Web Programming

Python Notes

important content only!

Computer Engineering Department, SVNIT

Surat, Gujarat. www.svnit.ac.in

Contents

Table of Contents

Key Points

In python everything is an 'Object'.

Opensource

Freeware

Portable

Dynamically Typed

Rich Set of Libraries

Introduction to Python

History

- Developed by "Guido Van Rossum"
- Launched in 20th February 1991
- Most of the syntax borrowed from C and ABC programming Language.
- In python everything is an 'Object'.

Features

1. Simple and Easy to Learn

- English like language to perform operations/Instructions

Example

```
x=10 if 20<30 else 15
```

- Assign value 10 to x if 20 is less then 30, otherwise Assign 15 to variable x

```
for i in range(10): print(i)
```

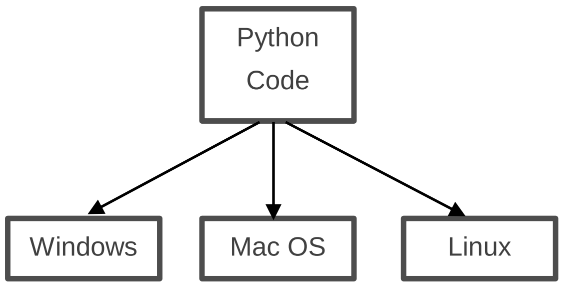
- for i starting from 0 to 10 print value of i

- Same operation if you want to perform in other language for example(Java) then we need to write larger code

```
class base(){
           public static void main(String[] args){
               int i, x;
               if(20<30)
                   x=10;
               else
                   x=15;

               for(i=0;i<10;i++){
                   print(i);
               }
           }
       }
```

2. **Less Keywords**
   - In Python there is around 33 keywords, where as in Java 53 Keywords are there.
3. **License for Python**
   - Python is Freeware: We're not going to pay anything OR freely available to use.
   - Python is Opensource: We're able to see the source code of python. How python is written and if you wish you can change or modify default code of python and develop your own version of python.
   - Example: jython, Java python ...
4. **High Level Programming Language**
   - Python use interpreter to convert python code to convert and execute through machine code.
5. **Platform Independent**
   - Python is platform independent: Same code we can execute on different machines or with different operating system.
   - Write Once, Run Anytime... (WORA)



6. **Portable**
   - Migrating python code from one machine/OS to another machine/OS.
   - Same like mobile number portability: We can change mobile operator without changing our mobile number.
   - We can move our python application/code from Mac OS to windows machine or Linux.
7. **Dynamically Typed**
   - No need to declare variable with data type.
   - Python is able to detect data type automatically based on assigned value.
8. **Both Procedure Oriented and Object Oriented**
   - We can write code without declaring any class and execute it. Just like C programming and we can also write different classes and use Object Oriented Concepts.
9. **Extensible and Embedded**
   - We can use other program written in (C, C++, Java ..) in Python.

Key Points

Backward Compatibility Issue

Not supported for Mobile Application

- ➢ We can easily extend some application which written in other programming language.
- ➢ Python code can be embedded in other programming languages.

10. Extensive Libraries

- ➢ Readymade functionalities are already available through libraries.
- ➢ We have to import and use the modules for various operations.

Drawbacks

1. **No Backward Compatibility**
   - ➢ New versions of python are not providing support to older version of python.
   - ➢ Application/code written Python2 can't be executed on Python3.0.
   - ➢ Python3.0 not supports Python2.0
2. **Not supported for Mobile Applications**
3. **Performance is low compared to assembly level language and some other programming language.**

Flavors

1. **CPython (Standard Flavors of Python)**
   - ➢ Used to work with C language
2. **Jython or JPython**
   - ➢ Run with Java Programming Language
3. **IronPython**
   - ➢ To work with C# application or C#.Net
4. **PyPy**
   - ➢ Python with speed. (performance wise improved)
   - ➢ Uses Python Virtual Machine(PVM) & Just in Time(JIT).
5. **Ruby Python**
   - ➢ To work ruby based program.
6. **Anaconda Python**
   - ➢ To handle big data. (Hadoop)
   - ➢ Large volume of Data Processing.
7. **Stackless**
   - ➢ Python for concurrency control

Versions

1. **Python 1.0 (Jan 1994) (Deprecated)**
2. **Python 2.0 (Oct 2000)**
3. **Python 3.0 (Dec 2008)**

Current Version:

- Python 3.6.3 (2016)

Software Rule:

- Any new version of software should provide support for old version program.
- This rule is not followed by python. [Big Issue with Python]
- Backward compatibility is not there in python.
- By 2020 support for python2 will be removed completely.

Identifiers in Python

Introduction

Name which can be used for identification. it can be:

- ➢ variable name
- ➢ method name
- ➢ class name

Example

```
x=10                               #variable name
def f1():                          #method name
    print(x)
class test(Exception):             #class name
    …
    …
```

Rules for Identifiers

1. **Allowed symbols:**
   - ➢ Alphabet symbols (Upper case & lower case both)
   - ➢ Digits (0...9)
   - ➢ Underscore _ (Only allowed special character in python)

Examples

```
cash = 100            -valid
ca$h = 100            -not valid - $ symbol is not allowed
total = 123           -valid
total23 = 456         -valid
123total = 567        -not valid - Starts with digits not allowed
percent% = 55.55      -not valid - % symbol is not allowed
total marks = 500     -not valid - space is not allowed
total_marks = 555     -valid
total-marks = 400     -not valid - - symbol is not allowed
```

2. **Identifiers must not start with digit.**

3. **Python identifiers are case sensitive**

```
total = 10
Total = 10
TOTAL = 10
>> These are 3 different variables
```

4. **Keywords as identifier is not allowed**

```
x = 10      -valid
if = 10     -not valid – ‘if’ is reserved keyword in python
```

5. **There is no limit of the length of python identifier.**
6. **If identifier starts with underscore its indicate “private”**

_ (underscore)	Private
__ (double underscore)	Strongly Private
__Identifier__	Language Specific Identifier

Keywords in Python

Reserved Words

To represent meaning or functionality. Approx. 33 Reserved words in Python3 (Since python3.5)

- True, False, None
- and, or, not, is
- if, else, elif
- while, for, break, continue, return, in, yield
- try, except, finally, raise, assert
- import, from, as, class, def, pass, global, nonlocal, lambda, del, with

There are only alphabets in all reserved words. No any other symbols or digits is there in any reserved word names.

Except first 3, all reserved words is in small/lower case only.

- switch case and do...while... is not available in python
- public, private, protected.... these type of keywords is not available in python.

To display all keywords, execute following code.

```
import keyword
print(keyword.kwlist)
```

Output:

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'brea
k', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finall
y', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonl
ocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', '
yield']
```

Data Types in Python

Introduction

Represents type of value

1. int
2. float
3. complex
4. bool
5. str
6. bytes
7. bytearray
8. range
9. list
10. tuple
11. set
12. frozenset
13. dict
14. None

14 Inbuilt Data Types

Function to explore more with data types

➢ print()	-to print
➢ type()	-to check the data type
➢ id()	-to get the memory address of data

Example

```
>>> a=10
>>> print(a)
10
>>> type(a)
<class 'int'>
>>> id(a)
1942316352          <-- Address of variable a where 10 is stored
```

int data type

For Integral values (Numbers without decimal point)

- Long data type is not available from python3, so larger numbers also stored as int data type only.
- In other programming languages the size of primitive data types is fixed.
- In C programming size of int is 2 or 4 byte (based on architecture). But in python int data also stored as object so there is no such limit for int.
- If you wish you can store unlimited integral data (depending on your memory capacity)
- In C, range for int is -32768 to +32767. In python there is no such thing like range for storing data (for any type of data not just for int).
- In python everything is an Object. And getting size of an object is difficult.
- int can represent:
    1. Decimal form
    2. Binary form
    3. Octal form
    4. Hexadecimal form

1. **Decimal (Base 10)**
    - 0 to 9

Example

```
>>> a = 7869
>>> a
7869
>>> type(a)
<class 'int'>
```

2. **Binary (Base 2)**
    - 0 and 1
    - Work with both negative and positive numbers
    - Starts with Zero then small b or capital B - indicates binary values

Example

```
>>> a=0b1111        -valid
>>> a=0B1111        -valid
>>> a
15                 # Decimal of binary value 1111 is 15.
>>> bin(a)
```

```
0b1111                 # bin() function to represent in binary
>>> type(a)
<class 'int'>
```

3. **Octal (Base 8)**
   - 0 to 7
   - Work with both negative and positive numbers
   - Starts with Zero then small o or capital O - indicates Octal values

Example

```
>>> a=777
>>> a
777                    #Decimal (Default)
>>> a=0o777
>>> a
511                    #Decimal of octal value 777 is 511
>>> oct(a)
0o777                  #oct() function to represent in octal
>>> type(a)
<class 'int'>
```

4. **Hexa Decimal (Base 16)**
   - 0 to 9, a to f   -> Both a to f OR A to F (lower case and capital case allowed)
   - Work with both negative and positive numbers
   - Starts with Zero then small x or capital X - indicates Hexa Decimal values

Example

```
>>> a=10
>>> a
10                     #Decimal (Default)
>>> a = 0xFFFF
>>> a
65535                  #Decimal of Hexa Decimal value FFFF is 65535
>>> hex(a)             #hex() function to represent in Hexadecimal
0xffff
>>> type(a)
<class 'int'>
```

Key Points

a=10

#decimal

b=0b10

#binary

c=0o10

#octal

d=0x10

#hexa decimal

Utility functions for int

➢ Various base conversion can be performed. Default form is decimal if you don't specify any other data type. All these utility function will return string data.

1. **bin()**

Example

```
>>> bin(15)        #converts decimal to binary
'0b1111'
>>> bin(0o777)     #converts octal to binary
'0b111111111'
>>> bin(0x10fA)    #converts hexa decimal to binary
'0b1000011111010'
```

2. **oct()**

Example

```
>>> oct(0b1100)     #converts binary to octal
'0o14'
>>> oct(0xABA1)     #converts hexa decimal to octal
'0o125641'
>>> oct(99)         #converts decimal to octal
'0o143'
>>> oct(0o777)      #returns string of octal
'0o777'
```

3. **hex()**

Example

```
>>> hex(13)                    #converts decimal to hexa decimal
'0xd'
>>> hex(0b1010101111001010)    #converts binary to hexa decimal
'0xabca'
>>> hex(0xabca)                #returns string of hexadecimal
'0xabca'
>>> hex(0o7711771155)          #converts octal to hexa decimal
'0x3f27f26d'
>>> type(hex(0o7711771155))    #Return string value
<class 'str'>
```

float data type

Float: to store floating point values

Example

```
>>> f=123.456            #Decimal point in value indicates float value
>>> type(f)
<class 'float'>
>>> salary=30000.00      #Decimal point in value indicates float value
>>> price=530            #No decimal point in value indicates int value
>>> salary
30000.0
>>> price
530
>>> type(price)          #price is int type of variable
<class 'int'>
>>> price=530.0          #reassigning price with float type of data
>>> price
530.0
>>> type(price)          #now price is float type of data
<class 'float'>
```

- Binary, octal, hexadecimal values are not allowed in float

Example

```
>>> f=0b1010.1010         #binary data cannot be stored as float data
SyntaxError: invalid syntax
>>> f=0o127.12            #Octal data cannot be stored as float data
SyntaxError: invalid syntax
>>> f=0xAA10.AA     #Hexa Decimal data cannot be stored as float data
AttributeError: 'int' object has no attribute 'AA'
```

- Exponential data can be represented with float data type
- Assume we saved data as 1.2e5 => 1.2 * 10^5

  => 1.2 * 100000

  => 120000

- Through this facility we can store larger numbers within small values

Example

```
>>> f=1.2e5
>>> f
120000.0
>>> type(f)
<class 'float'>
```

complex data type

Complex: to store complex data

- Useful in mathematics based, scientific applications…
- Format for complex number:

  a+bj

  - where, a is real part (both int with all forms (binary, octal, hexadecimal) and float values can be accepted)
  - b is imaginary part (both int without forms (binary, octal, hexadecimal) and float values can be accepted)
  - $j^2 = -1$
  - $j = \sqrt{-1}$

Example

```
>>> a=10+20j                  #variable a having complex data
>>> type(a)
<class 'complex'>
>>> a                         #printing variable a will return complex data
(10+20j)
>>> b=20+30j                  #variable b having complex data
>>> a+b                       #arithmetic operations
(30+50j)
>>> c=10.10+20.20j            #real and imaginary supports int and float
>>> d=-10.15-30.30j
>>> c
(10.1+20.2j)
>>> d
(-10.15-30.3j)
>>> type(d)
<class 'complex'>
>>> e=22*5j                   #complex form with multiplication operator
>>> type(e)
<class 'complex'>
>>> f=32+23i             #compulsory we have to use 'j'
SyntaxError: invalid syntax
>>> f=22+j45             #compulsory we have to maintain the order of j.
NameError: name 'j45' is not defined
>>> f=3j                 #Real part is optional. valid
>>> type(f)
<class 'complex'>
```

➢ Real part can accept binary, octal or hexa decimal data.

Example

```
>>> a = 0b1011 + 12j
>>> b = 0o666 + 32j
>>> a+b       #perform arithmetic and gives decimal real part as output
(449+44j)
>>> c = 0xAAFF + 22.05j
>>> type(c)
<class 'complex'>
```

➢ Imaginary part will never accept binary, octal or decimal form of int

```
>>> a = 0b1100 + 0b1101j
SyntaxError: invalid syntax
>>> b = 34.34 + 0o457j
SyntaxError: invalid syntax
>>> c = 0xAA + 0xBBj
SyntaxError: invalid syntax
```

➢ To know the real and imaginary part of complex data

```
>>> a=10+20j
>>> a.real
10.0
>>> a.imag
20.0
>>> b=0xFF+12j
>>> b.real
255.0
```

```
>>> b.imag
12.0
>>> type(a.real)
<class 'float'>
```

- In above example: a=10+20j
  - real part inserted by user is 'int' type
  - but a.real operation is giving 'float' type
  - That is because Internally python treats real part or imag part as 'float' only.

bool data type

Boolean: to represent logical values

- Allowed values (Compulsory 'T' and 'F' is capital only)
  - True
  - False

Example

```
>>> a=True
>>> type(a)
<class 'bool'>
>>> a
True
>>> b=true      #here true is not int, float, str.
                #Python treat true as variable which not defined yet
                #so it'll generate error.
NameError: name 'true' is not defined
>>> a=10        #assigning int value to a
>>> b=20        #assigning int value to b
>>> c=a<b       #comparing a less then b and assigning reult in c
>>> c           #c stores bool type of data
True
>>> b='true'    #here 'true' is str not bool type
>>> type(b)
<class 'str'>
>>> c='True'    #here 'True' is str not bool type
>>> type(c)
<class 'str'>
```

- Internally True treated as 1.
- And False treated as 0.

```
>>> True+True     #True means 1, so 1 + 1 = 2
2
>>> True+False    # 1 + 0 = 1
1
>>> int(True)     #checking int value of True
1
>>> bin(True)     #binary of True
'0b1'
>>> hex(False)                                #hexa Decimal of True
'0x0'
>>> int(-True)                                #int of negative True
-1
```