PYTHON PROGRAMMING

Presented By

POTU NARAYANA

Research Scholar

Osmania University

Dept. of CSE

B.Tech, M.Tech, M.A, PGDHR, MIAENG, MIS,(Ph.D)

Email : pnarayana@osmania.ac.in

Contact : +91 9704868721

Python Language Introduction

- Python is a widely used general-purpose, high level programming language.

•Python was developed by Guido Van Rossam in 1989 while working at National Research Institute at Netherlands.

•But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.

•The name Python was selected from the TV Show

"The Complete Monty Python's Circus", which was broadcasted in BBC from 1969 to 1974. Guido developed Python language by taking almost all programming features from different languages –

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script
4. Modular Programming Features from Modula-3

Most of syntax in Python Derived from C and ABC languages.

- It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

- Python is a programming language that lets you work quickly and integrate systems more efficiently.

- There are two major Python versions- Python 2 and Python 3. Both are quite different.

Features of Python

1) Simple and easy to learn:

•Python is a simple programming language. When we read Python program, we can feel like reading English statements.
• The syntaxes are very simple and only 30+ keywords are available.
• When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity.
• We can reduce development and cost of the project.

2) Freeware and Open Source:

• We can use Python software without any license and it is freeware.
• Its source code is open, so that we can we can customize based on our requirement.

3) High Level Programming language:

• Python is high level programming language and hence it is programmer friendly language.

•Being a programmer we are not required to concentrate low level activities like memory management and security etc.

4) Platform Independent:

•Once we write a Python program, it can run on any platform without rewriting once again.

•Internally PVM is responsible to convert into machine understandable form.

5) Portability:

•Python programs are portable. i.e. we can migrate from one platform to another platform very easily. Python programs will provide same results on any platform.

6)Dynamically Typed:

•In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language.

•But Java, C etc are Statically Typed Languages b'z we have to provide type at the beginning only.

•This dynamic typing nature will provide more flexibility to the programmer.

7) Both Procedure Oriented and Object Oriented:

•Python language supports both Procedure oriented (like C, Pascal etc) and object oriented (like C++, Java) features. Hence we can get benefits of both like security and reusability etc

8) Interpreted:

•We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation.

•If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

9) Extensible:

We can use other language programs in Python.
The main advantages of this approach are:
We can use already existing legacy non-Python code
We can improve performance of the application

10) Embedded:

We can use Python programs in any other language programs.
i.e we can embed Python programs anywhere.

11) Extensive Library:

- Python has a rich inbuilt library.
- Being a programmer we can use this library directly and we are not responsible to implement the functionality. Etc

Flavors of Python:

1) CPython: It is the standard flavor of Python. It can be used to work with C language Applications.
2) Jython OR JPython: It is for Java Applications. It can run on JVM
3) IronPython: It is for C#.Net platform
4) PyPy: The main advantage of PyPy is performance will be improved because JIT compiler is available inside PVM.
5) RubyPython : For Ruby Platforms
6) AnacondaPython : It is specially designed for handling large volume of data processing.

Python Versions:

•Python 1.0V introduced in Jan 1994
•Python 2.0V introduced in October 2000
•Python 3.0V introduced in December 2008

Note: Python 3 won't provide backward compatibility to Python2 i.e there is no guarantee that Python2 programs will run in Python3.
Current versions
Python 3.6.1 Python 2.7.13

Limitations of Python

- Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).
- Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.
- As compared to other languages, Python is evolving continuously and there is little substantial documentation available for the language.
- As of now, there are few users of Python as compared to those using C, C++ or Java.
- It has very limited commercial support point.
- Python is slower than C or C++ when it comes to computation heavy tasks and desktop applications.
- It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical.

Applications of Python

- **Embedded scripting language:** Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
- **3D Software:** 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
- **Web development:** Python is an easily extensible language that provides good integration with database and other web standards.

GUI-based desktop applications: Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.

- ***Image processing and graphic design applications:*** Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Lightwave and Maya.

Applications of Python

- ***Scientific and computational applications:*** Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.

Games: Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.

- ***Enterprise and business applications:*** Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.
- ***Operating Systems:*** Python forms an integral part of Linux distributions.

To print Hello world

JAVA	C	Python

1) public class Hello world
2) {
3) p s v main(String[] args)
4) {
5) SOP("Hello world");
6) }
7) }
	1) #include<stdio.h>
	2) void main()
	3) {
	4) print("Hello world");
	5) }
		print("Hello World")

print the sum of 2 numbers

JAVA	C	Python:

1) public class Add
2) {
3) public static void main(String[] args)
4) {
5) int a,b;
6) a =10;
7) b=20;
8) System.out.println("The Sum:"+(a+b));
9) }
10) }
	1) #include <stdio.h>
	2) void main()
	3) {
	4) int a,b;
	5) a =10;
	6) b=20;
	7) printf("The sum:%d",(a+b));
	9) }
		1) a=10
		2) b=20
		3) print("The Sum:",(a+b))

Python Objects

- Python uses the object model abstraction for data storage. Any construct that contains any type of value is an object. Although Python is classified as an "object-oriented programming (OOP) language," OOP is not required to create perfectly working Python applications.

- You can certainly write a useful Python script without the use of classes and instances. However, Python's object syntax and architecture encourage or "provoke" this type of behavior.

- All Python objects have the following three characteristics:

  an *identity, a type, and a value.*

- IDENTITY : Unique identifier that differentiates an object from all others. Any object's identifier can be obtained using the id() built-in function (BIF). This value is as close as you will get to a "memory address" in Python (probably much to the relief of some of you). Even better is that you rarely, if ever, access this value, much less care what it is at all.
- TYPE : An object's type indicates what kind of values an object can hold, what operations can be applied to such objects, and what behavioral rules these objects are subject to. You can use the type() BIF to reveal the type of a Python object. Since types are also objects in Python (did we mention that Python was object-oriented?), type() actually returns an object to you rather than a simple literal.
- VALUE : Data item that is represented by an object.

All three are assigned on object creation and are read-only with one exception, the value. If an object supports updates, its value can be changed; otherwise, it is also read-only. Whether an object's value can be changed is known as an object's *mutability.*

Object Attributes:

Certain Python objects have attributes, data values or executable code such as methods, associated with them. Attributes are accessed in the dotted attribute notation, which includes the name of the associated object The most familiar attributes are functions and methods, but some Python types have data attributes associated with them. Objects with data attributes include (but are not limited to): classes, class instances, modules, complex numbers, and files.

IDENTIFIER

- A Name in Python Program is called Identifier.

It can be Class Name OR Function Name OR Module Name OR Variable Name.

- a = 10

Rules to define Identifiers in Python:

The only allowed characters in Python are

alphabet symbols(either lower case or upper case)

digits(0 to 9)

underscore symbol(_)

- By mistake if we are using any other symbol like $ then we will get syntax error.

cash = 10 √

ca$h =20

Identifier should not starts with digit

- 123total
- total123 √

Identifiers are case sensitive. Of course Python language is case sensitive language.

- total=10
- TOTAL=999
- print(total) #10
- print(TOTAL) #999

- **Identifier:**
- **1) Alphabet Symbols (Either Upper case OR Lower case)**
- **2) If Identifier is start with Underscore (_) then it indicates it is private.**
- **3) Identifier should not start with Digits.**
- **4) Identifiers are case sensitive.**
- **5) We cannot use reserved words as identifiers**
- **Eg: def = 10**
- **6) There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.**
- **7) Dollor ($) Symbol is not allowed in Python.**

Reserved Words

- In Python some words are reserved to represent some meaning or functionality. Such types of words are called reserved words.
- There are 33 reserved words available in Python.
  - True, False, None
  - and, or ,not,is
  - if, elif, else
  - while, for, break, continue, return, in, yield
  - try, except, finally, raise, assert
  - import, from, as, class, def, pass, global, nonlocal, lambda, del, with
  - You can observe these using

```
import keyword
keyword.kwlist
```

- Note:
  - 1. All Reserved words in Python contain only alphabet symbols.
  - 2. Except the following 3 reserved words, all contain only lower case alphabet symbols.
    - True
    - False
    - None

Data Types

- Data Type represents the type of data present inside a variable.
- In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically. Hence Python is dynamically Typed Language.
- Python contains the following standard data types
  - 1) Int
  - 2) Float
  - 3) Complex
  - 4) Bool
  - 5) Str
  - 6) List
  - 7) Tuple
  - 8) Dict
  - 6) Bytes
  - 7) Bytearray
  - 8) Range

Other Built-in types in Python

- 1) Type
- 2) None
- 3) File

4) Set/Frozenset
5) Function / Method
6) Module

7) Class

- Python contains several inbuilt functions
- to check the type of variable
  - 1) type()
- to get address of object
  - 2) id()
- to print the value
  - 3) print()
- In Python everything is an Object.

int Data Type

- We can use int data type to represent whole numbers (integral values)
  - Eg: a = 10
  - type(a) #int
- Note:
  - In Python2 we have long data type to represent very large integral values.
  - But in Python3 there is no long type explicitly and we can represent long values also by using int type only.
- We can represent int values in the following ways
  - 1) Decimal form
  - 2) Binary form
  - 3) Octal form
  - 4) Hexa decimal form

- **Decimal Form (Base-10):**
  - It is the default number system in Python
  - The allowed digits are: 0 to 9
  - Eg: a =10
- **Binary Form (Base-2):**
  - The allowed digits are : 0 & 1
  - Literal value should be prefixed with 0b or 0B
  - Eg: a = 0B1111
  - a = 0B123
  - a = b111
- **Octal Form (Base-8):**
  - The allowed digits are : 0 to 7
  - Literal value should be prefixed with 0o or 0O.
  - Eg: a = 0o123
  - a = 0o786
- **Hexa Decimal Form (Base-16):**
  - The allowed digits are: 0 to 9, a-f (both lower and upper cases are allowed)
  - Literal value should be prefixed with 0x or 0X
  - Eg: a = 0XFACE
  - a = 0XBeef
  - a = 0XBeer

Examples

- a=10
- b=0o10
- c=0X10
- d=0B10
- print(a)10
- print(b)8
- print(c)16
- print(d)2

Float Data Type

- We can use float data type to represent floating point values (decimal values)
  - Eg: f = 1.234
  - type(f) float
- We can also represent floating point values by using exponential form (Scientific Notation)
  - Eg: f = 1.2e3 instead of 'e' we can use 'E'
  - print(f) 1200.0
- The main advantage of exponential form is we can represent big values in less memory.

- Note:
  - We can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.

Complex Data Type

- A complex number is of the form
  - a + bj
  - a is Real Part and bj is Imaginary Part
  - a’ and ‘b’ contain Integers OR Floating Point Values
  - Eg: 3 + 5j
  - 10 + 5.5j
  - 0.5 + 0.1j
- In the real part if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form.
- But imaginary part should be specified only by using decimal form.
  - 1) >>> a=0B11+5j
  - 2) >>> a
  - 3) (3+5j)
  - 4) >>> a=3+0B11j
  - 5) SyntaxError: invalid syntax
- Even we can perform operations on complex type values.
  - 1) >>> a=10+1.5j
  - 2) >>> b=20+2.5j
  - 3) >>> c=a+b
  - 4) >>> print(c)
  - 5) (30+4j)
  - 6)>>> type(c)
  - 7) <class 'complex'>

- Note:
  - Complex data type has some inbuilt attributes to retrieve the real part and imaginary part
  - c = 10.5+3.6j
  - c.real - 10.5
  - c.imag - 3.6
- We can use complex type generally in scientific Applications and electrical engineering Applications.

Boolean Data Type:

- We can use this data type to represent Boolean values.
- The only allowed values for this data type are:
- True and False
- Internally Python represents True as 1 and False as 0
  - b = True
  - type(b) - bool

  Eg:
    - a = 10
    - b = 20
    - c = a<b
    - print(c) - True
    - True+True - 2
    - True-False - 1

str Data Type

- str represents String data type.
- A String is a sequence of characters enclosed within single quotes or double quotes.
  - s1='python'
  - s1=" python "
- Slicing of Strings:
  - 1) slice means a piece
  - 2) [ ] operator is called slice operator, which can be used to retrieve parts of String.
  - 3) In Python Strings follows zero based index.
  - 4) The index can be either +ve or -ve.
  - 5) +ve index means forward direction from Left to Right
  - 6) -ve index means backward direction from Right to Left

-6	-5	-4	-3	-2	-1
p	y	t	h	o	n
0	1	2	3	4	5

```
S=python
>>> s[0]    --- p
>>> s[-1]   ---- n
>>> s[10] --- IndexError: string index out of range
```

- >>> S[1:40] --- ython
- >>> s[:1] --- ython
- >>> s[:4] --- pyth
- >>> s[:] --- python
- >>> s[*3] --- pythonpythonpython
- >>> len[s] --- 6

Note:

- 1) In Python the following data types are considered as Fundamental Data types
  - int
  - float
  - complex
  - bool
  - str
- 2) In Python, we can represent char values also by using str type and explicitly char type is not available.
  - 1) >>> c='a'
  - 2) >>> type(c)
  - 3) <class 'str'>
- 3) long Data Type is available in Python2 but not in Python3. In Python3 long values also we can represent by using int type only.
- 4) In Python we can present char Value also by using str Type and explicitly char Type is not available.

bytes Data Type

- bytes data type represents a group of byte numbers just like an array
  - >>> x = [10,20,30,40]
  - >>> b = bytes(x)
  - >>> type(b) -- bytes
  - >>> print(b[0]) -- 10
  - >>> print(b[-1]) -- 40
  - >>> for i in b : print(i)

```
10
20
30
40
```

Note 1 : **The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.**

2. Once we creates bytes data type value, we cannot change its values, otherwise we will get TypeError .

Eg:
```
>>> x=[10,20,30,40]
>>> b=bytes(x)
>>> b[0]=100
TypeError: 'bytes' object does not support item assignment
```

bytearray Data Type

- bytearray is exactly same as bytes data type except that its elements can be modified.

Eg 1

- x=[10,20,30,40]
- b = bytearray(x)
- for i in b : print(i)
  - 10
  - 20
  - 30
  - 40
- b[0]=100
- for i in b: print(i)
  - 100
  - 20
  - 30
  - 40

  - Eg 2:
- >>> x =[10,256]
- >>> b = bytearray(x)
- ValueError: byte must be in range(0, 256)

Range Data Type

- range Data Type represents a sequence of numbers.
- The elements present in range Data type are not modifiable. i.e range Data type is immutable.

Form-1:

range(10)

generate numbers from 0 to 9

Eg:

```
r = range(10)
for i in r : print(i) --  0 to 9
```

Form-2: range(10, 20)

generate numbers from 10 to 19

Eg:

```
r = range(10,20)
for i in r : print(i) --- 10 to 19
```

Form-3: range(10, 20, 2)

2 means increment value

Eg:

```
r = range(10,20,2)
for i in r : print(i) ---  10,12,14,16,18
```

- We can access elements present in the range Data Type by using index.
- Eg:
  - r = range(10,20)
  - r[0] -- 10
  - r[15] -- IndexError: range object index out of range
  - We cannot modify the values of range data type
- Eg:
  - r[0] = 100
  - TypeError: 'range' object does not support item assignment

List Data Type

- If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.
  - 1) Insertion Order is preserved
  - 2) Heterogeneous Objects are allowed
  - 3) Duplicates are allowed
  - 4) Growable in nature
  - 5) Values should be enclosed within square brackets.

  Eg:
  - list=[10,10.5,”python”,True,10]
  - print(list)

Eg:

```
list=[10,20,30,40]

 >>> list[0] ---  10

>>> list[-1]  ---  40

>>> list[1:3] --- [20, 30]

>>> list[0]=100

>>> for i in list:print(i)

 ...

100

20

30

40
```

- list is growable in nature. i.e based on our requirement we can increase or decrease the size.

- >>> list=[10,20,30]
- >>> list.append(“python")
- >>> list
- [10, 20, 30, ’python']
- >>> list.remove(20)
- >>> list
- [10, 30, ‘python']
- >>> list2=list*2
- >>> list2
- [10, 30, ‘python’, 10, 30, 'python’']

- **Note: An ordered, mutable, heterogeneous collection of elements is nothing but list, where duplicates also allowed.**

Tuple Data Type

- tuple data type is exactly same as list data type except that it is immutable.i.e we cannot change values.
- Tuple elements can be represented within parenthesis
- Eg:
  - 1) t=(10,20,30,40)
  - 2) type(t)
  - 3) <class 'tuple'>
  - 4) t[0]=100
  - 5) TypeError: 'tuple' object does not support item assignment
  - 6) >>> t.append("durga")
  - 7) AttributeError: 'tuple' object has no attribute 'append'
  - 8) >>> t.remove(10)
  - 9) AttributeError: 'tuple' object has no attribute 'remove'

- Note: tuple is the read only version of list

dict Data Type

- If we want to represent a group of values as key-value pairs then we should go for dict data type.
- Eg: d = {101:'durga',102:'ravi',103:'shiva'}
- Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value
- Eg:
  - >>> d={101:'durga',102:'ravi',103:'shiva'}
  - >>> d[101]='sunny'
  - >>> d
  - {101: 'sunny', 102: 'ravi', 103: 'shiva'}

We can create empty dictionary as follows

```
d={ }
```

We can add key-value pairs as follows

```
d['a']='apple'
d['b']='banana'
print(d)
```

- Note: dict is mutable and the order won't be preserved.
- Note:
- 1) In general we can use bytes and bytearray data types to represent binary information like images, video files etc
- 2) In Python2 long data type is available. But in Python3 it is not available and we can represent long values also by using int type only.
- 3) In Python there is no char data type. Hence we can represent char values also by using str type.

set Data Type

- we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

  1)Insertion order is not preserved

  2) Duplicates are not allowed

  3) Heterogeneous objects are allowed

  4) Index concept is not applicable

  5) It is mutable collection

  6) Growable in nature
- Eg:
  - s={100,0,10,200,10,’python'}
  - s # {0, 100, ‘python', 200, 10}
  - s[0] -- TypeError: 'set' object does not support indexing
- set is growable in nature, based on our requirement we can increase or decrease the size.
- >>> s.add(60)
- >>> s
- {0, 100, ‘python', 200, 10, 60}
- >>> s.remove(100)
- >>> s
- {0, ‘python', 200, 10, 60}

frozenset Data Type

- It is exactly same as set except that it is immutable.
- Hence we cannot use add or remove functions.

```
>>> s={10,20,30,40}
>>> fs=frozenset(s)
>>> type(fs)
<class 'frozenset'>
>>> fs
frozenset({40, 10, 20, 30})
>>> for i in fs:print(i)
...
40
10
20
30

>>> fs.add(70)
AttributeError: 'frozenset' object has no attribute 'add'
>>> fs.remove(10)
AttributeError: 'frozenset' object has no attribute 'remove'
```

None Data Type

- None means nothing or No value associated.

  If the value is not available, then to handle such type of cases None introduced.

  It is something like null value in Java.

Eg:

```
def m1():
a=10
print(m1())
None
```

Internal Types

- Code
- Frame
- Traceback
- Slice
- Ellipsis
- Xrange

Code Objects

- Code objects are executable pieces of Python source that are byte-compiled, usually as return values from calling the compile() Code objects themselves do not contain any information regarding their execution environment, but they are at the heart of every user-defined function, all of which *do contain some execution context.*

Frame

- These are objects representing execution stack frames in Python. Frame objects contain all the information the Python interpreter needs to know during a runtime execution environment. Some of its attributes include a link to the previous stack frame, the code object (see above) that is being executed, dictionaries for the local and global namespaces, and the current instruction. Each function call results in a new frame object, and for each frame object, a C stack frame is created as well. One place where you can access a frame object is in a traceback object

Traceback Objects

- When you make an error in Python, an exception is raised. If exceptions are not caught or "handled," the interpreter exits with some diagnostic information similar to the output shown below:
  - Traceback (innermost last):
  - File "<stdin>", line N?, in ???
  - ErrorName: error reason
- The traceback object is just a data item that holds the stack trace information for an exception and is created when an exception occurs.
- If a handler is provided for an exception, this handler is given access

  to the traceback object.

Slice Objects

- Slice objects are created using the Python extended slice syntax. This extended syntax allows for different types of indexing. These various types of indexing include *stride indexing, multi-dimensional* indexing, and indexing using the Ellipsis type. The syntax for multi-dimensional indexing is *sequence*

  [start1 : end1, start2 : end2], or using the ellipsis, sequence [..., start1 : end1].

  Slice objects can also be generated by the slice() BIF.
- Stride indexing for sequence types allows for a third slice element that allows for "step"-like access with a syntax of *sequence[starting_index : ending_index : stride].*

- Ellipsis Objects
  - Ellipsis objects are used in extended slice notations as demonstrated above. These objects are used to represent the actual ellipses in the slice syntax (...). Like the Null object None, ellipsis objects also have a single name, Ellipsis, and have a Boolean True value at all times.
- XRange Objects
  - XRange objects are created by the BIF xrange(), a sibling of the range() BIF, and used when memory is limited and when range() generates an unusually large data set. You can find out more about range() and xrange()

Python Operators

- The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.
  - Arithmetic operators : (+ , - , *, /, %, //(floor division), **(exponent))
  - Comparison operators : (==, !=, <=, >=, >, <)
  - Assignment Operators : (=, +=, -=, *=, %=, **=, //=)
  - Logical Operators : and , or , not
  - Bitwise Operators : (&, !, ^, ~, <<, >>)
  - Membership Operators : in , not in
  - Identity Operators : is , is not

Assignment Operators

Eg:

a = 10.5

b=2

a+b= 12.5

a-b= 8.5

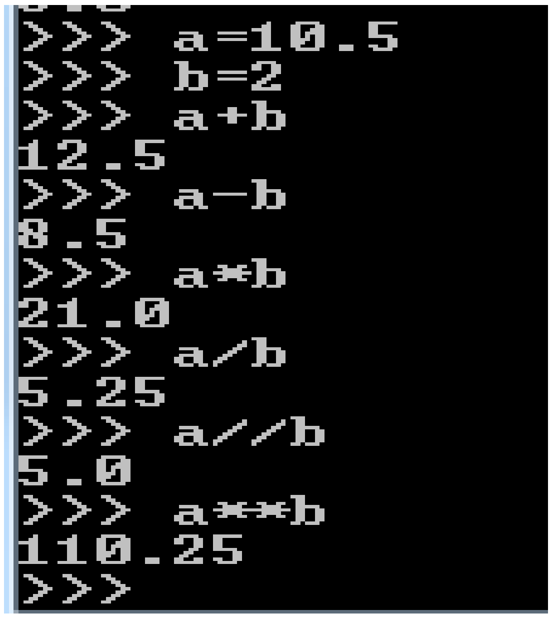
a*b= 21.0

a/b= 5.25

a//b= 5.0

a%b= 0.5

ab= 110.25**



```
>>> a=10.5
>>> b=2
>>> a+b
12.5
>>> a-b
8.5
>>> a*b
21.0
>>> a/b
5.25
>>> a//b
5.0
>>> a**b
110.25
>>>
```

Comparison Operators

- Eg:

```
a=10
b=20
print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is ",a<b)
print("a <= b is ",a<=b)
Print(“a==b is “,a==b)
Print(“a!=b is “,a!=b)
```

Assignment Operators

Eg:

```
x=10
x+=20
print(x) -- 30
```

Eg:

```
x=10
x&=5
print(x) -- 0
```

Logical Operators

- For boolean Types Behavior:

  and - If both arguments are True then only result is True

  or - If at least one argument is True then result is True

  Not - Complement
- True and False = False
- True or False = True
- not False = True
- For non-boolean Types Behaviour:
  - 0 means False
  - non-zero means True
  - empty string is always treated as False
- x and y:

  If x is evaluates to false return x otherwise return y

  Eg:

  10 and 20 =20

  0 and 20 = 0
- If first argument is zero then result is zero otherwise result is y
- x or y:
  - If x evaluates to True then result is x otherwise result is y
  - 10 or 20 = 10
  - 0 or 20 = 20

- **not x:**

  If x is evaluates to False then result is True otherwise False

  not 10 = False

  not 0 = True

Bitwise Operators

- These operators are applicable only for int and boolean types.
- By mistake if we are trying to apply for any other type then we will get Error.
- &, |, ^, ~, <<, >>
- print(4&5) -- Valid
- print(10.5 & 5.6) -- Type Error: unsupported operand type(s) for &: 'float' and 'float'
- print(True & True) -- Valid
- & → If both bits are 1 then only result is 1 otherwise result is 0
- |→ If at least one bit is 1 then result is 1 otherwise result is 0
- ^ → If bits are different then only result is 1 otherwise result is 0
- ~ → bitwise complement operator
- 1 → 0 & 0 → 1
- << → Bitwise Left Shift
- >> → Bitwise Right Shift
- print(4&5) -- 4
- print(4|5) --5
- print(4^5) -- 1

Operator Description

- &	If both bits are 1 then only result is 1 otherwise result is 0
- |	If atleast one bit is 1 then result is 1 otherwise result is 0
- ^	If bits are different then only result is 1 otherwise result is 0
- ~	bitwise complement operator i.e 1 means 0 and 0 means 1
- >>	Bitwise Left shift Operator
- <<	Bitwise Right shift Operator

- Bitwise Complement Operator (~):

  We have to apply complement for total bits.

  Eg: print(~5) → -6

Note:

The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value.

ositive numbers will be repesented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form

Membership Operators

- We can use Membership operators to check whether the given object present in the given collection. (It may be String, List, Set, Tuple OR Dict)
- In → Returns True if the given object present in the specified Collection
- not in → Returns True if the given object not present in the specified Collection
- Eg:
  - 1) x="hello learning Python is very easy!!!"
  - 2) print('h' in x) True
  - 3) print('d' in x) False
  - 4) print('d' not in x) True
  - 5) print('Python' in x) True
- Eg:
  - 1) list1=["sunny","bunny","chinny","pinny"]
  - 2) print("sunny" in list1) True
  - 3) print("tunny" in list1) False
  - 4) print("tunny" not in list1) True

Identity Operators

- We can use identity operators for address comparison.
- There are 2 identity operators are available
  - 1) is
  - 2) is not
- r1 is r2 returns True if both r1 and r2 are pointing to the same object.
- r1 is not r2 returns True if both r1 and r2 are not pointing to the same object.
- Eg:
  - a=10
  - b=10
  - print(a is b) True
  - x=True
  - y=True
  - print(x is y) True

- Eg:

```
a="durga"
b=“maata"
print(id(a))
print(id(b))
print(a is b)
```

- Eg:

```
list1=["one","two","three"]
list2=["one","two","three"]
print(id(list1))
print(id(list2))
print(list1 is list2) False
print(list1 is not list2) True
print(list1 == list2) True
```

- Note: We can use is operator for address comparison where as == operator for content comparison

Ternary Operator OR Conditional Operator

Syntax: x = firstValue if condition else secondValue

- If condition is True then firstValue will be considered else secondValue will be considered.
  Eg 1:
  - 1) a,b=10,20
  - 2) x=30 if a<b else 40
  - 3) print(x) #30
- Eg 2: Read two numbers from the keyboard and print minimum value
  - 1) a=int(input("Enter First Number:"))
  - 2) b=int(input("Enter Second Number:"))
  - 3) min=a if a<b else b
  - 4) print("Minimum Value:",min)

  Output:
  - Enter First Number:10
  - Enter Second Number:30
  - Minimum Value: 10
- Note: Nesting of Ternary Operator is Possible.
- Q) Program for Minimum of 3 Numbers
    - 1) a=int(input("Enter First Number:"))
    - 2) b=int(input("Enter Second Number:"))
    - 3) c=int(input("Enter Third Number:"))
    - 4) min=a if a<b and a<c else b if b<c else c
    - 5) print("Minimum Value:",min)

Standard Type Built-in Functions

- cmp() :It compares the elements of both tuples.

  If both tuples are equal then returns 0

  If the first tuple is less than second tuple then it returns -1

  If the first tuple is greater than second tuple then it returns +1

  - **t1=(10,20,30)**
  - **t2=(40,50,60)**
  - **t3=(10,20,30)**
  - **print(cmp(t1,t2)) → -1**
  - **print(cmp(t1,t3)) → 0**
  - **print(cmp(t2,t3)) → +1**

  Note: cmp() function is available only in Python2 but not in Python 3

- repr(obj) or `obj` Returns evaluatable string representation of *obj*

  The repr() method takes a single parameter:

  obj - object whose printable representation has to be returned

  var = 'foo'

  repr(var)

- str(obj) Returns printable string representation of *obj*

```
>>> str(10)
 '10'
>>> str(10.5)
'10.5'
>>> str(10+5j)
'(10+5j)'
>>> str(True)
 'True'
```

- type(obj) Determines type of *obj and return type object*

```
>>> a=10
>>> type(a)
```

Categorizing the Standard types

Data Type	*Storage Model*	*Update Model*	*Access Model*
Numbers	Scalar	Immutable	Direct
Strings	Scalar	Immutable	Sequence
Lists	Container	Mutable	Sequence
Tuples	Container	Immutable	Sequence
Dictionaries	Container	Mutable	Mapping

Unsupported Types

- **char or byte**

  Python does not have a char or byte type to hold either single character or 8-bit integers. Use strings of length one for characters and integers for 8-bit numbers.

- **pointer**

  Since Python manages memory for you, there is no need to access pointer addresses. The closest to an address that you can get in Python is by looking at an object's identity using the id() BIF. Since you have no control over this value, it's a moot point. However, under Python's covers, everything is a pointer.

Built in Functions

```
– >>> cmp(-6, 2)
– -1
– >>> cmp(-4.333333, -2.718281828)
– -1
– >>> cmp(0xFF, 255)
– 0
– >>> str(0xFF)
– '255'
– >>> str(55.3e2)
– '5530.0'
– >>> type(0xFF)
– <type 'int'>
– >>> type(98765432109876543210L)
– <type 'long'>
– >>> type(2-1j)
– <type 'complex'>
```

Numeric Type Functions

- >>> int(4.25555)
- 4
- >>> long(42)
- 42L
- >>> float(4)
- 4.0
- >>> complex(4)
- (4+0j)
- >>>
- >>> complex(2.4, -8)
- (2.4-8j)
- >>>
- >>> complex(2.3e-10, 45.3e4)
- (2.3e-10+453000j)

Operational Type Functions

- >>> abs(-1)
- 1
- >>> abs(10.)
- 10.0
- >>> abs(1.2-2.1j)
- 2.41867732449
- >>> abs(0.23 - 0.78)
- 0.55
- The divmod() built-in function combines division and modulus operations into a single function call that returns the pair (quotient, remainder) as a tuple. The values returned are the same as those given for the classic division and modulus operators for integer types. For floats, the quotient returned is math.floor(*num1/num2) and for complex numbers, the quotient is math.floor((num1/num2).real).*
- >>> divmod(10,3)
- (3, 1)
- >>> divmod(3,10)
- (0, 3)
- file:///D|/1/0132269937/ch05lev1sec6.html (3 von 8) [13.11.2007 16:23:00]
- Section 5.6. Built-in and Factory Functions
- >>> divmod(10,2.5)
- (4.0, 0.0)
- >>> divmod(2.5,10)
- (0.0, 2.5)
- >>> divmod(2+1j, 0.5-1j)
- (0j, (2+1j))

- Both pow() and the double star (**) operator perform exponentiation; however, there are differences other than the fact that one is an operator and the other is a built-in function.
- The ** operator did not appear until Python 1.5, and the pow() built-in takes an optional third parameter, a modulus argument. If provided, pow() will perform the exponentiation first, then return the result modulo the third argument. This feature is used for cryptographic applications and has better performance than pow(x,y) % z since the latter performs the calculations in Python rather than in C-like pow(x, y, z).
  - >>> pow(2,5)
  - 32
- The round() built-in function has a syntax of round(*flt,ndig=0). It normally rounds a floating point*
- number to the nearest integral number and returns that result (still) as a float. When the optional *ndig*
- option is given, round() will round the argument to the specific number of decimal places.
  - >>> round(3)
  - 3.0
  - >>> round(3.45)
  - 3.0
  - >>> round(3.4999999)
  - 3.0
  - >>> round(3.4999999, 1)
  - 3.5

Integer Only Functions

- >>> hex(255)
- '0xff'
- >>> hex(23094823l)
- '0x1606627L'
- >>> hex(65535*2)
- '0x1fffe'
- >>> oct(255)
- '0377'
- >>> oct(23094823l)
- '0130063047L'
- >>> oct(65535*2)
- '0377776'

ASCII Conversion

- Python also provides functions to go back and forth between ASCII (American Standard Code for Information Interchange) characters and their ordinal integer values. Each character is mapped to a unique number in a table numbered from 0 to 255. This number does not change for all computers using the ASCII table, providing consistency and expected program behavior across different systems. chr() takes a single-byte integer value and returns a one-character string with the equivalent ASCII character.
- ord() does the opposite, taking a single ASCII character in the form of a string of length one and returns the corresponding ASCII value as an integer:
- >>> ord('a')
- 97
- >>> ord('A')
- 65
- >>> ord('0')
- 48
- >>> chr(97)
- 'a'
- >>> chr(65L)
- 'A'

Related Modules

- There are a number of modules in the Python standard library that add on to the functionality of the operators and built-in functions for numeric types. Table 5.8 lists the key modules for use with numeric types. Refer to the literature or online documentation for more information on these modules.

Module	***Contents***
Decimal	Decimal floating point class Decimal
array	Efficient arrays of numeric values (characters, ints, floats, etc.)
math/cmath	Standard C library mathematical functions; most functions available in math are implemented for complex numbers in the cmath module
operator	Numeric operators available as function calls, i.e., operator.sub*(m, n) is equivalent to the* difference *(m - n) for numbers m and n*
random	Various pseudo-random number generators (obsoletes rand and wHRandom)

- *The random module is the general-purpose place to go if you are looking for random numbers. This module comes with various pseudorandom number generators and comes seeded with the current timestamp so it is ready to go as soon as it has loaded. Here are some of the most commonly used functions in the random module:*
- randint()   *Takes two integer values and returns a random integer between those values inclusive*
- randrange()   *Takes the same input as range() and returns a random integer that falls within that range*
- uniform()   *Does almost the same thing as randint(), but returns a float and is inclusive only of the smaller number (exclusive of the larger number)*
- random()   *Works just like uniform() except that the smaller number is fixed at 0.0, and the larger number is fixed at 1.0*
- Choice()   *randomly selects and returns a sequence item*

References

- Core Python Programming , Wesley J.chun, Second Edition, Pearson.
- Python Programming Using Problem Solving Approach, Reema Thareja.
- Core Python Programming, Dr. R.Nageswara Rao
- Core Python Material by Durga Software Solutions