PYTHON

Introduction to python:

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Uses of python:

- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.
- Python can be used on a server to create web applications.

Running python

1. Interactive Mode:

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.

Ex 1: >>>”RGUKT” *5

O/P ‘RGUKT RGUKT RGUKT RGUKT RGUKT’

EX2:

```
$ python3
>>> 3+4
>>>7
```

2.Script Mode:

In script mode, You write your code in a text file then save it with a . py extension which stands for "Python".

PYTHON

EX1:

n1=4

n2=5

c=n1+n2

print (c)Values and types:

• int

• float

• complex

• bool

• str

Complex data type:

Integers, floating point numbers and complex numbers falls under Python numbers category.

They are defined as int , float and complex class in Python. ... 1 is integer, 1.0 is floating point number.

Complex numbers are written in the form, x + yj , where x is the real part and y is the imaginary part.

Ex:

```
>>>a=3+4j
>>>print(a)
o/p
3+4j
>>>type(a)
<class 'complex'>
```

Types:

Types means which type of data type can be assigned in value.

Like are, int, float,bool,complex,str.list.tuple,dict.

Data Types:

A data type or simply type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Python language supports the following datatypes.

PYTHON

1. Int
2. float
3. complex
4. bool
5. str

These are fundamaental data types.

6. List
7. tuple
8. set
9. dict

These are collection data types.

• Every data type i python language is internally implemented as a class.

• Python language data types are categorised into two types.

1. Fundamental Types.
2. Collection Types.

• Fundamental Types:

character, integer, float, and void are fundamental data types. Pointers,
arrays, structures and unions are derived data types. char, Signed char, Unsigned char. Pointers are used for storing address of variables.

• Collection Types:

Collection types are the common variations of data collections, such as hash
tables, queues, stacks, bags, dictionaries, and lists. Collections are based on the ICollection interface, the IList interface, the IDictionary interface, or their generic counterparts.

Python supports the following collection types.

1. List
2. tuple
3. set
4. dict

• List:

PYTHON

In computer science, a list or sequence is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once.

Ex:1

```
>>> l=[1,2,3,4]
>>>print(l)
[1.2.3.4]
type(l)
<class ‘list’
```

Ex:2

```
>>>l1=[‘a’,’b’,’c’,’d’]
>>>print(l1)
[‘a’,’b’,’c’,’d’]
type(l1)
<class’list’>
```

• Tuple:

Tuple is a new data type which includes two set of values of different data types. Consider the following example of number, string and tuple type variables. ... The same thing can be achieved by using a single tuple type variable. employee is the tuple type variable with two values of number and string type.

Ex: >>>tuple=(1,2,3,4,5,6)

```
>>>print (tuple)
(1,2,3,4,5,6)
>>>type (tuple)
<class ‘tuple’>
```

• Set:

A set is an abstract data type that can store unique values, without any particular order. It is a computer implementation of the mathematical concept of a finite set.

Ex:

```
>>> set={12,13,14,15}
```

PYTHON

```
print(set)
{12,13,14,15}
>>> type(set)
<class ‘set’>
```

• Dict:

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized.Ex:

```
>>>dict={1:”one”,2:”two”,3:”three”}
>>>print(dict)
{1:”one”,2:”two”,3:”three”}
type(dict)
<class.’dict’>
```

• int:

int (signed integers) − They are often called just integers or ints. They are positive or negative whole numbers with no decimal point. Integers in Python 3 are of unlimited size. Python 2 has two integer types - int and long.

Ex:

```
>>> type(10)
<class 'int'>
>>> type(0o10)
<class 'int'>
>>> type(0x10)
<class 'int'
```

• Float:

FLOAT(n) The FLOAT data type stores double-precision foating-point numbers with up to 17 significant digits. FLOAT corresponds to IEEE 4-byte foating-point, and to the double data type in C. The range of values for the FLOAT data type is the same as the range of the C double data type on your computer.

PYTHON

• Complex:

Some programming languages provide a complex data type for complex number storage and arithmetic as a built-in data type. In some programming environments the term complex data type is a synonym for the composite data type.

Ex:

```
include <stdio.h>
int main ()
{
int grades[5];
int i;
for (i=0; i<5; i++) {
}
printf("grades[%d]=%d\n", i, grades[i]);
}
return 0;
```

• Bool:

In computer science, the Boolean data type is a data type that has one of two possible values which is intended to represent the two truthvalues of logic and Boolean algebra. It is named after George Boole, who first defined an algebraic system of logic in the mid 19th century.

• Str:

A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class. Strings in Python can be created using single quotes or double quotes or even triple quotes.

PYTHON

➢ TYPES OF ERRORS:

The most common reason of an error in a Python program is when a certain statement is not in accordance with the prescribed usage. Such an error is called a syntax error. The Python interpreter immediately reports it, usually along with the reason.

```
>>> print "hello"
SyntaxError: Missing parentheses in call to 'print'. Did you mean
print("hello")?
```

In Python 3.x, print is a built-in function and requires parentheses. The statement above violates this usage and hence syntax error is displayed.

Many times though, a program results in an error after it is run even if it doesn't have any syntax error. Such an error is a runtime error, called an exception. A number of built-in exceptions are defined in the Python library. Let's see some common error types.

IndexError is thrown when trying to access an item at an invalid index.

```
>>> L1=[1,2,3]
>>> L1[3]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>L1[3]
IndexError: list index out of range
```

ModuleNotFoundError is thrown when a module could not be found.

```
>>> import notamodule
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
import notamodule
ModuleNotFoundError: No module named 'notamodule'
```

KeyError is thrown when a key is not found.

```
>>> D1={'1':"aa", '2':"bb", '3':"cc"}
>>> D1['4']
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
```

PYTHON

```
D1['4']
KeyError: '4'
```

ImportError is thrown when a specified function can not be found.

```
>>> from math import cube
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module>
from math import cube
ImportError: cannot import name 'cube'
```

StopIteration is thrown when the `next()` function goes beyond the iterator items.

```
>>> it=iter([1,2,3])
>>> next(it)
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
next(it)
StopIteration
```

TypeError is thrown when an operation or function is applied to an object of an inappropriate type.

```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

ValueError is thrown when a function's argument is of an inappropriate type.

```
>>> int('xyz')
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
int('xyz')
ValueError: invalid literal for int() with base 10: 'xyz'
```

NameError is thrown when an object could not be found.

PYTHON

```
>>> age
Traceback (most recent call last):
 File "<pyshell#6>", line 1, in <module>
 age
NameError: name 'age' is not defined
```

ZeroDivisionError is thrown when the second operator in the division is zero.

```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
x=100/0
ZeroDivisionError: division by zero
```

KeyboardInterrupt is thrown when the user hits the interrupt key (normally Control-C) during the execution of the program.

```
>>> name=input('enter your name')
enter your name^c
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
name=input('enter your name')
KeyboardInterrupt
```

The following table lists important built-in exceptions in Python.

Exception	Description
AssertionError	Raised when the assert statement fails.
AttributeError	Raised on the attribute assignment or reference fails.
EOFError	Raised when the input() function hits the end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in the local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when a system operation causes a system-related error.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.

PYTHON

Exception	Description
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
SyntaxError	Raised by the parser when a syntax error is encountered.
IndentationError	Raised when there is an incorrect indentation.
TabError	Raised when the indentation consists of inconsistent tabs and spaces.
SystemError	Raised when the interpreter detects internal error.
SystemExit	Raised by the sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of an incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of a division or module operation is zero

➢ OPERATORS AND OPERANDS IN PYTHON :

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the **operand**.

For example:

```
>>> 2+35
```

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

➢ TYPES OF OPERATORS:

1. ARITHMETIC OPERATORS
2. COMPARISION OPERATORS
3. LOGICAL OPERATORS
4. BITWISE OPERATORS
5. ASSIGNMENT OPERATORS
6. SPECIAL OPERATORS

PYTHON

7. MEMBERSHIP OPERATORS

◆ ARITHMETIC OPERATORS:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Example #1: Arithmetic operators in Python

```
x = 15y = 4

# Output: x + y = 19

print('x + y =',x+y)

# Output: x - y = 11

print('x - y =',x-y)

# Output: x * y = 60

print('x * y =',x*y)

# Output: x / y = 3.75

print('x / y =',x/y)

# Output: x // y = 3

print('x // y =',x//y)
```

PYTHON

```
# Output: x ** y = 50625

print('x ** y =',x**y)
```

When you run the program, the output will be:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

◆ COMPARISION OPERATORS:

Comparison operators are used to compare values. It either returns True or False according to the condition.

Comparision operators in Python

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example #2: Comparison operators in Python

```
x = 10y = 12

# Output: x > y is False

print('x > y  is',x>y)

# Output: x < y is True

print('x < y  is',x<y)

# Output: x == y is False

print('x == y is',x==y)

# Output: x != y is True

print('x != y is',x!=y)
```

PYTHON

```
# Output: x >= y is False

print('x >= y is',x>=y)

# Output: x <= y is True

print('x <= y is',x<=y)
```

◆ LOGICAL OPERATORS:

Logical operators are the and, or, not operators.

Logical operators in Python

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example #3: Logical Operators in Python

```
x = Truey = False

# Output: x and y is False

print('x and y is',x and y)

# Output: x or y is True

print('x or y is',x or y)

# Output: not x is False

print('not x is',not x)
```

◆ BITWISE OPERATORS:

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

For example, 2 is `10` in binary and 7 is `111`.

In the table below: Let *x* = 10 (`0000 1010` in binary) and *y* = 4 (`0000 0100` in binary)

Bitwise operators in Python

Operator	Meaning	Example

PYTHON

&	Bitwise AND	x& y = 0 (`0000 0000`)
|	Bitwise OR	x | y = 14 (`0000 1110`)
~	Bitwise NOT	~x = -11 (`1111 0101`)
^	Bitwise XOR	x ^ y = 14 (`0000 1110`)
>>	Bitwise right shift	x>> 2 = 2 (`0000 0010`)
<<	Bitwise left shift	x<< 2 = 40 (`0010 1000`)

◆ ASSIGNMENT OPERATORS:

Assignment operators are used in Python to assign values to variables.

`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`.

Assignment operators in Python

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

◆ SPECIAL OPERATORS:

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

PYTHON

IDNETITY OPERATORS:

`is` and `is not` are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Identity operators in Python

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example #4: Identity operators in Python

```
X1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

# Output: False

print(x1 is not y1)

# Output: True

print(x2 is y2)

# Output: False

print(x3 is y3)
```

Here, we see that *x1* and *y1* are integers of same values, so they are equal as well as identical. Same is the case with *x2* and *y2* (strings).

But *x3* and *y3* are list. They are equal but not identical. It is because interpreter locates them separately in memory although they are equal.

MEMBERSHIP OPERATORS:

`in` and `not in` are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

PYTHON

In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example #5: Membership operators in Python

```
x = 'Hello world'

y = {1:'a',2:'b'}

# Output: True

print('H' in x)

# Output: True

print('hello' not in x)

# Output: True

print(1 in y)

# Output: False

print('a' in y)
```

Here, 'H' is in *x* but 'hello' is not present in *x* (remember, Python is case sensitive). Similary, 1 is key and 'a' is the value in dictionary *y*. Hence, 'a' in y returns False.

➢ MUTABLE AND IMMUTABLE VARIABLES

Every variable in python holds an instance of an object. There are two types of objects in python i.e. **Mutable** and **Immutable objects**. Whenever an object is instantiated, it is assigned a unique object id. The type of the object is defined at the runtime and it can't be changed afterwards. However, it's state can be changed if it is a mutable object.

To summarise the difference, mutable objects can change their state or contents and immutable objects can't change their state or content.

◆ Immutable Objects :

These are of in-built types like **int, float, bool, string, unicode, tuple**. In simple words, an immutable object can't be changed after it is created.

PYTHON

```
# Python code to test that
# tuples are immutable
   tuple1 = (0, 1, 2, 3)
   tuple1[0] = 4
   print(tuple1)
```

Mutable Objects :

These are of type **list, dict, set** . Custom classes are generally mutable.

```
# Python code to test that
# lists are mutable
   color = ["red", "blue", "green"]
   print(color)
   color[0] = "pink"
```

Output:

```
['red', 'blue', 'green']
['pink', 'blue', 'orange']
```

- **Conclusion**

- Mutable and immutable objects are handled differently in python. Immutable objects are quicker to access and are expensive to **change** because it involves the creation of a copy.
  Whereas mutable objects are easy to change.
- Use of mutable objects is recommended when there is a need to change the size or content of the object.
- **Exception :** However, there is an exception in immutability as well. We know that tuple in python is immutable. But the tuple consists of a sequence of names with unchangeable bindings to objects.
  Consider a tuple

```
tup = ([3, 4, 5], 'myname')
```

- The tuple consists of a string and a list. Strings are immutable so we can't change its value. But the contents of the list can change. **The tuple itself isn't mutable but contain items that are mutable.**

INPUT AND OUTPUT STATEMENTS:

Python has two functions designed for accepting data directly from the user:

- `input()`
- `raw_input()`

- **INPUT:**

PYTHON

- **raw_input()**

raw_input() asks the user for a string of data (ended with a newline), and simply returns the string. It can also take an argument, which is displayed as a prompt before the user enters the data. E.g.

```
print raw_input('What is your name? ')
```

prints output:

What is your name? <user input data here>

- **OUTPUT:**

The basic way to do output is the print statement.

print 'Hello, world'

To print multiple things on the same line separated by spaces, use commas between them, like this:

print 'Hello,', 'World'

This will print out the following:

Hello, World

While neither string contained a space, a space was added by the print statement because of the comma between the two objects. Arbitrary data types can be printed this way:

print 1,2,0xff,0777,(10+5j),-0.999

This will output the following:

1 2 255 511 (10+5j) -0.999