Hello',8000]
print(list_new[-3]) # Print the 3th element from right
```

Output:

```
15.37
```

Matrices

In python, a matrix can be represented as a list of lists (a list, where each element is in turn a list).

Example:

1.

```
mat=[[1,2,3,4],[100,200,300,400],[1000,2000,3000,4000]] # mat is a 3X4 matrix
print("Matrix mat is:")
print(mat)
```

Output:

```
Matrix mat is:
[[1, 2, 3, 4], [100, 200, 300, 400], [1000, 2000, 3000, 4000]]
```

2.

```
m=[[12,34.5],   # matrix assignment in table form
    [23,87.9],
    [34,87.5]]
print("matrix m is:")
print(m)
```

Output:

```
matrix m is:
[[12, 34.5], [23, 87.9], [34, 87.5]]
```

3.

```
student=[['Shyamal',1,85.0],['Ashok',2,93.5]] # student matrix formation with name ,roll and marks
print("Matrix student is:")
print(student)
```

Output:

```
Matrix student is:
[['Shyamal', 1, 85.0], ['Ashok', 2, 93.5]]
```

4.

```
m=[[1,2,3],
  [10,20,30],
  [100,200,300]]
print(m[0]) #print the 0th row of the matrix
print(m[2]) #print the 2th row of the matrix
```

Output:

[1,2,3]

[100,200,300]

5.

```
m=[[1,2,3],
   [10,20,30],
   [100,200,300]]
print(m[0][0]) # print matrix element of 0th row and 0th column
print(m[0][2]) # print matrix element of 0th row and 2th column
print(m[2][1]) # print matrix element of 2th row and 1th column
```

Output:

1

3

200

6.

```
m1=[[1,2,3],
   [10,20,30],
   [100,200,300],
   [1000,2000,3000]]
print(m1[-1])  #print 1 th row from bottom, there is no concept of 0th row from bottom
print(m1[-3]) #print 3 th row from bottom
```

Output:

[1000, 2000, 3000]

[10, 20, 30]

7.

```
m2=[[1,2,3,4],
   [10,20,30,40],
   [100,200,300,400],
   [1000,2000,3000,4000]]
print(m2[0][-2]) #print element at 0 th row from top and 2th column from right of the row
```

```
print(m2[2][-3]) #print element at 2 th row from top and 3th column from right of the row
```

Output:

```
3
200
```

8.

```
m1=[[1,2,3,4],
   [10,20,30,40],
   [100,200,300,400],
   [1000,2000,3000,4000]]
print(m1[-3][-2]) #print element at 3th row from bottom and 2th column from right of the row
```

Output:

```
30
```

Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are:

- Lists are enclosed in third brackets i.e [ ] but tuples are enclosed in parentheses i.e ( )
- List elements and size can be changed but tuples cannot be updated. Tuples can be thought of as read-only lists.

Example:

1.

```
tuple1=(100,200)
print("Tuple is:")
print(tuple1)
```

Output:

```
Tuple is:
(100, 200)
```

2.

```
t1=('Sachin',1,93.5)
t2=('Sourav',2,81.0)
print("Tuples are:")
print(t1)
print(t2)
```

Output:

```
Tuples are:
('Sachin', 1, 93.5)
('Sourav', 2, 81.0)
```

3.

```
t1=(100,200.9,300,'Narayan')
print("t1[0] is=",t1[0]) #print 0th element of tuple t1
print("t1[1] is=",t1[1]) #print 1th element of tuple t1
print("t1[2] is=",t1[2]) #print 2th element of tuple t1
print("t1[3] is=",t1[3]) #print 3th element of tuple t1
```

Output:

```
t1[0] is= 100
t1[1] is= 200.9
t1[2] is= 300
t1[3] is= Narayan
```

4.

```
t1=(100,200.9,300,'Narayan')
print("t1[0] is=",t1[0])
t1[1]=452.8 #updating tuple element is not possible
```

Output:

```
Error: 'tuple' object does not support item assignment
```

5.

```
t1=('Harish','Ramen')
t2=(100,200)
t3=(3.45,7.98)
```

```
t4=t1+t2+t3  #Concatenating more than one tuples
print("Resultant tuple is:")
print(t4)
```

Output:

```
Resultant tuple is:
('Harish', 'Ramen', 100, 200, 3.45, 7.98)
```

6.

```
t1=('Hello')
print(4*t1) #print the content of tuple t1 4 times
```

Output:

```
HelloHelloHelloHello
```

7.

```
t1=('Hello','World','Everyone')
print("Length of tuple t1 is:",len(t1)) #Print the length of tuple t1
print("Length of 2th element of tuple t2 is:",len(t1[2])) #Print the length of  2th element of t1
```

Output:

```
Length of tuple t1 is: 3
Length of 2th element of tuple t2 is: 8
```

8.

```
t1=('Hello','everyone','I','am','here')
print('am' in t1) #Check the membership of 'am' in tuple t1
```

Output:

```
True
```

14.5 Dictionaries

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces i.e { } and values can be assigned and accessed using square braces i.e []

Example:

1.

```
d={} #Empy dictionary
print(d)
```

Output:

{}

2.

```
d={'name':'amal','roll':1,'marks':76.5}
print("Keys of dictionary d are:")
print(d.keys())
```

Output:

Keys of dictionary d are:

dict_keys(['name', 'roll', 'marks'])

3.

```
d={'name':'amal','roll':1,'marks':76.5}
print("Values of dictionary d are:")
print(d.values())
```

Output:

Values of dictionary d are:

dict_values(['amal', 1, 76.5])

4.

```
d={'name':'amal','roll':1,'marks':76.5}
print("name is:",d['name']) #Accessing dictionary value by using the key 'name'
print("roll is:",d['roll']) #Accessing dictionary value by using the key 'roll'
print("marks is:",d['marks']) #Accessing dictionary value by using the key 'marks'
```

Output:

name is: amal

roll is: 1

marks is: 76.5

5.

```
player={} #Empty dictionary
player['name']="sachin" #Adding key and values to the dictionary
player['match']=10
print("name is:", player['name'] , "and match played are:" , player['match'])
```

Output:

name is: sachin and match played are: 10

6.

```
player={'name':'sachin','match':10}
player['runs']=500 #Adding new key and value to filled up dictionary
print("name is:", player['name'])
print("match played are:" , player['match'])
print("runs scored are:",player['runs'])
```

Output:

name is: sachin

match played are: 10

runs scored are: 500

7.

```
player={'name':'sachin','match':10,'runs':500}
player['match']=12 #Updating value related to key 'match'
player['runs']=650 #Updating value related to key 'runs'
print("name is:", player['name'])
print("Updated match played are:" , player['match'])
print("Updated runs scored are:",player['runs'])
```

Output:

name is: sachin

Updated match played are: 12

Updated runs scored are: 650

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement.

Example:

8.

```
player={'name':'sachin','match':10,'runs':500}
del player['name'] #Delete entry with key 'name'
print("After deleting name, dictionary is:")
print(player)
player.clear() #Delete all entries in dictionary
del player #Delete entire dictionary
```

Output:

```
After deleting name, dictionary is:
{'match': 10, 'runs': 500}
```

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

Example:

```
player={'name':'sachin','match':10,'runs':500,'match':12} #Assigning duplicate value with key 'match'
print("match is:",player['match'])
```

Output:

```
match is: 12
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Example:

```
player={['name']:'sachin','match':10,'runs':500,'match':12}
print("name is:",player['name'])
```

Output:

Error: unhashable type: 'list'

15Operator

Python language supports the following types of operators-

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

15.1Arithmetic Operators

15.1.1. Addition (+)

This operator adds values on either side of the operator.

Example:

```
a=200
b=100
c=a+b
print("Result of addition is =",c)
```

Output:

Result of addition is = 300

15.1.2. Subtraction (-)

This operator subtracts right hand operand from left hand operand.

Example:

```
a=200
b=100
c=a+b
print("Result of subtraction is =",c)
```

Output:

```
Result of subtraction is = 100
```

15.1.3. Multiplication (*)

This operator multiplies values on either side of the operator.

Example:

```
a=200
b=100
c=a*b
print("Result of multiplication is =",c)
```

Output:

```
Result of multiplication is =20000
```

15.1.4. Division (/)

This operator divides left hand operand by right hand operand.

Example:

1.

```
a=200
b=100
c=a/b
print("Result of division is =",c)
```

Output:

```
Result of division is is =2
```

2.

```
a=5
b=2
c=a/b
print("Result of division is =",c)
```

Output:

Result of division is = 2.5

3.

```
a=5.5
b=2.5
c=a/b
print("Result of division is =",c)
```

Output:

Result of division is = 2.2

15.1.5. Floor Division (//)

The operator performs division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity).

Example:

1.

```
a=5
b=2
c=a//b
print("Result of floor division is =",c)
```

Output:

Result of floor division is = 2

2.

```
a=5.5
b=2.5
c=a//b
print("Result of floor division is =",c)
```

Output:

Result of floor division is = 2.0

3.

```
a=-15
b=2.0
c=a//b
print("Result of floor division is =",c)
```

Output:

Result of floor division is = -8.0

4.

```
a=-15
b=-2.0
c=a//b
print("Result of floor division is =",c)
```

Output:

Result of floor division is = 7.0

15.1.6. Modulus (%)

This operator divides left hand operand by right hand operand and returns remainder.

Example:

```
a=14
b=3
c=a%b
print("Result of modulo operation is =",c)
```

Output:

Result of modulo operation is = 2

15.1.7. Exponent (**)

This operator performs exponential (power) calculation on operands.

Example:

```
a=3
b=2
```

```
c=a**b
print("Result of exponential operation is =",c)
```

Output:

```
Result of exponential operation is = 9
```

15.2.Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

15.2.1. Equality Checking (==)

If the values of two operands are equal, then the condition becomes true.

Example:

```
a=3
b=3
if(a==b):
    print("same")
else:
    print("not same")
```

Output:

```
Same
```

15.2.2. Not equals to Checking (!=)

Example:

```
a=3
b=5
if(a!=b):
    print(" not same")
else:
    print("same")
```

Output:

```
not same
```

15.2.3. Greater than Checking (>)

If the value of left operand is greater than the value of right operand, then condition becomes true.

Example:

```
a=32
b=5
if(a>b):
    print("greater")
else:
    print("smaller")
```

Output:

greater

15.2.4. Smaller than Checking (<)

If the value of left operand is less than the value of right operand, then condition becomes true.

Example:

```
a=3
b=5
if(a<b):
    print("smaller")
else:
    print("greater ")
```

Output:

smaller

15.2.5. Greater than or equals to Checking (>=)

If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.

Example:

```
a=5
b=5
if(a>=b):
```

```
    print("greater or equals to")
else:
    print("smaller")
```

Output:

greater or equals to

15.2.6. Smaller than or equals to Checking (<=)

If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

Example:

```
a=3
b=3
if(a<=b):
    print("smaller or equals to")
else:
    print("greater ")
```

Output:

smaller or equals to

15.3.Assignment Operators

These operators perform different kind of assignments.

15.3.1. Normal Assignment (=)

Assigns values from right side operands to left side operand.

Example:

```
b=10  #Assigning 10 into b
print("b=",b)
```

Output:

b=10

15.3.2. Add and Assign (+=)

It adds right operand to the left operand and assigns the result to left operand.

Example:

```
b=10
```

```
b+=5
print("b=",b)
```

Output:

```
b=15
```

15.3.3. Subtract and Assign (- =)

It subtracts right operand from the left operand and assigns the result to left operand.

Example:

```
b=10
b - =5
print("b=",b)
```

Output:

```
b=5
```

15.3.4. Multiply and Assign (* =)

It multiplies right operand with the left operand and assigns the result to left operand.

Example:

```
b=10
b*=5
print("b=",b)
```

Output:

```
b=50
```

15.3.5. Divide and Assign (/ =)

It divides left operand with the right operand and assigns the result to left operand.

Example:

```
b=15
b /=7
print("b=",b)
```

Output:

```
b= 2.142857142857143
```

15.3.6. Floor Divide and Assign (// =)

It performs floor division on operators and assigns value to the left operand.

Example:

```
b=15
b /=7
print("b=",b)
```

Output:

b= 2

15.3.7. Modulus and Assign (%=)

It takes modulus using two operands and assigns the result to left operand.

Example:

```
b=15
b %=7
print("b=",b)
```

Output:

b= 1

15.3.8. Exponent and Assign (**=)

Performs exponential (power) calculation on operators and assigns value to the left operand.

Example:

```
b=15
b**=2
print("b=",b)
```

Output:

b=225

15.4.Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation.

15.4.1. Bitwise or Binary AND (&)

Operator copies a bit to the result if it exists in both operands.

Example:

```
a=14
b=11
```

```
c=a&b
print("c=",c)
```

Output:

```
c=10
```

15.4.2. Bitwise or Binary OR (|)

It copies a bit if it exists in either operand.

Example:

```
a=14
b=11
c=a | b
print("c=",c)
```

Output:

```
c=15
```

15.4.3. Bitwise or Binary XOR (^)

It copies the bit if it is set in one operand but not both.

Example:

```
a=14
b=11
c=a^b
print("c=",c)
```

Output:

```
c=5
```

15.4.4. Bitwise or Binary Ones Complement (~)

It is unary and has the effect of 'flipping' bits.

Example:

1.

```
a=10
b=~a
print("b=",b)
```

Output:

```
b= -11
```

Explanation:

a = 10 = 0 0 0 0 1 0 1 0

1's Complement = *1* 1 1 1 0 1 0 1

Now, *1* at the MSB of 1's complement representation above (denoted by italicized), denotes the sign. Now, to represent the result of ~a, 2's complement of the 1's complement representation has to be done without the MSB. So, we perform 2's complement of 1110101.2's complement of 1110101 = 0001011=11.

After that, - sign is placed before 11, because of the MSB 1 in the 1's complement representation of 10.

So, the result of ~a= -11

2.

a= -11

b=~a

print("b=",b)

Output:

b= 10

Explanation:

a = -11 = 1000 1011

2's Complement = *0* 1 1 1 0 1 0 1

Now, 0 at the MSB of 2's complement representation above (denoted by italicized), denotes the sign. Now, to represent the result of ~a, 1's complement of the 2's complement representation has to be done without the MSB. So, we perform 1's complement of 1110101. 1's complement of 1110101 = 0001010=10.

After that, + sign is placed before 10, because of the MSB 0 in the 2's complement representation of -11.

So, the result of ~a= 10

15.4.5. Bitwise or Binary Left Shift (<<)

The left operands value is moved left by the number of bits specified by the right operand.

Example:

1.

a=15

```
a=a<<1
print("a=",a)
```

Output:

a=30

2.

```
a=15
a=a<<2
print("a=",a)
```

Output:

a=60

15.4.6. Bitwise or Binary Right Shift (>>)

The left operands value is moved right by the number of bits specified by the right operand.

Example:

1.

```
a=15
a=a>>1
print("a=",a)
```

Output:

a=7

2.

```
a=15
a=a>>2
print("a=",a)
```

Output:

a=3

15.5.Logical Operators

There are following logical operators supported by Python language.

15.5.1. Logical AND (and)

If both the operands are true then condition becomes true.

Example:

```
a=15
b=12
if (a>10 and b>10):
 print("Both a and b are greater than 10")
else:
 print("Not both a and b are greater than 10")
```

Output:

Both a and b are greater than 10

15.5.2. Logical OR (or)

If any of the two operands are non-zero then condition becomes true.

Example:

```
a=15
b=1
if (a>10 or b>10):
 print("At least one among a and b is greater than 10")
else:
 print("None among a or b is greater than 10")
```

Output:

At least one among a and b is greater than 10

15.5.3. Logical NOT (not)

Used to reverse the logical state of its operand.

Example:

```
a=15
if (not(a>10)):
 print("a is not greater than 10")
else:
 print("a is greater than 10")
```

Output:

a is greater than 10

15.6.Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists or tuples. There are two membership operators as explained below:

in

Evaluates to true if it finds a variable in the specified sequence and false otherwise.

Example:

1.

```
L1=[10,20,30,40]
print("Enter the element to be searched in the list")
e=(int)(input())
if(e in L1):
    print("Element is in the list L1")
else:
    print("Element is not in the list L1")
```

Output:

```
Enter the element to be searched in the list
20
Element is in the list L1
```

2.

```
T1=(1,2,3,4,5)
print("Enter the element to be searched in the tuple")
e=(int)(input())
if(e in T1):
    print("Element is in the tuple T1")
else:
    print("Element is not in the tuple T1")
```

Output:

```
Enter the element to be searched in the tuple
4
Element is in the tuple T1
```

3.

```
s1="hello everyone"
print("Enter the element to be searched in the string")
e=input()
if(e in s1):
    print("Element is in the string s1")
else:
    print("Element is not in the string s1")
```

Output:

```
Enter the element to be searched in the string
hello
Element is in the string s1
```

not in

Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

Example:

1.

```
L1=[10,20,30,40]
print("Enter the element to be searched in the list")
e=(int)(input())
if(e not in L1):
    print("Element is not in the list L1")
else:
    print("Element is in the list L1")
```

Output:

```
Enter the element to be searched in the list
50
Element is not in the list L1
```

2.

```
T1=(1,2,3,4,5)
print("Enter the element to be searched in the tuple")
e=(int)(input())
```

```
if(e not in T1):
    print("Element is not in the tuple T1")
else:
    print("Element is in the tuple T1")
```

Output:

```
Enter the element to be searched in the tuple
8
Element is not in the tuple T1
```

3.

```
s1="hello everyone"
print("Enter the element to be searched in the string")
e=input()
if(e not  in s1):
    print("Element is not in the string s1")
else:
    print("Element is in the string s1")
```

Output:

```
Enter the element to be searched in the string
hi
Element is not in the string s1
```

15.7.Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below:

is

Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.

Example:

1.

```
a=10
b=10
if( a is b ):
```

```
    print("same object")
else:
    print("different object")
```

Output:

same object

2.

```
a=10
b=20
if( a is b ):
    print("same object")
else:
    print("different object")
```

Output:

different object

is not

Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example:

1.

```
a=10
b=20
if( a is not b ):
    print("different object")
else:
    print("same object")
```

Output:

different object

2.

```
a=10
b=10
if( a is not b ):
```

```
    print("different object")
else:
    print("same object")
```

Output:

```
same object
```

16. Operators Precedence

The following table lists all operators from highest precedence to lowest

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

17.Conditional Statements

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. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

Python programming language provides following types of decision making statements:

if statement

The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Example:

1.

```
a=2 #Non zero value is true
b=0 #Zero value is false
if(a):
    print("a")
if(b):
    print("b")
```

Output:

a

2.

```
print("Enter the character")
c1=input()
a=ord(c1)#conver the chracter into its ASCII value
if(a>=65 and a<=90):
 print("Character",c1,"is upper case")
if(a>=97 and a<=122):
```

```
print("Character",c1,"is lower case")
```

Output:

Enter the character

B

Character B is upper case

if-else statement

An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The else statement is an optional statement and there could be at most only one else statement following if.

Example:

1.

```
a=2
b=0
if(a):
    print("true")
else:
    print("false")
if(b):
    print("true")
else:
    print("false")
```

Output:

true

false

2.

```
print("Enter the number")
n=(int)(input())
if(n%2==0): #Checking whether the number is even or not
```

```
    print("Entered number",n,"is even")
else:
    print("Entered number",n,"is odd")
```

Output:

```
Enter the number
12
Entered number 12 is even
```

3.

```
print("Enter the number")
n=(int)(input())
if(n>0): #Checking whether the number is positive or negative
    print("Entered number",n,"is positive")
else:
    print("Entered number",n,"is negative")
```

Output:

```
Enter the number
-8
Entered number -8 is negative
```

4.

```
print("Enter the number")
n=(int)(input())
if(n==1):
 print("Entered number",n,"is power of 2")
 exit(0)
a=n
b=n-1
c=a&b
if(c==0):
    print("Entered number",n,"is power of 2")
else:
    print("Entered number",n,"is not power of 2")
```

Output:

Enter the number

32

Entered number 32 is power of 2

5.

```
print("Enter two numbers")
n1=(int)(input())
n2=(int)(input())
if(n1>n2):
 print(n1,"is greater than",n2)
else:
    print(n2,"is greater than",n1)
```

Output:

Enter two numbers

12

10

12 is greater than 10

Nested if-else

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-else construct.

Example:

1.

```
print("Enter three numbers")
n1=(int)(input())
n2=(int)(input())
n3=(int)(input())
if(n1>n2):
    if(n1>n3):
        print(n1,"is greatest")
    else:
```

```
        print(n3,"is greatest")
else:
    if(n2>n3):
        print(n2,"is greatest")
    else:
        print(n3,"is greatest")
```

Output:

```
Enter three numbers
14
24
12
24 is greatest
```

2.

```
print("Enter the year")
y=(int)(input())

if(y%100==0):
    if(y%400==0):
        print(y,"is a leap year")
    else:
        print(y,"is not a leap year")
else:
    if(y%4==0):
        print(y,"is a leap year")
    else:
        print(y,"is not a leap year")
```

Output:

```
Enter the year
1996
1996 is a leap year
```

3.

```
import math
```

```
print("Enter three coefficients a,b,c of quadratic equation")
a=(int)(input())
b=(int)(input())
c=(int)(input())
d=pow(b,2)-4*a*c
if(d>0):
    s=math.sqrt(d)
    x1=(-b)/(2*a)+s/(2*a)
    x2=(-b)/(2*a)-s/(2*a)
    print(" Real roots are:x1=",x1,"and x2=",x2)
if(d==0):
    x1=(-b)/(2*a)
    print(" Real roots are:x=",x1)
else:
   x=(-b)/(2*a)
   s=math.sqrt(d*(-1))
   y=s/(2*a)
   print("Imaginary roots are:x1=",x," + i",y)
   print("Imaginary roots are:x2=",x," - i",y)
```

Output:

```
Enter three coefficients a,b,c of quadratic equation
2
2
3
Imaginary roots are:x1= -0.5  + i 1.118033988749895
Imaginary roots are:x2= -0.5  - i 1.118033988749895
```

- **Swap the value of two variable** :

```
1)A=5
  B=6
  A,B=B,A
  print(A,B)
2) A=5
   B=6
   A=A+B
   B=A-B
   A=A-B
   print(A,B)
```

3) A=5
   B=6
   A=A*B
   B=A/B
   A=A/B
   print(A,B)

- **Simple Method to use ternary operator:**

```
# Program to demonstrate conditional operator
a, b = 10, 20

# Copy value of a in min if a < b else copy b
min = a if a < b else b

print(min)
```