

# Web Programming

## Python Notes

*important content only!*

## Table of Contents

## Contents

Introduction to Python	1
History	1
Features	1
Drawbacks	3
Flavors	3
Versions	4
Identifiers in Python	5
Introduction	5
Rules for Identifiers	5
Keywords in Python	7
Reserved Words	7
Data Types in Python	8
Introduction	8
Function to explore more with data types	8
int data type	9
Utility functions for int	11
float data type	12
complex data type	13
bool data type	15
String Data Type & Operations	17
str data type	17
Slice Operator for String [:]	20

## Table of Contents

float()	28
complex()	30
bool()	31
Immutability	35
Immutability of int	36
Immutability of float	44
Immutability of complex	47
Immutability of bool	48
Immutability of String	49

### 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 than 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++){
```

**2. Less Keywords**

- In Python there is around 33 keywords, where as in Java 53 Keywords are there.

**3. License for Python**

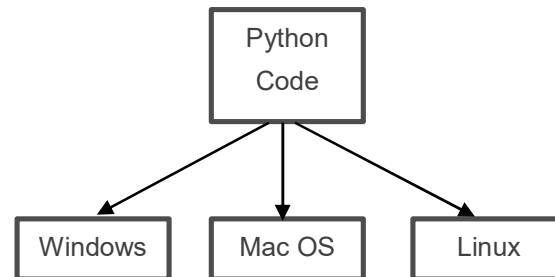
- Python is Freeware: We're not going to pay anything OR freely available to use.
- Python is Opensource: We're able to see the source code of python. How python is written and if you wish you can change or modify default code of python and develop your own version of python.
- Example: jython, Java 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**

**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**

## **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
total23_456_789    -valid - all symbols are allowed
```



**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”**

<code>_</code> (underscore)	Private
<code>__</code> (double underscore)	Strongly Private
<code>__Identifier__</code>	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', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', '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)
```

## 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
```

```
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)
0xffff      #hex() function to represent in Hexadecimal
>>> type(a)
```

**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
'0b11111111'
>>> 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(0o7771771155)  #converts octal to hexa decimal
'0x3f27f26d'
```

## 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  $1.2e5 \Rightarrow 1.2 * 10^5$

$\Rightarrow 1.2 * 100000$

```
>>> 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(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
```

```
>>> 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 result in c
>>> c           #c stores bool type of data
True
>>> b='true'    #here 'true' is str not bool type
```

- 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
```

## String Data Type & Operations

### str data type

String: To store string values

- Character data type is not available in python.
- Even single character is also string with string length 1.

#### Example

```
>>>s='r'  
>>>type(s)  
<class 'str'>
```

- Sequence of characters enclosed with single quote, double quote or even triple single quotes and triple double quotes.
- Conventionally single quotes should be used.

#### Example

```
>>> s='abc'           #valid  
>>> s="abc"           #valid  
>>> s='''abc'''       #valid  
>>> s="""abc"""       #valid  
>>> s  
'abc'  
>>> type(s)  
<class 'str'>
```

- while writing python program if you need to print statement having multiple lines then we can use triple quotes.
- Triple quotes is used for multiline comment as well.

Program

```
s="abc xyz"
print(s)
t='svnit'
print(t)
u=''coed'''
print(u)

v=''abc
  xyz
    svnit
  coed'''

print(v)
```

Output

```
abc xyz
svnit
coed
abc
  xyz
    svnit
  coed
```

Example [Single Quotes for Multiline Strings: Invalid]

```
>>> s = 'abc
      xyz'
#invalid
```

Program

```
s='abc
xyz'
print(s)
```

```
      ^
SyntaxError: EOL while scanning string literal
```

Example [Triple Double Quotes for Multiline Strings: Valid]

```
>>> s = """Abc
        xyz"""          #valid
```

Program

```
s="""abc
    xyz"""
print(s)
```

Output

```
abc
xyz
```

Example [Double Quotes for Multiline Strings: Invalid]

```
>>> s = "Ronak
        Ahir"          #invalid
```

Program

```
s="ronak
    ahir"
print(s)
```

Output

```
File "test2.py", line 3
    s="ronak
      ^
SyntaxError: EOL while scanning string literal
```

```
s='Computer Engineering Department of "SVNIT" college'
```

### Program

```
s='1. Computer Engineering Department of "SVNIT" college'
print(s)
s="2. Computer Engineering Department of 'SVNIT' college"
print(s)
s='''3. Computer Engineering Department of "SVNIT" college'''
print(s)
s="""4. Computer Engineering Department of "SVNIT" college"""
print(s)
s='''5. Computer Engineering Department of 'SVNIT' college'''
print(s)
s="""6. Computer Engineering Department of 'SVNIT' college"""
print(s)
```

### Output

```
1. Computer Engineering Department of "SVNIT" college
2. Computer Engineering Department of 'SVNIT' college
3. Computer Engineering Department of "SVNIT" college
4. Computer Engineering Department of "SVNIT" college
5. Computer Engineering Department of 'SVNIT' college
6. Computer Engineering Department of 'SVNIT' college
```

## Slice Operator for String [:]

- Piece of word of substring is slice

### Example

- COEDSVNIT
  - COED -> 1st Slice
  - SVNIT-> 2nd Slice
- String can be accessed by index.

- Negative index: right to left

**Example**

← Right to Left									
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
r	o	n	a	k		a	h	i	r
0	1	2	3	4	5	6	7	8	9
Left to Right →									

**Example**

```
>>> s='ronak ahir'
>>> s[0]
'r'
>>> s[1]
'o'
>>> s[2]
'n'
>>> s[15]           #Accessing index which is not defined gives error
IndexError: string index out of range
>>> s[-1]
'r'
>>> s[-2]
'i'
>>> s[-3]
'h'
>>> s[-4]
'a'
>>> s[-45]          #Accessing index which is not defined gives error
IndexError: string index out of range
```

➤ Slice Operator Syntax (for some string S):



**S [ begin : end ]**Example

-5	-4	-3	-2	-1
r	O	n	a	k
0	1	2	3	4

Example

```
>>> s='ronak'
>>> s
'ronak'
>>> s[1:4]
'ona'
#Begin from index [1] and end before index[4] that is upto index[3]
#Return substring => from begin to end-1

>>> s[2:4]
'na'
#Begin from index [2] and end before index[4] that is upto index[3]
#Return substring => from begin to end-1

>>> s[1:]
'onak'
#if end index is not given then by default end of the string is
considered as end index
#Return substring from begin to end ***Important point

>>> s[3:]
'ak'
#Begin index to end of the string
#Return substring from begin to end of the string
```

```
>>> s[:4]
'rona'
#if begin index is not given then by default index [0] is considered as
begin index
#Return substring => from index[0] to end-1 **Important

>>> s[:2]
'ro'
#default begin index[0] to end-1 index
#Return substring => from index[0] to end-1 **Important

>>> s[:]
'ronak'
#if both begin and end index is not given then take index[0] as begin an
d index[length] as end
#substring: begin[0] to end[5]
```

- If begin index is higher than end index, then slice return empty string.
- Printing flow is left to right (in case of syntax 1. S[ begin : end] )

### Example

```
>>> s[4:2]          #Begin index is higher than end index
''

>>> s[5:4]          #Begin index is higher than end index
''

>>> s[-1:]
'k'
#Begin index [-1] to end of the string that is index[5]

>>> s[-2:]
```

```
#Begin index is default [0] to end-1 that is -1-1 => -2
#[Reason end index=index-1]
#index[0] to index[-2]
>>> s[:-]
SyntaxError: invalid syntax
#Only negative sign as begin or end is not allowed

>>> s[-2:-1]
'a'
#begin index is [-2] and end index is -1. that is -1-1=>-2
#[Reason end index=index-1]
#Begin and End on index[-2]

>>> s[-5:1]
'r'
#begin index is [-5] and end index is 1. that is 1-1=>0
#[Reason end index=index-1]
#Begin index [-5] and End index[0] having character 'r'
```

- Index starts from -1 to right side gives nothing

```
>>> s[-1:1]
''

>>> s[-1:1]
''

>>> s[-1:0]
''

>>> s[-1:2]
''
```

- In case of slice operator larger index will not generate any error.
- If index out of bound is detected, by default length of the string is considered.

```
>>> s[2:100]          #End index is 100 but still it will work file.
'nak'
```

Indexing

-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
C	o	m	p	u	t	e	r		D	e	p	a	r	t	m	r	n	t		S	V	N	I	T
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

```
>>> s[1:10:2]
'optrD'
#Begin[1], End[9], Steps[2]
#(End is [9] Because : index-1=>10-1=>9)
#Jump or print every second character due to step 2 is given.

>>> s[:18:3]
'CpeDam'
#Begin[0](default if not specified), End[17], Steps[3]

>>> s[10::2]
'eatetSNT'
#Begin[10], End[24](Length of the string if not specified), Steps[2]

>>> s[::-1]
'TINVS tnemtrapeD retupmoC'
#Begin[24], End[0], Steps[-1]
#Prints in reverse order step is -1

>>> s[:15:-3]
'TVt'
#Begin[24], End[15], Steps[-3]

>>> s[5::-2]
'tpo'
#Begin[5], End[0], steps[-2]
```

Abcababc

- Compulsory one argument must be int only
- Order for an \* operator is not mandatory to maintain

### Example

[illegible]

## Type Casting

### Available Functions

There are many functions for type casting. The below are few of them to demonstrate covered topics so far.

- `int()`
- `float()`
- `complex()`
- `bool()`
- `str()`

### `int()`

To convert from other data type to int.

#### Example

```
>>> int(123.456)      #float to int
123                  #return int data
>>> int(123)          #int to int
123                  #return same int data
>>> int(10+20j)       #complex to int      #not allowed      #Error
TypeError: can't convert complex to int
>>> int(True)         #boolean to int
1                    #return internal int value of True
>>> int(False)        #boolean to int
0                    #return internal int value of False
>>> int("10")         #string(having digits only not symbols & alphabets) to int
10                  #return int data
>>> int("85.85")
```

```
ValueError: invalid literal for int() with base 10: '0b101010'  
#string(int value in binary form) to int  #not allowed  #Error  
  
>>> int(0xABC1)          #int(Hexadecimal form) to int  
43969                    #return int data  
  
>>> int("svnit")  
ValueError: invalid literal for int() with base 10: 'svnit'  
#String(Having alphabets) to int  #not allowed  #Error
```

- If you want to convert string value to int, compulsory string values must integral

## float()

To convert from other data type to float

### Example

```
>>> float(10)  
10.0  
#int to float  
#Returns int value  
  
>>> float(10+20j)  
TypeError: can't convert complex to float  
#Complex to float  #not allowed  #Error  
>>> float(True)  
1.0  
#bool to float  
#Internally True converted to int that is 1 then returns float of 1=>1.0  
  
>>> float(False)  
0.0  
#bool to float  
#Internally False converted to int that is 0 then returns float of 0=>0.  
0
```

```
#returns float value

>>> float("85.85")
85.85
#str(having float) to float
#returns float value

>>> float("ten")
ValueError: could not convert string to float: 'ten'
#str(having alphabets) to float      #not allowed      #Error

>>> float(0b1111)
15.0
#int(binary form) to float
#returns corresponding decimal value type casted to float

>>> float("0b1111")
ValueError: could not convert string to float: '0b1111'
#str(having int(binary form)) to float      #not allowed      #Error

>>> float(0o777)
511.0
#int(octal form) to float
#returns corresponding decimal value type casted to float

>>> float(-0xFFFF)
-65535.0
#int(negative hexadecimal form) to float
#returns corresponding decimal value type casted to float

>>> float(*0b1010)
TypeError: type object argument after * must be an iterable, not int
#int(multiplication symbol binary form) to float
#Not allowed      #Error
```



## complex()

Converts other data type to complex type

- 1. `complex(x)`  $\Rightarrow x+0j$ 
  - Single argument in `complex()` function treated as real part only.
  - By default, imaginary part will be 0
- 2. `complex(x,y)`  $\Rightarrow x+yj$ 
  - Two arguments in `complex()` function.
  - First argument treated as real part of complex number
  - Second argument treated as imaginary part of complex number

### Example [ `complex(x) $\Rightarrow x+0j$ ]`

```
>>> complex(10)          #int to complex
(10+0j)

>>> complex(10.5)        #float to complex
(10.5+0j)

>>> complex(True)
(1+0j)
#bool to complex    =>bool True treated as int of 1

>>> complex(False)
0j
#bool to complex    =>bool False treated as int of 0

>>> complex("10")
(10+0j)
#str(having int) to complex

>>> complex("10.5")
(10.5+0j)
#str(having float) to complex
```

```
>>> complex(0b111)
(7+0j)
#int(binary form) to complex    #valid for all other int forms

>>> complex("0b111")
ValueError: complex() arg is a malformed string
#str(having int(binary form)) to complex    #invalid    #Error
```

Example [ `complex(x,y) => x+yj` ]

```
>>> complex(10,20)          #complex(int,int)          #valid
(10+20j)
>>> complex(10.15,20.25)    #complex(float,float)     #valid
(10.15+20.25j)
>>> complex(10,25.26)       #complex(int,float)        #valid
(10+25.26j)
>>> complex(10.15,25)       #complex(float,int)         #valid
(10.15+25j)
>>> complex(True,False)     #complex(bool,bool)        #valid
(1+0j)
>>> complex("10",25)        #complex(str,int)           #invalid
TypeError: complex() can't take second arg if first is a string
>>> complex(10,"25")         #complex(int,str)           #invalid
TypeError: complex() second arg can't be a string
>>> complex("10","20")      #complex(str,str)           #invalid
TypeError: complex() can't take second arg if first is a string
>>> complex(int("10"),float("25.25"))
(10+25.25j)
#complex(int(str),float(str))    #valid
#first str to int and str to float performed thats why it is valid
```

## bool()

Convert any type to Boolean type

Example

```
>>> bool(0)          #bool of zero      #False
False
>>> bool(1)          #bool of non-zero   #True
True
>>> bool(85)         #bool of non-zero   #True
True
>>> bool(-24)        #bool of non-zero   #True
True
>>> bool(-0)         #bool of zero       #False
False
>>> bool(0.0)        #bool of zero       #False
False
>>> bool(0.23)       #bool of non-zero   #True
True
>>> bool(0.00000001) #bool of non-zero   #True
True
>>> bool(0.00000000) #bool of zero       #False
False
>>> bool(-0.123)     #bool of non-zero   #True
True
```

- bool() with complex data as an argument
- If both real and imaginary part is zero then False, otherwise True

Example

```
>>> bool(10+20j)     #both real and imag part is non-zero
True
>>> bool(0+1j)       #real part is zero but imag part is non-zero
True
>>> bool(1+0j)       #imag part is zero but real part is non-zero
True
>>> bool(0+0j)       #both real and imag part is zero
```

- bool() with string data as an argument
- If argument is empty string then False, otherwise True

**Example**

```
>>> bool('')          #Empty string. just single quote open and close
False

>>> bool('ahir')      #Non empty string
True

>>> bool(' ')         #Non empty string, space between quotes
True

>>> bool(', ')        #Non-empty string, comma symbol between quotes
True
```

- str() function : for single arguments, never going to raise any error.
- **Example**

```
>>> str(20)           #str of int          #valid
'20'

>>> str(25.5)         #str of float        #valid
'25.5'

>>> str(True)         #str of bool         #valid
'True'

>>> str(False)        #str of bool         #valid
'False'

>>> str(10+20j)        #str of complex     #valid
'10+20j'
```

```
#str of arithmetic operation of int      #valid
#first arithmetic will be performed and then answer converted to str

>>> str("abc"*5)      #str with repetition of string
'abcabcabcabcabc'
```

## Immutability

Python stores data in the form of object. In other programming language primitive data types having fixed size (range for storage) to store data.

- Immutability of an object states its nature of unchangeable once its created.
- Once we create object, we can't perform any changes in that object.
- If we try to make change in already created object, then python will create new object with changed value.
- In python we have various data types that are mutable or immutable. Let's understand immutability concepts in detail with examples.

[ pythonshell allows to create/read/edit and execute your programs under same roof without touching command line ]

### Python Interpreter works with 2 modes

1. Interactive Mode
2. Script Mode

1. **Interactive mode:** python interpreter waits for you to enter command. When you type the command, python interpreter goes ahead and executes the command, then it wait again for your next command.

2. **Script mode:** python interpreter runs a program from source file. First we have to write program in separate file as filename.py  
Here, py is the extension for python file.  
To run python file/script you have to execute following command in terminal/command prompt.  
C:/>py filename.py  
OR  
C:/>python filename.py  
OR  
C:/>python3 filename.py

## Immutability of int

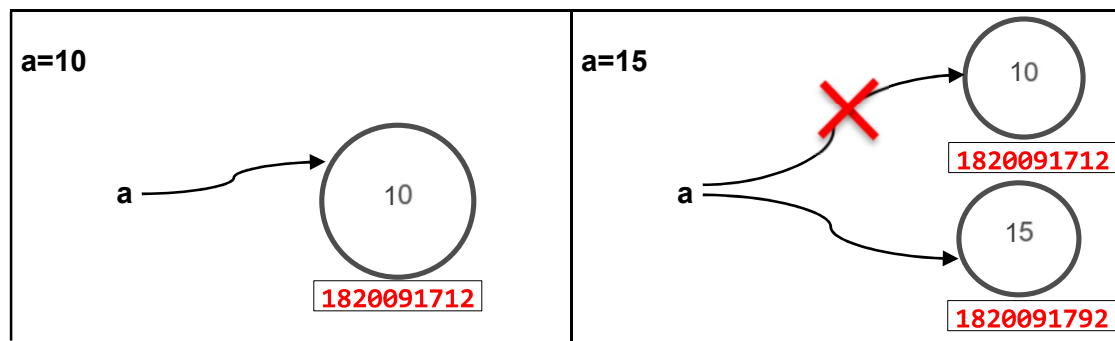
- Not changeable

### Example (Script Mode):

```
a=10
print('Address of a:',id(a))
print('Reassigning new value to same variable a')
a=15
print('Address of a after reassignment:',id(a))
```

### Output

```
Address of a: 1820091712
Reassigning new value to same variable a
Address of a after reassignment: 1820091792
```



- Here in above example first we declared variable a with value 10.
- Initially variable a is stored on some memory location **1820091712**
- Now when we reassign same variable with some new value it will create new object that is stored on different memory location **1820091792**
- So here we can see that this object is immutable.
- Reason behind this immutability is python uses same objects for multiple variables. So if

Example(Script Mode)

```
a=10
b=10
c=20
d=10
e=25
f=10
g=10
h=20
a=55
print('Address of a:',id(a))
print('Address of b:',id(b))
print('Address of c:',id(c))
print('Address of d:',id(d))
print('Address of e:',id(e))
print('Address of f:',id(f))
print('Address of g:',id(g))
print('Address of h:',id(h))
print('Address of h:',id(a))
```

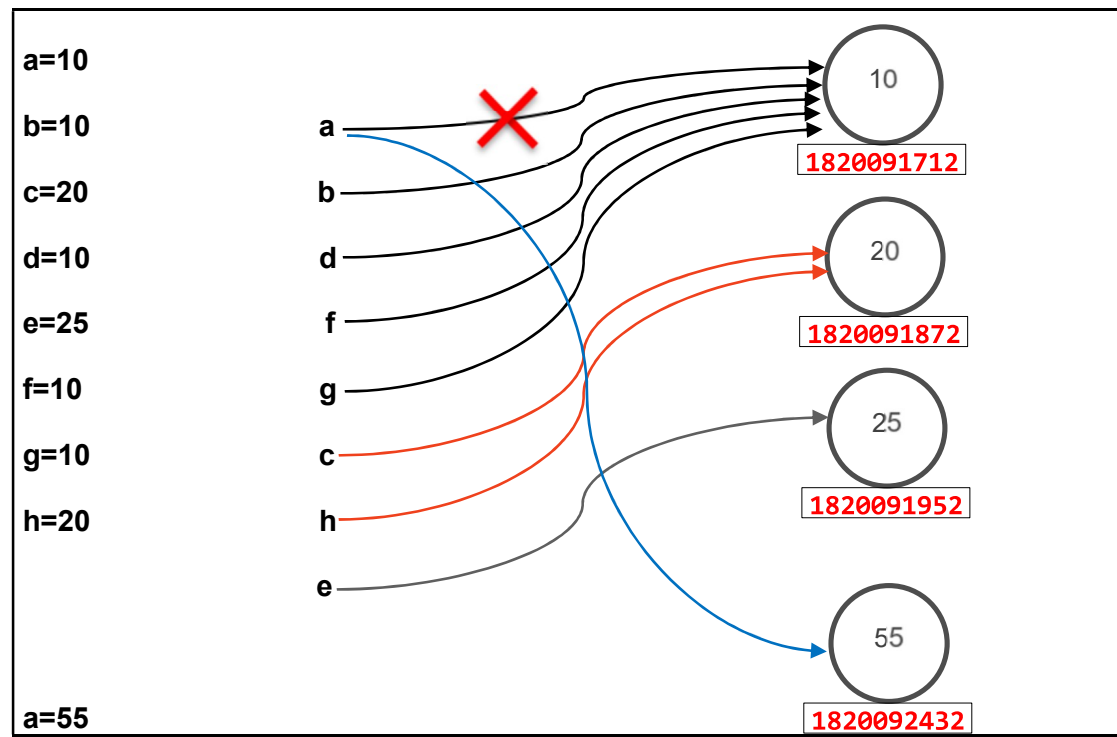
Output

```
Address of a: 1820091712
Address of b: 1820091712
Address of c: 1820091872
Address of d: 1820091712
Address of e: 1820091952
Address of f: 1820091712
Address of g: 1820091712
Address of h: 1820091872
Address of a: 1820092432
```

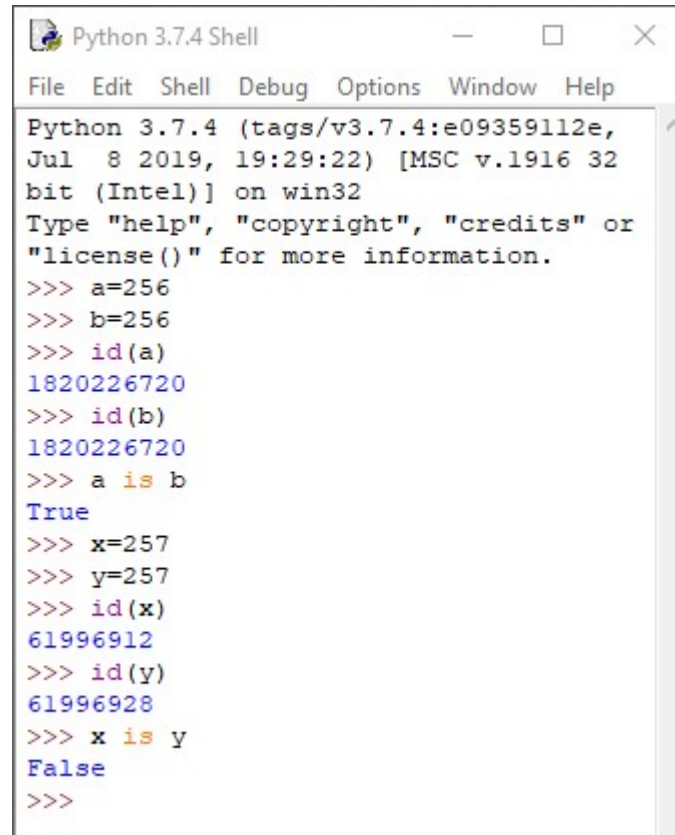
- Here in above example notice that address of variables having value 10 is same. i.e. variable a, b, d, f, g having address **1820091712**
- Other variables c and h having same value 20. And address of these variable is also same



- By considering variable, a, b, d, f, g, it's clear that python uses same object for different variable in case of same value.
- Now assume that after some lines of code if we need to change the value of variable a with new value 55.
- If python changes the same object having value 10 to 55 then all other variables pointing to that object gets affected. i.e. value of b, d, f, g is also changed to 55.
- So to prevent these kind of problem python creates new object with value 55 for variable a.
- After creating new object with value 55, now variable a no longer points to old object and it points new object with value 55.



- Here, we can conclude that **int is Immutable data type**.
- Every time we need not to check the address using `id()` method. We can check whether these variables are sharing same objects or not by using `'is'` method.
- Let's execute few more example on pythonshell.

Example (Interactive Mode)

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e,
Jul  8 2019, 19:29:22) [MSC v.1916 32
bit (Intel)] on win32
Type "help", "copyright", "credits" or
"license()" for more information.
>>> a=256
>>> b=256
>>> id(a)
1820226720
>>> id(b)
1820226720
>>> a is b
True
>>> x=257
>>> y=257
>>> id(x)
61996912
>>> id(y)
61996928
>>> x is y
False
>>>
```

- Consider the above example. Here address of variable **a** and **b** is same. We can check whether these two variables are pointing to same object or not by using '**is**' operator.
- If both a and b pointing to same object, **a is b** instruction give True, Otherwise False.
- Now in case of x and y variable both variables have same value but here it will create two different object in pythonshell. That is because in case of 'int' data type if we try to assign values that is more than 256 then it will create new objects.
- For the range -5 to 256 python will not going to create different objects if multiple variables have assigned same values in pythonshell.
- But after 256, python create new objects for even same values that assigned to different

- Now consider the variable **x** and **y** in above example. Both are assigned 257 as value and see the address of variable x and y.
- The address of x and y is different. So here two different objects are created for same value.

Example (Interactive Mode)

```
>>> x=1000
>>> y=1000
>>> x is y
False
>>> a=-1
>>> b=-1
>>> a is b
True
>>> a=-2
>>> b=-2
>>> a is b
True
>>> a=-5
>>> b=-5
>>> a is b
True
>>> a=-6
>>> b=-6
>>> a is b
False
>>> a=-25
>>> b=-25
>>> a is b
False
>>> p=0
>>> q=0
>>> p is q
True
>>>
```

- Considering above example: values from -5 to 256, python will not create separate objects for same values for int data type in pythonshell.
- Now reason for this kind of behavior is, -5 to 256 is most commonly used values that is used by developers/users.
- So when we start pythonshell, python internally creates objects starting from -5 to 256. So at run time while executing instruction python don't waste time in creating objects which

- The reason is, if python creates 1 lakh objects at starting than it will take much longer time to start pythonshell. And waste of memory will increase due to unnecessary objects creation.

Now let's see example for Script Mode

**Let's perform same operations in separate python file and execute that python file through command line.**

#### Example(Script Mode)

```
#file name is : LAB8.py

a=256
b=256
print(a is b)

x= 257
y= 257
print(x is y)

print('Address x:',id(x))
print('Address y:',id(y))

p=-6
q=-6

print('Address p:',id(p))
print('Address q:',id(q))
print(p is q)
```

#### Output

```
E:\workspace\Ronak\WP> py LAB8.py
True
True
```

- Consider the file LAB8.py file. Here, there is no such range -5 to 256 is created internally.
- Check the address of variables that denotes same memory locations. Unlike pythonshell, command line execution of file treats all values as same kind of objects.

**So int() type of data is immutable**

### Immutability of float

- Not changeable
- Float type of data is also immutable in python
- For floating point data reusing same object is not possible in interactive mode.

#### Example (Interactive Mode)

```
>>> a=5.5
>>> b=5.5
>>> a is b
False
>>> id(a)
50474880
>>> id(b)
50471024
>>> a=0.0
>>> b=0.0
>>> a is b
False
>>> a=-1
>>> b=-1
>>> a is b
True
>>> a=-1.1
>>> b=-1.1
>>> a is b
False
>>>
```

- For all the values there will be new object only for float type having different memory location.

Example (Interactive Mode)

```
>>> a=5.5
>>> b=5.5
>>> id(a)
55243184
>>> id(b)
50471024
>>> a=10.15
>>> id(a)
50474880
>>> a=15.20
>>> id(a)
55243184
>>>
```

- In above pythonshell script address of **a** and **b** are different however its value is same. So python creates new object every time when it executes float variable for pythonshell.
- Now when we change the value of already created object, still python is going to create new object only. i.e. see the memory address of **a** after reassigning value of **a**.

Example (Script Mode)

```
#file name is : LAB8.py

a=25.25
b=25.25
print(a is b)

x= 0.0
y= 0.0
print(x is y)
print('Address x:',id(x))
print('Address y:',id(y))

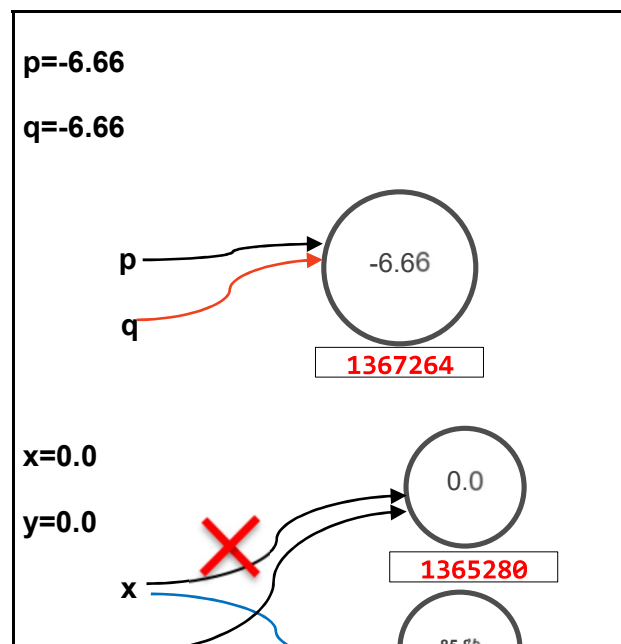
x=85.85
print('Address x:',id(x))
```



Output

```
E:\workspace\Ronak\WP> py LAB8.py
True
True
Address x: 1365280
Address y: 1365280
Address x: 46481936
Address p: 1367264
Address q: 1367264
True
```

- Objects will be reused in float.
- Reassigning of float variable creates new object. Won't change existing object because that object might have referred by other variable also.



### Immutability of complex

- Not changeable
- Complex type of data is immutable in python
- For complex data reusing same object is not possible in interactive mode.
- While executing on script mode complex data object can be shared. It's immutable because change in object may lead to problem if same object is shared by other variables.

#### Example (Interactive Mode)

```
>>> x=10+20j
>>> id(x)
13051952
>>> y=10+20j
>>> id(y)
50575096
>>> x is y
False
>>>
```

- Here in above example both variables x and y having same values but stored as different object on different memory location.

#### Example (Script Mode)

```
x=10+20j
y=10+20j
z=10+35j
print('Address of x:',id(x))
print('Address of y:',id(y))
print('Address of z:',id(z))
print('x is y:',x is y)
x=20+20j
print('Address of x:',id(x))
print('x is y:',x is y)
```

#### Output

```
x is y: True
Address of x: 13121704
x is y: False
```

- In script mode we can observe that reassigning variable x creates new object. So complex data is immutable.

## Immutability of bool

- When python interpreter starts it creates objects with value True and False for Boolean data.
- Every time we create variable with bool value, already created object get pointed to that variable.
- Bool data object is immutable. At any point of time when reassign new value to existing value it's just get reference to one of the already created Boolean object.
- For both interactive mode and script mode, bool object works same.

### Example (Interactive Mode)

```
>>> a=True
>>> id(a)
1820005584
>>> b=True
>>> id(b)
1820005584
>>> a is b
True
>>> c=False
>>> id(c)
1820005600
>>> a=False
>>> id(a)
1820005600
>>> a is b
False
>>>
```

For above example only two objects is created.

```
b=True
print('Address of b:',id(b))
c=a
print('Address of c:',id(c))
d=False
print('Address of d:',id(d))
a=False
print('Address of a after reassigning value:',id(a))
```

#### Output

```
E:\workspace\Ronak\WP> py LAB8.py
Address of a: 1820005584
Address of b: 1820005584
Address of c: 1820005584
Address of d: 1820005600
Address of a after reassigning value: 1820005600
```

- Two objects created with value True and False. These objects are pointed by various variables to represent the bool value.

## Immutability of String

- All the string literals are going to be reused in python.
- String objects are immutable.

#### Example (Interactive Mode)

```
>>> a="acabc123123"
>>> b="acabc123123"
>>> a is b
True
>>> x='abc xyz'
>>> y='abc xyz'
>>> x is y
```

- In interactive mode of python, string containing space or multiple words created as new object.
- For string having single word creates object that can be shared by various variables.
- Let us see the example with script mode.

#### Example (Script Mode)

```
s1='this is sample text'
s2='svnit computer engineering'
s3='this is sample text'
s4='computer'
s5='svnit computer engineering'
s6='''this is computer engineering department
    svnit college'''
s7="""this is computer engineering department
    svnit college"""

print('id(s1):',id(s1),'\nid(s2):',id(s2),'\nid(s3):',id(s3),'\nid(s4):',
      id(s4),'\nid(s5):',id(s5),'\nid(s6):',id(s6),'\nid(s7):',id(s7))
```

#### Output

```
E:\workspace\Ronak\WP> py LAB8.py
id(s1): 62452128
id(s2): 61207680
id(s3): 62452128
id(s4): 61087360
id(s5): 61207680
id(s6): 30793936
id(s7): 30793936
```

- Considering above example string literals that are assigned to new variable won't create new object.

- Let's take real life example, if we need to create student information system which requires following information of student.
  - Student name
  - Student email id
  - Student address
    - Society/Apartment Number and Name
    - Street
    - City
    - State
    - Country
    - Pin code
  - Mobile number
  - Branch/Department
  - Semester
  - Gender
  - Date of Birth
  - ...
- Now assume that this application stores information for 50000 students. From 50000, let's assume 20000 students are from Gujarat.
- So here python create single string object with value Gujarat to store information of all 20000 students.
- Let's assume one of the student change his/her address from Gujarat to Delhi. In this case python is not going to make change in existing object having value Gujrat to Delhi. Python will create new object or if Delhi is already created then give reference of that object to the student who want to change his address from Gujarat to Delhi.
- So we can consider that python string object is immutable. Once it is created it cannot be changed.