Fundamentals of Python Programming

Introduction to python programming

- High level, interpreted language
- Object-oriented
- General purpose
  - Web development (like: Django and Bottle),
  - Scientific and mathematical computing (Orange, SciPy, NumPy)
  - Desktop graphical user Interfaces (Pygame, Panda3D).

- Other features of python includes the following:
  - Simple language which is easier to learn
  - Free and open source
  - Portability
  - Extensible and embeddable
  - Standard large library

- Created by Guido van Rossum in 1991
- Why the name Python?
  - Not after a dangerous snake.
  - Rossum was fan of a comedy series from late seventies.
  - The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Reasons to Choose Python as First Language

- Simple Elegant Syntax
- Not overly strict
- Expressiveness of the language
- Great Community and Support

Installation of Python in Windows

- Go to Download Python page on the official site (https://www.python.org/downloads/) and click Download Python 3.6.5 (You may see different version name)
- When the download is completed, double-click the file and follow the instructions to install it.
- When Python is installed, a program called IDLE is also installed along with it. It provides graphical user interface to work with Python.

- Open IDLE, Write the following code below and press enter.

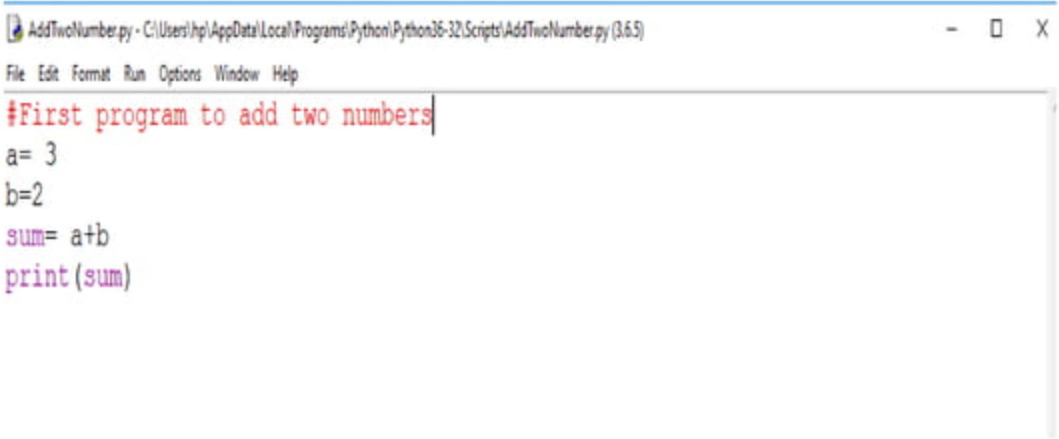
```
print("Hello, World!")
```

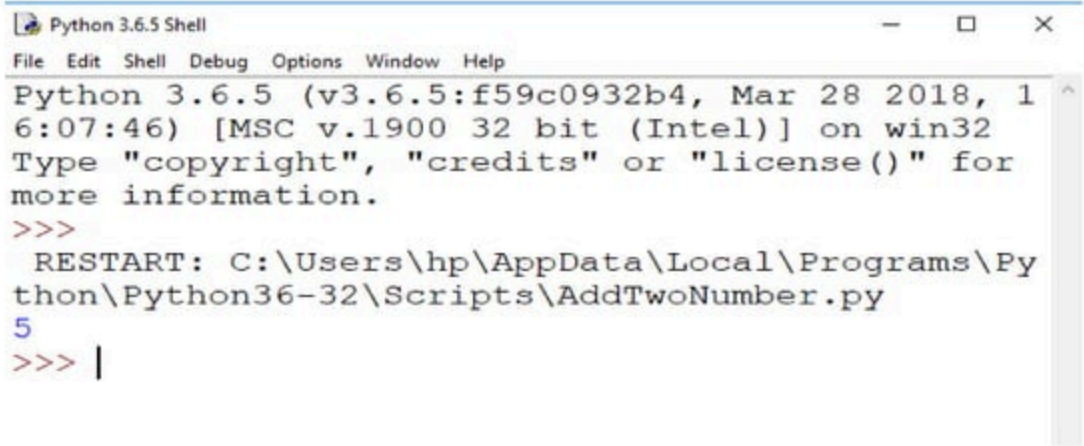
- To create a file in IDLE, go to File > New Window (Shortcut: Ctrl+N).
- Write Python code and save (Shortcut: Ctrl+S) with .py file extension like: hello.py or your-first-program.py

```
print("Hello, World!")
```

- Go to Run > Run module (Shortcut: F5) and you can see the output.

First Python Program



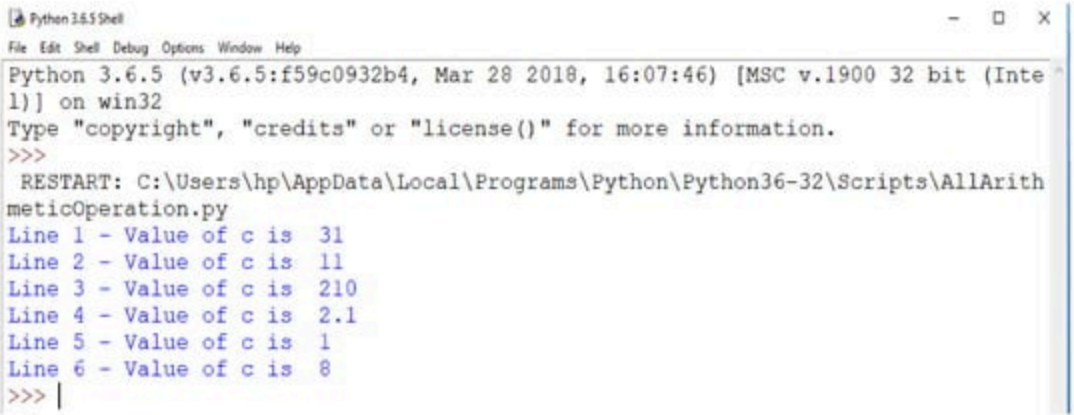

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
6:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>>
RESTART: C:\Users\hp\AppData\Local\Programs\Py
thon\Python36-32\Scripts\AddTwoNumber.py
5
>>>

Second Program

AllArithmeticOperation.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/AllArithmeticOperation.py (3.6.5)

File Edit Format Run Options Window Help

```
#Program to illustrate all the arithmetic operations

a = 21
b = 10
c = 0

c = a + b
print ("Line 1 - Value of c is ", c)

c = a - b
print ("Line 2 - Value of c is ", c )

c = a * b
print ("Line 3 - Value of c is ", c )

c = a / b
print ("Line 4 - Value of c is ", c )

c = a % b
print ("Line 5 - Value of c is ", c)

a = 2
b = 3
c = a**b
print ("Line 6 - Value of c is ", c)
```


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\hp\AppData\Local\Programs\Python\Python36-32\Scripts\AllArith
meticOperation.py
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2.1
Line 5 - Value of c is 1
Line 6 - Value of c is 8
>>>

Few Important Things to Remember

- To represent a statement in Python, newline (enter) is used.
- Use of semicolon at the end of the statement is optional (unlike languages like C/C++)
- In fact, it's recommended to omit semicolon at the end of the statement in Python
- Instead of curly braces { }, indentations are used to represent a block

Python Keywords

- Keywords are the reserved words(can not be used as a identifier) in Python
- Are case sensitive

Keywords in Python programming language

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

Python Identifiers

- Name given to entities like class, functions, variables etc. in Python
- Rules for writing identifiers
  - Can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_)
  - myClass, var_1 and print_this_to_screen, all are valid examples
  - Cannot start with a digit
  - 1variable is invalid, but variable1 is perfectly fine
  - Keywords cannot be used as identifiers
  - Cannot use special symbols like !, @, #, $, % etc. in our identifier
  - Can be of any length.

Things to care about

- Python is a case-sensitive language.
  - Variable and variable are not the same.
- Multiple words can be separated using an underscore, this_is_a_long_variable
- Can also use camel-case style of writing, i.e., capitalize every first letter of the word except the initial word without any spaces
  - camelCaseExample

Python Statement

- Instructions that a Python interpreter can execute are called statements.
- Two types
  - Single line statement:
    - For example, a = 1 is an assignment statement
  - Multiline statement:
    - In Python, end of a statement is marked by a newline character.
    - So, how to make multiline statement?

- Technique1:
  - make a statement extend over multiple lines with the line continuation character (\).
  - For example:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

- Technique2:
  - make a statement extend over multiple lines with the parentheses ( )
  - For example:

$$a = (1 + 2 + 3 +$$

$$4 + 5 + 6 +$$

$$7 + 8 + 9)$$

- Technique3:
  - make a statement extend over multiple lines with the brackets [ ] and braces { }.
  - For example:

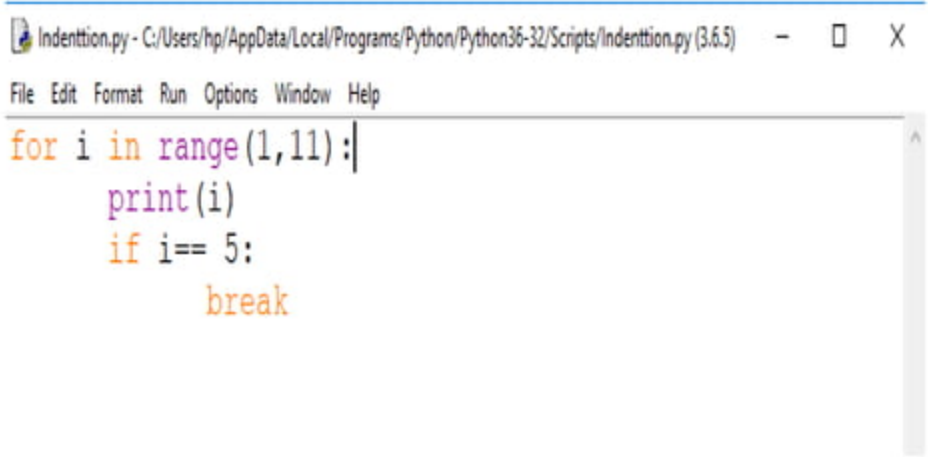
```
colors = ['red',
          'blue',
          'green']
```

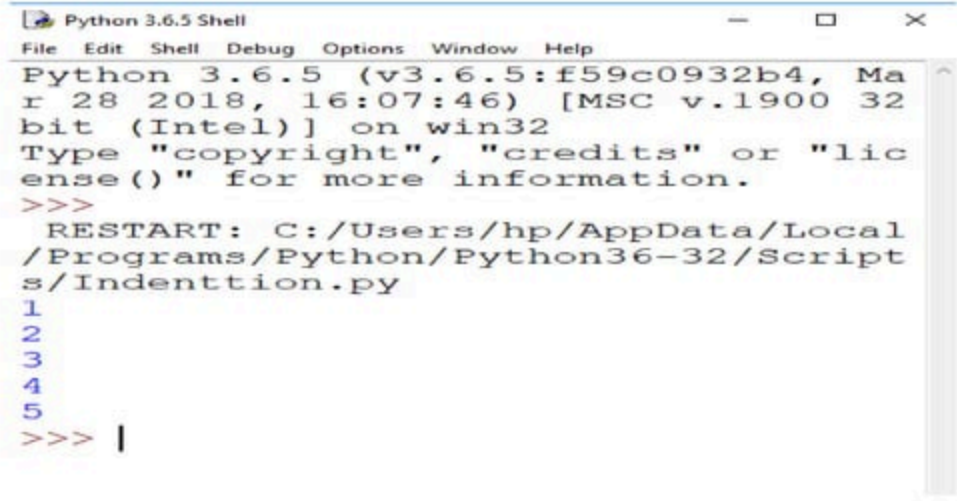
- We could also put multiple statements in a single line using semicolons, as follows

```
a = 1; b = 2; c = 3
```

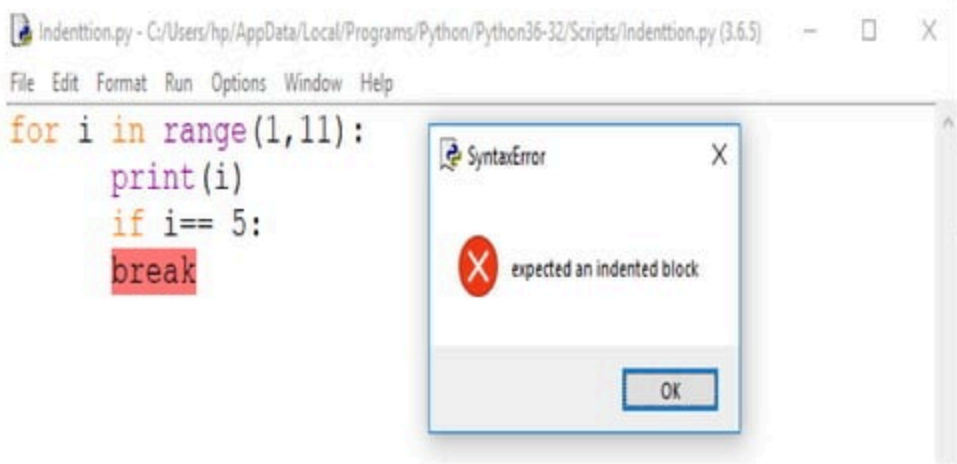
Python Indentation

- Most of the programming languages like C, C++, Java use braces { } to define a block of code
- Python uses indentation
- A code block (body of a function, loop etc.) starts with indentation and ends with the first un-indented line


Indenttion.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/Indenttion.py (3.6.5)
File Edit Format Run Options Window Help
for i in range(1,11):
print(i)
if i== 5:
break


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/Indenttion.py
1
2
3
4
5
>>>

Error due to incorrect indentation

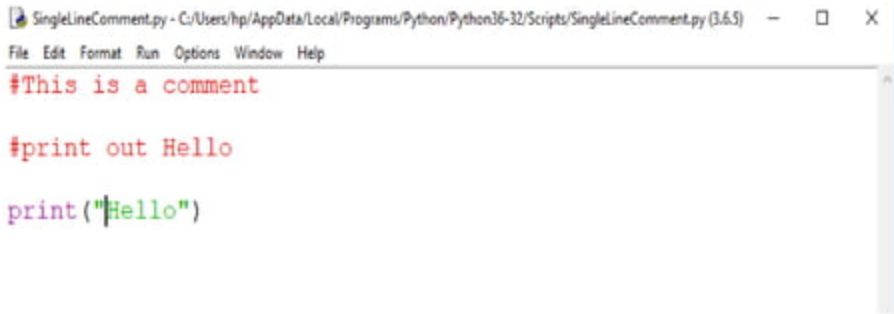


Python Comments

- To make the code much more readable.
- Python Interpreter ignores comment.
- Two types of comment is possible in python:
  - Single line comment and
  - Multi-line comment

Single line comment

- In Python, we use the hash (#) symbol to start writing a comment.
- It extends up to the newline character



Multi-line comments

- Two way:
  - By using # sign at the beginning of each line

```
#This is a long comment
#and it extends
#to multiple lines
```

  - By using either ‘ ‘ ‘ or “ “ “ (most common)

```
"""This is also a
perfect example of
multi-line comments"""
```

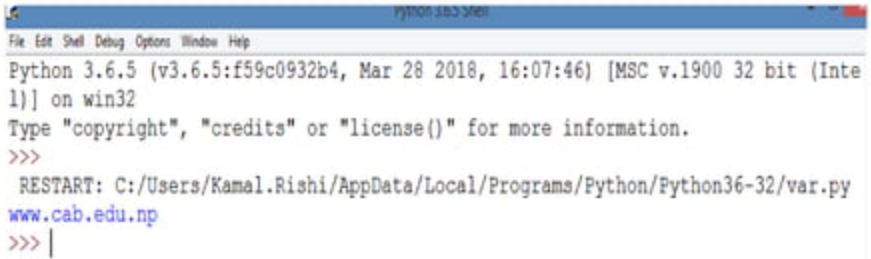
Python Variable

- a variable is a named location used to store data in the memory.
- Alternatively, variables are container that hold data which can be changed later throughout programming

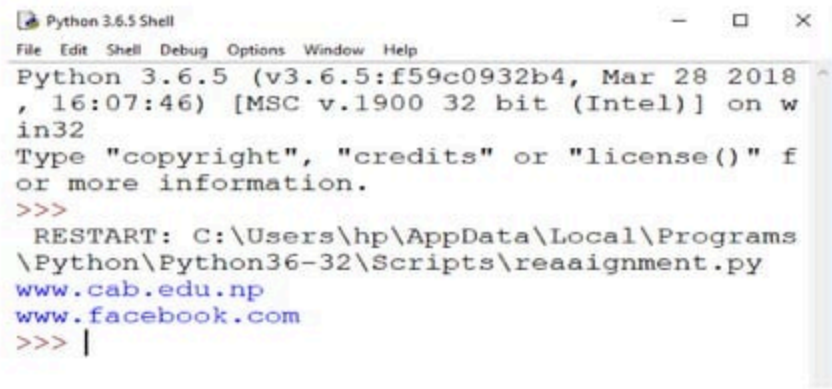
Declaring Variables

- In Python, variables do not need declaration to reserve memory space.
- The "variable declaration" or "variable initialization" happens automatically when we assign a value to a variable.
- We use the assignment operator = to assign the value to a variable.

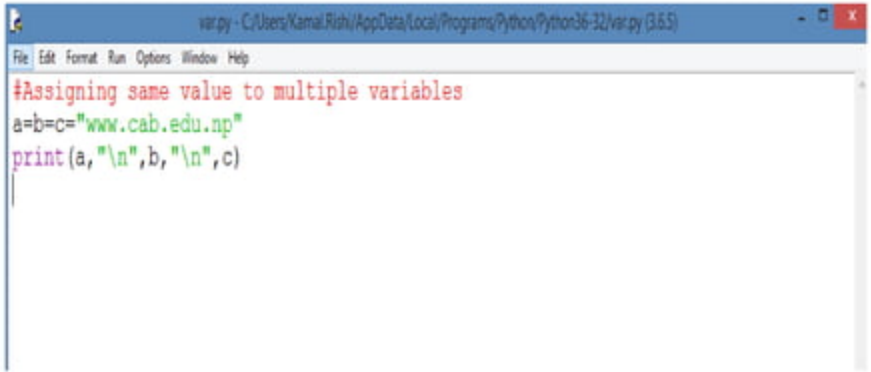
Assigning Values to variables

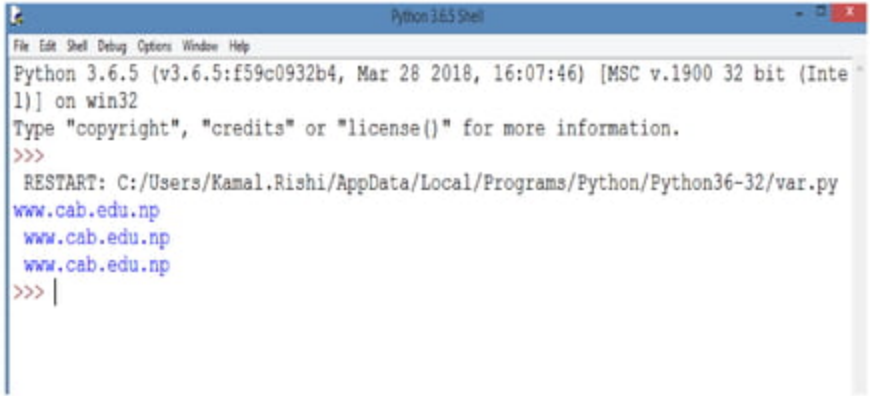

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Kamal.Rishi/AppData/Local/Programs/Python/Python36-32/var.py
www.cab.edu.np
>>>

Changing value of a variable

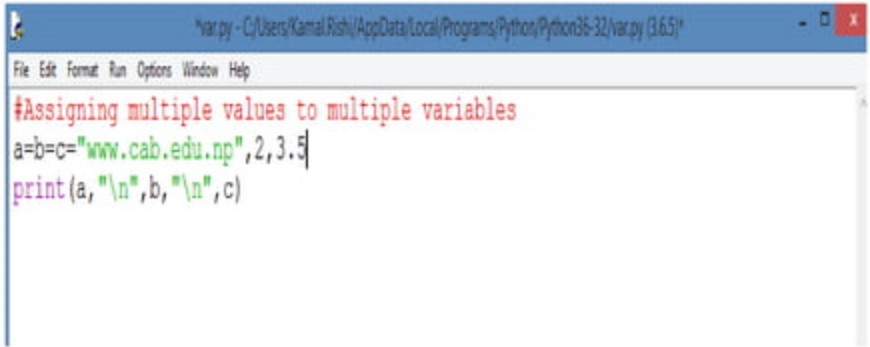

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018
, 16:07:46) [MSC v.1900 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" f
or more information.
>>>
RESTART: C:\Users\hp\AppData\Local\Programs
\Python\Python36-32\Scripts\reaaignment.py
www.cab.edu.np
www.facebook.com
>>>

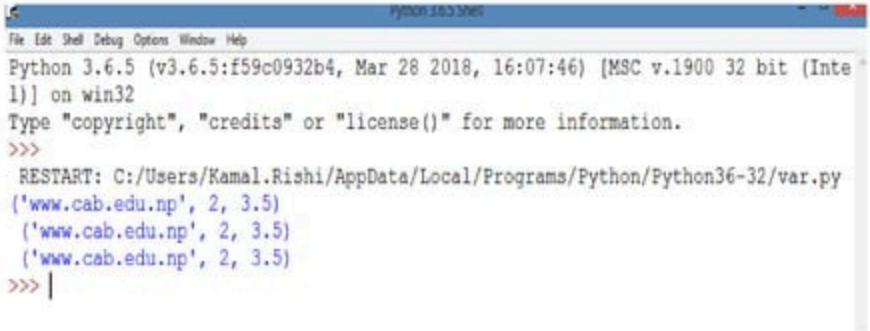
Assigning same value to the multiple variable




Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Kamal.Rishi/AppData/Local/Programs/Python/Python36-32/var.py
www.cab.edu.np
www.cab.edu.np
www.cab.edu.np
>>>

Assigning multiple values to multiple variables

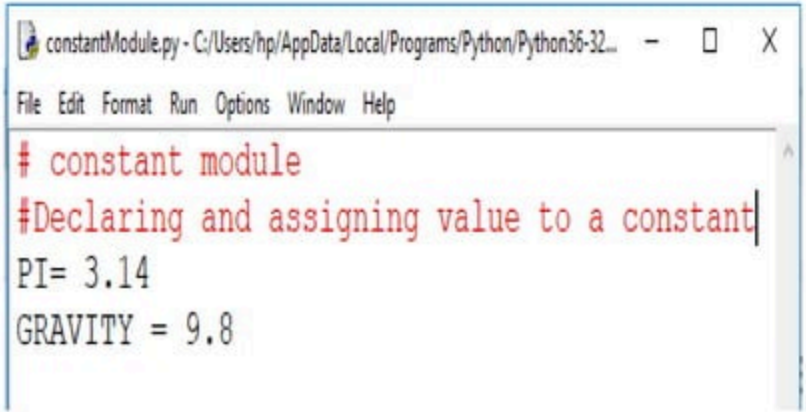


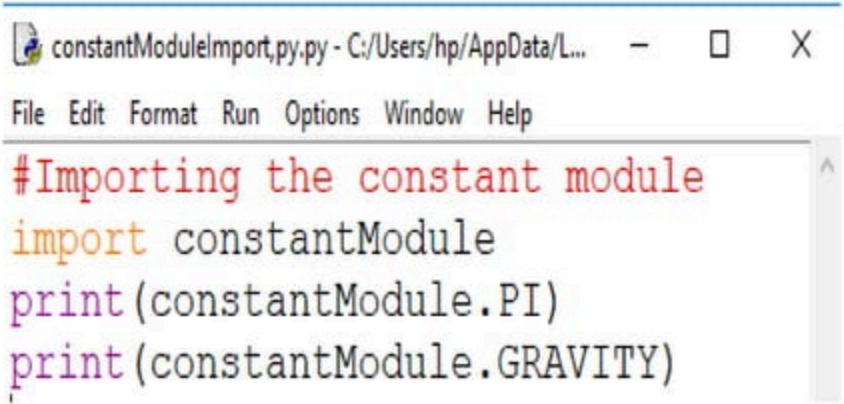

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Kamal.Rishi/AppData/Local/Programs/Python/Python36-32/var.py
('www.cab.edu.np', 2, 3.5)
('www.cab.edu.np', 2, 3.5)
('www.cab.edu.np', 2, 3.5)
>>>

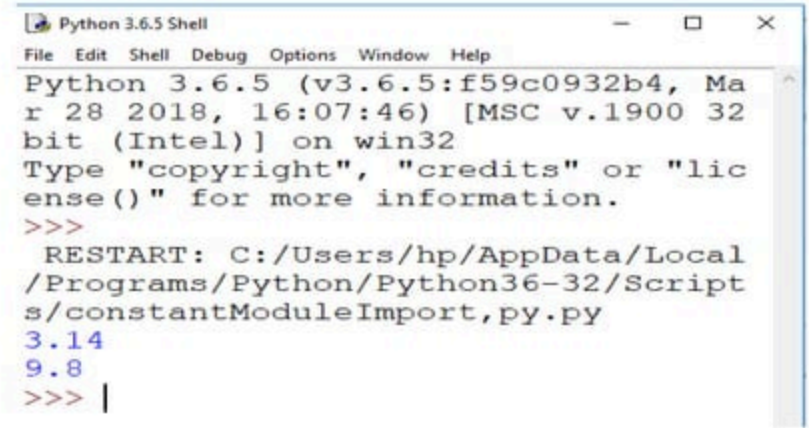
Constants

- Value that cannot be altered by the program during normal execution.
- In Python, constants are usually declared and assigned on a module
- And then imported to the file

Defining constantModule




constantModuleImport.py.py - C:/Users/hp/AppData/L...
File Edit Format Run Options Window Help
#Importing the constant module
import constantModule
print(constantModule.PI)
print(constantModule.GRAVITY)


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/constantModuleImport,py.py
3.14
9.8
>>>

Rules and Naming convention for variables and constants

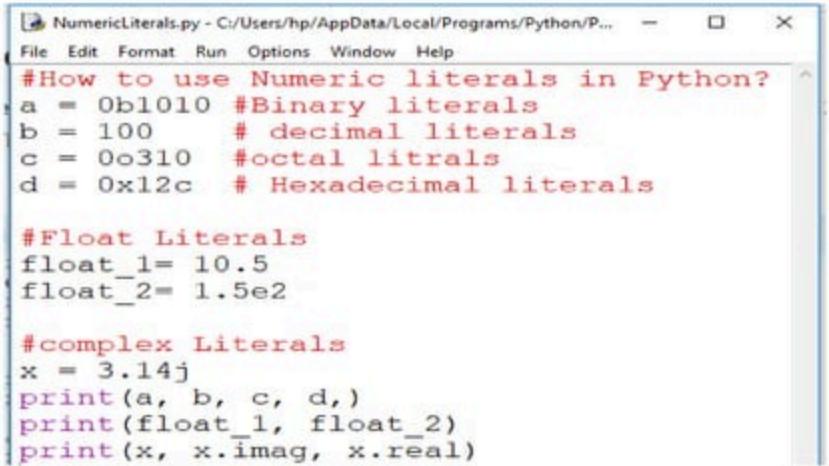
- Create a name that makes sense. Suppose, vowel makes more sense than v.
- Use camelCase notation to declare a variable
- For example: myName, myAge, myAddress
- Use capital letters where possible to declare a constant
- For example: PI,G,MASS etc.

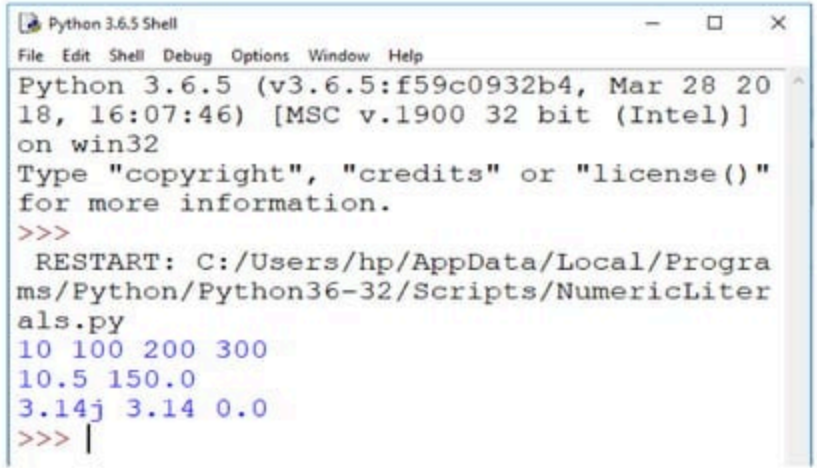
- Never use special symbols like !, @, #, $, %, etc.
- Don't start name with a digit.
- Constants are put into Python modules
- Constant and variable names should have combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- For example: snake_case, MACRO_CASE, camelCase, CapWords

Literals

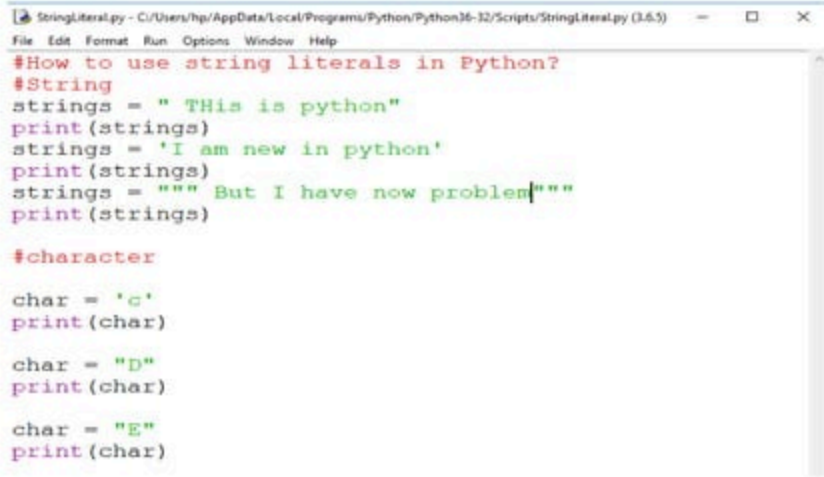
- Literal is a raw data given in a variable or constant. In Python, there are various types of literals they are as follows:
- Numeric literals
- String literals
- Boolean literals
- Special literals

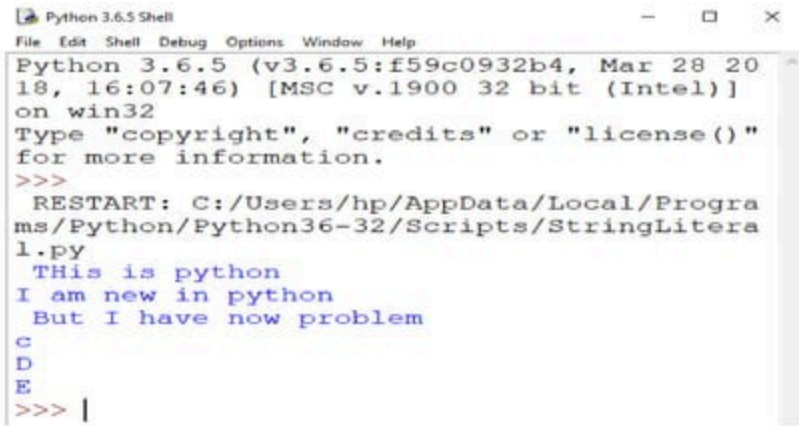
Numeric Literals



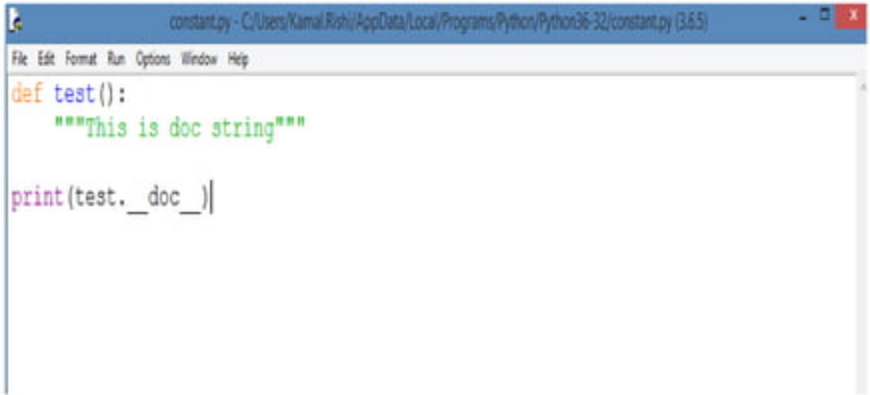

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 20
18, 16:07:46) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()"
for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Progra
ms/Python/Python36-32/Scripts/NumericLiter
als.py
10 100 200 300
10.5 150.0
3.14j 3.14 0.0
>>>

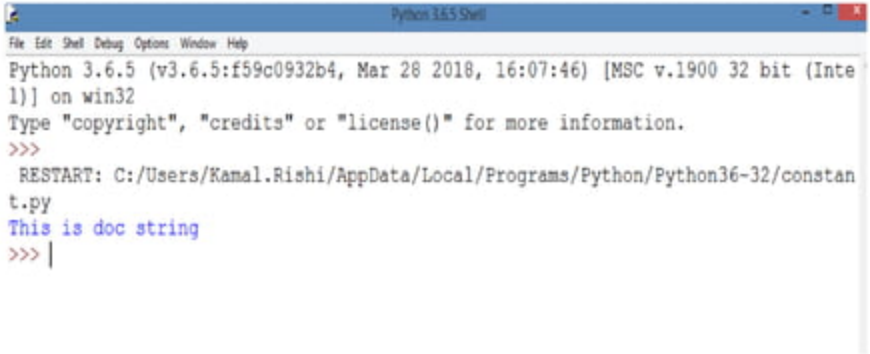
String Literals



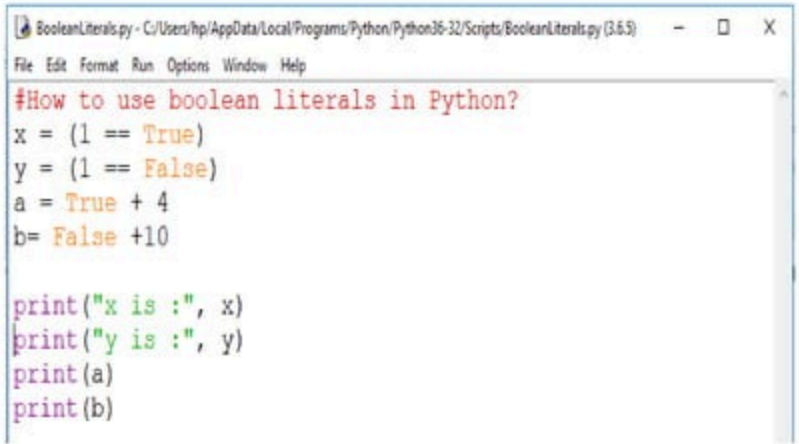

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 20
18, 16:07:46) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()"
for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Progra
ms/Python/Python36-32/Scripts/StringLitera
l.py
THis is python
I am new in python
But I have now problem
c
D
E
>>>

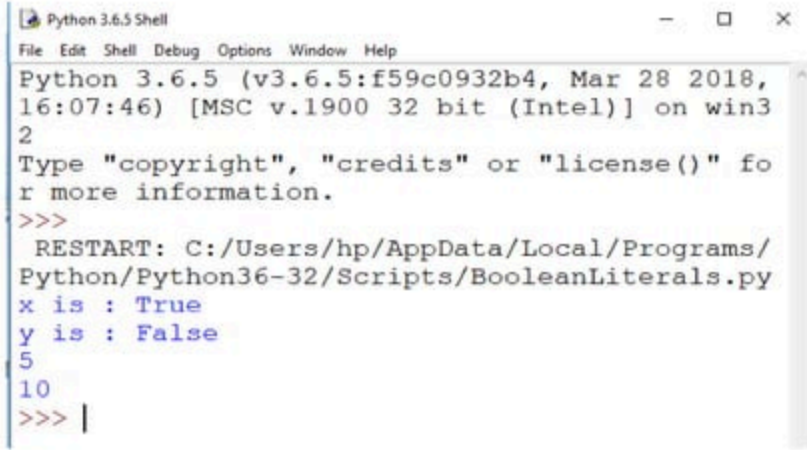
Docstrings


constant.py - C:/Users/Kamal.Rishi/AppData/Local/Programs/Python/Python36-32/constant.py (3.6.5)
File Edit Format Run Options Window Help
def test():
"""This is doc string"""
print(test.__doc__)

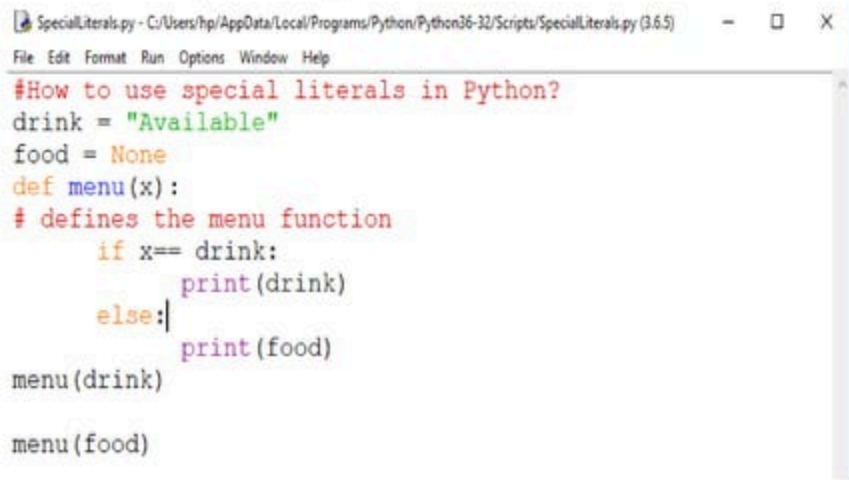

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Kamal.Rishi/AppData/Local/Programs/Python/Python36-32/constan
t.py
This is doc string
>>>

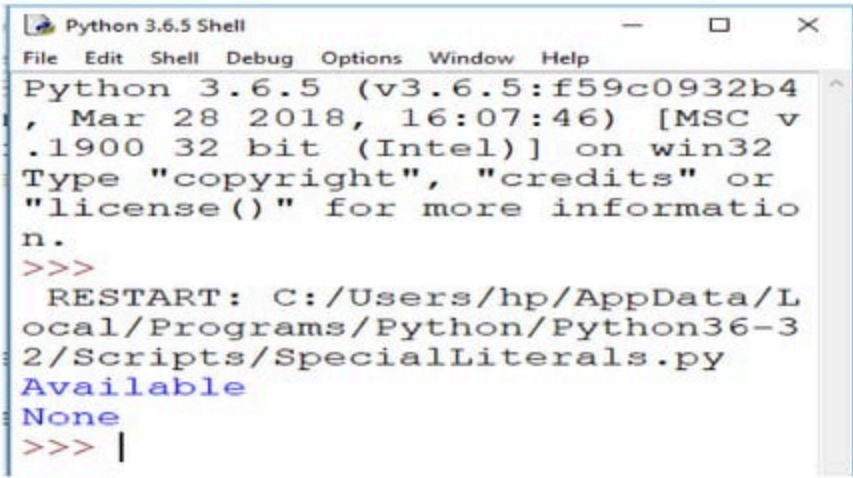
Using boolean literals



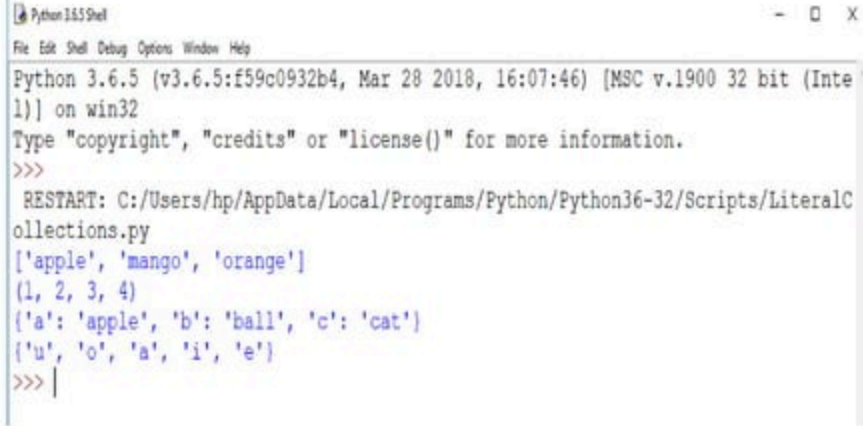

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
16:07:46) [MSC v.1900 32 bit (Intel)] on win3
2
Type "copyright", "credits" or "license()" fo
r more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/
Python/Python36-32/Scripts/BooleanLiterals.py
x is : True
y is : False
5
10
>>>

Special literal(None)




Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4
, Mar 28 2018, 16:07:46) [MSC v
.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or
"license()" for more informatio
n.
>>>
RESTART: C:/Users/hp/AppData/L
ocal/Programs/Python/Python36-3
2/Scripts/SpecialLiterals.py
Available
None
>>>

Literals Collection


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/LiteralC
ollections.py
['apple', 'mango', 'orange']
(1, 2, 3, 4)
{'a': 'apple', 'b': 'ball', 'c': 'cat'}
{'u', 'o', 'a', 'i', 'e'}
>>>

Data Types

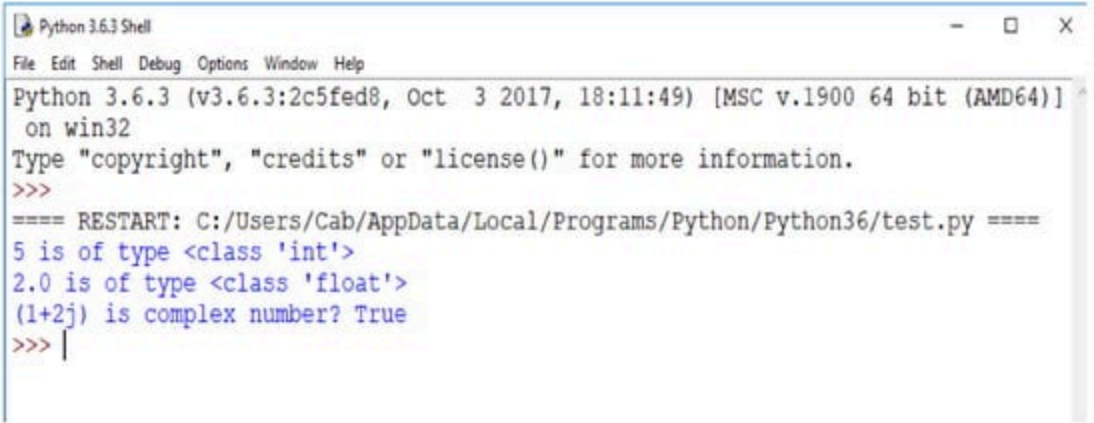
- In python data types are classes and variables are instance (object) of these classes
- Different types of data types in python are:
  - Python numbers
  - Python lists
  - Python tuples
  - Python strings
  - Python sets
  - Python dictionary

Python Numbers

test.py - C:/Users/Cab/AppData/Local/Programs/Python/Python36/test.py (3.6.3)

File Edit Format Run Options Window Help

```
a = 5
print(a, "is of type", type(a))

a = 2.0
print(a, "is of type", type(a))

a = 1+2j
print(a, "is complex number?", isinstance(a,complex))
```


Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Cab/AppData/Local/Programs/Python/Python36/test.py ====
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is complex number? True
>>>

Python List

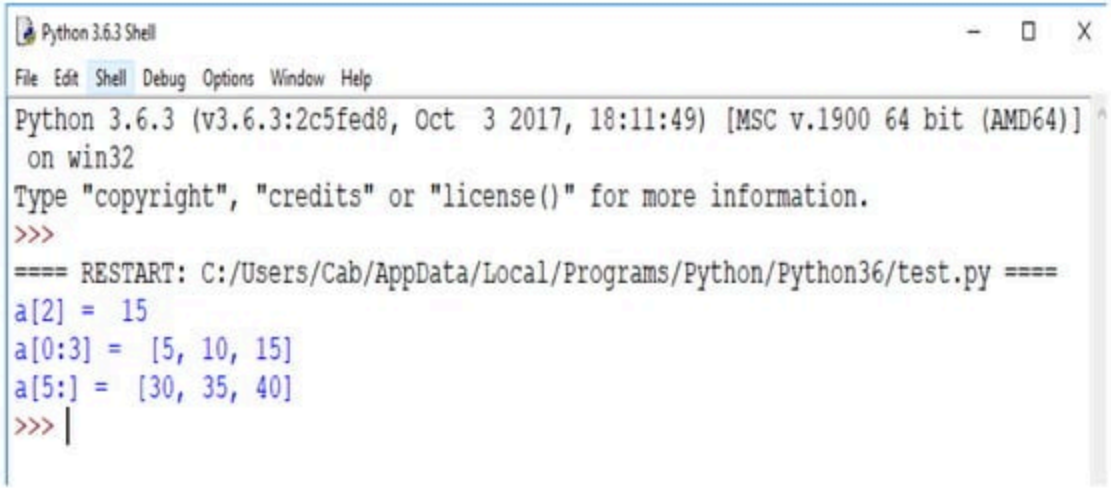
- Ordered sequence of items
- Can contain heterogeneous data
- Syntax:
  - Items separated by commas are enclosed within brackets [ ]
  - Example: a= [1,2.2,'python']

Extracting elements from the list

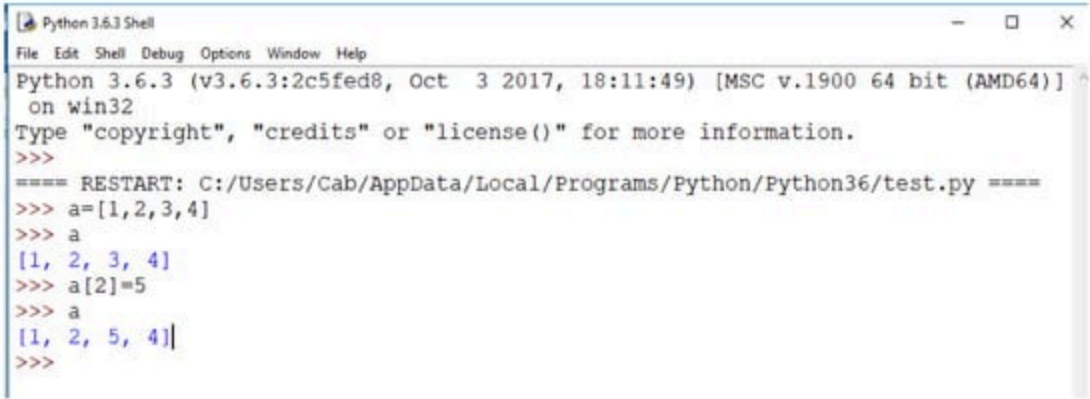
test.py - C:/Users/Cab/AppData/Local/Programs/Python/Python36/test.py (3.6.3)

File Edit Format Run Options Window Help

```
a = [5,10,15,20,25,30,35,40]

print("a[2] = ", a[2])

print("a[0:3] = ", a[0:3])

print("a[5:] = ", a[5:])
```

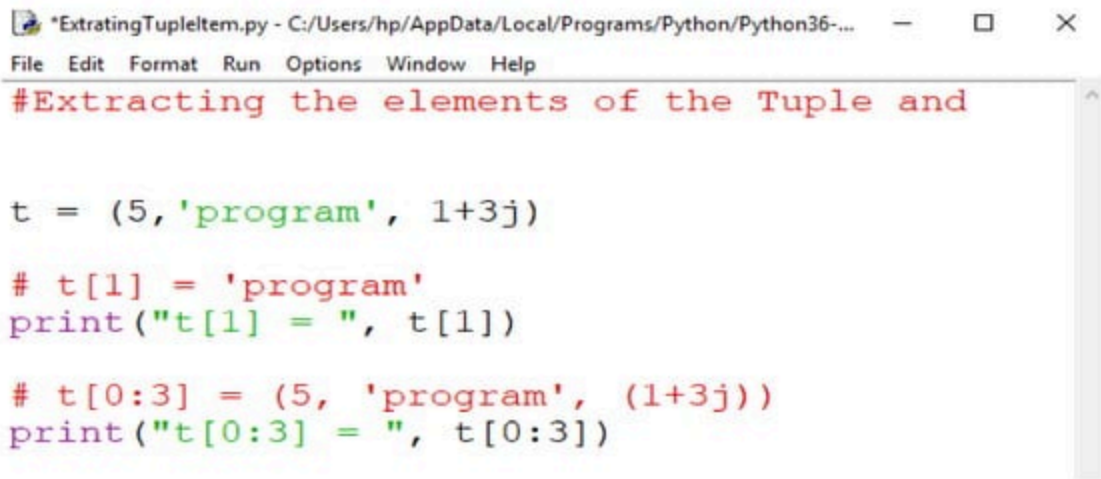

Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Cab/AppData/Local/Programs/Python/Python36/test.py ====
a[2] = 15
a[0:3] = [5, 10, 15]
a[5:] = [30, 35, 40]
>>>

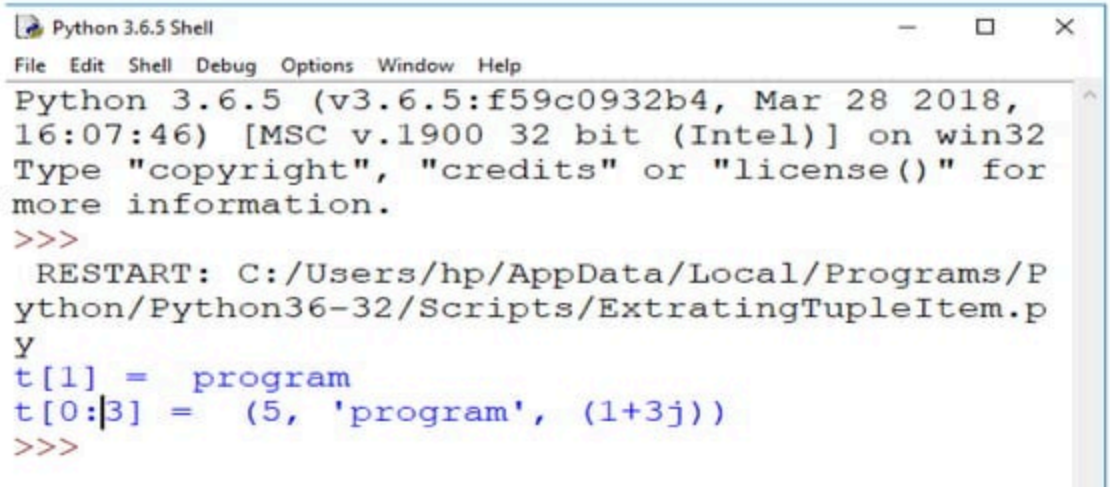
- Note: Lists are mutable, meaning, value of elements of a list can be altered.

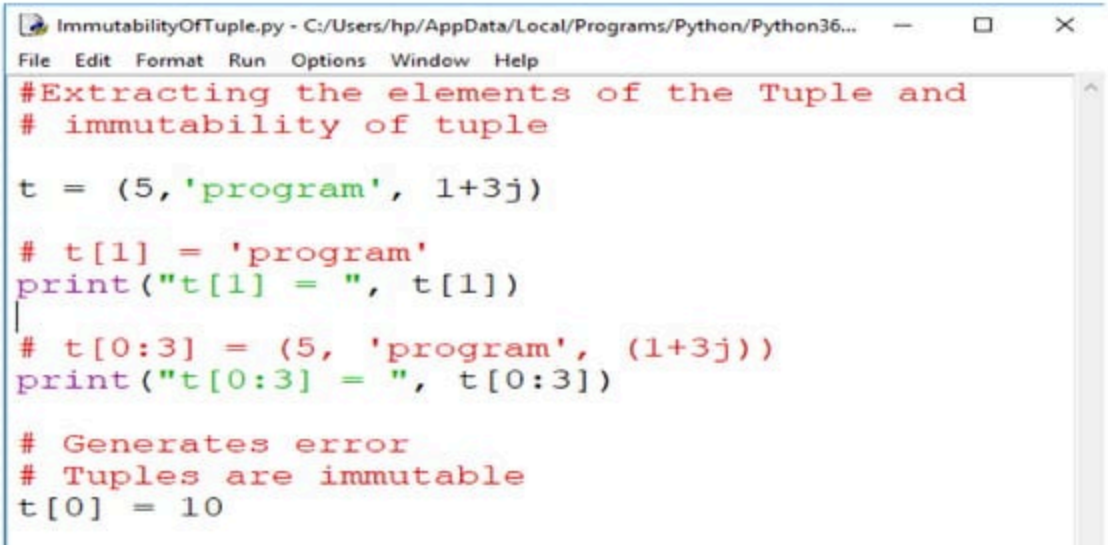


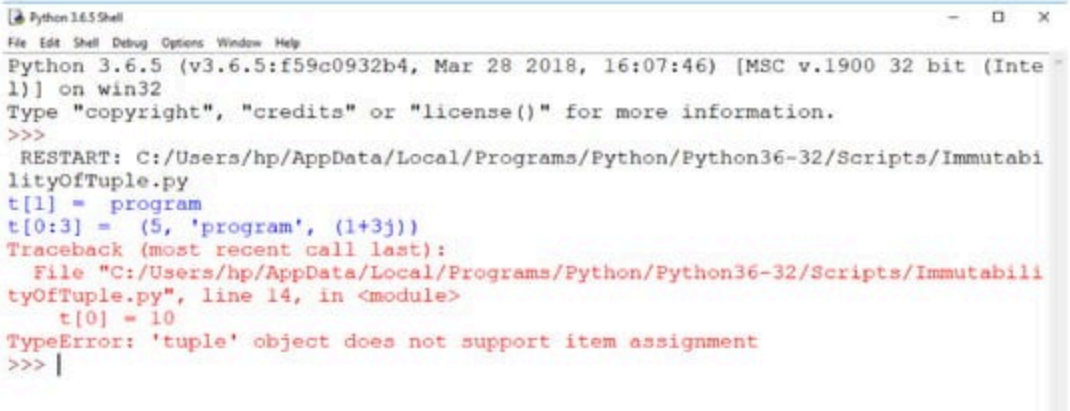
Python Tuple

- Same as list but immutable.
- Used to write-protect the data
- Syntax:
  - Items separated by commas are enclosed within brackets ( )
- Example:
  - t = (5,'program', 1+3j)


*ExtratingTupleItem.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-...
File Edit Format Run Options Window Help
#Extracting the elements of the Tuple and
t = (5,'program', 1+3j)
# t[1] = 'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/ExtratingTupleItem.py
t[1] = program
t[0:3] = (5, 'program', (1+3j))
>>>


ImmutabilityOfTuple.py - C:/Users/hp/AppData/Local/Programs/Python/Python36...
File Edit Format Run Options Window Help
#Extracting the elements of the Tuple and
# immutability of tuple
t = (5,'program', 1+3j)
# t[1] = 'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
# Generates error
# Tuples are immutable
t[0] = 10

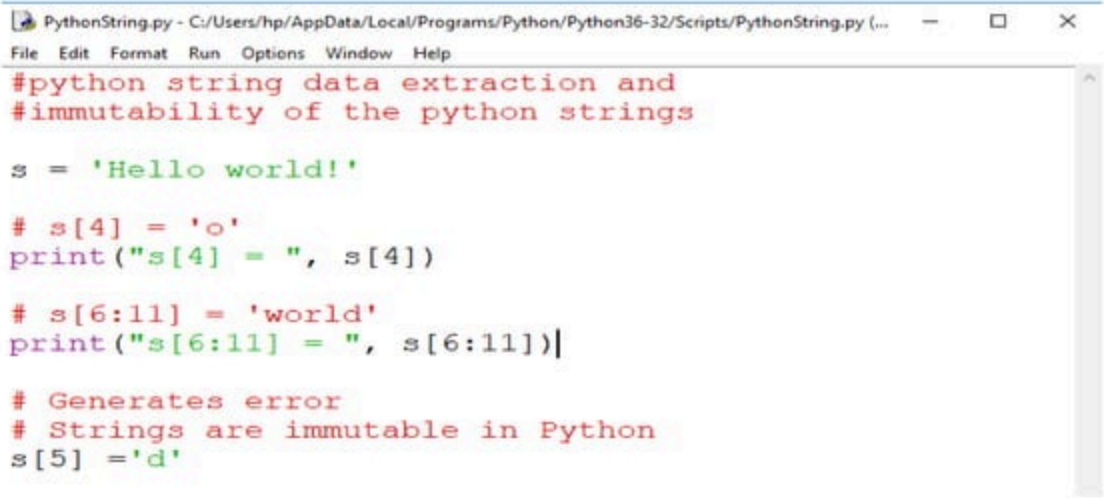

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/Immutabi
lityOfTuple.py
t[1] = program
t[0:3] = (5, 'program', (1+3j))
Traceback (most recent call last):
File "C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/Immutabili
tyOfTuple.py", line 14, in <module>
t[0] = 10
TypeError: 'tuple' object does not support item assignment
>>>

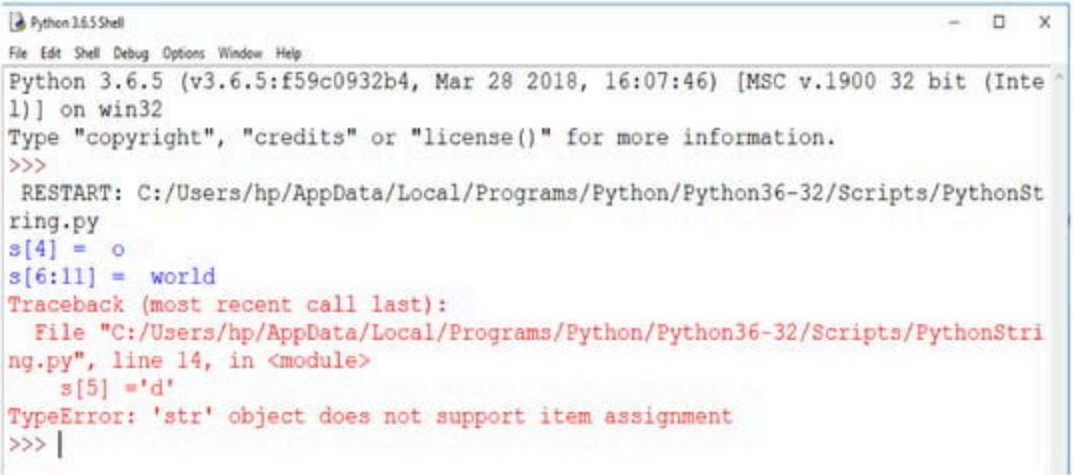
Python Strings

- use single quotes or double quotes to represent strings.
- Multi-line strings can be denoted using triple quotes, ''' or """.
- Example:

```
s = "This is a string"
s = '''a multiline
        string'''
```

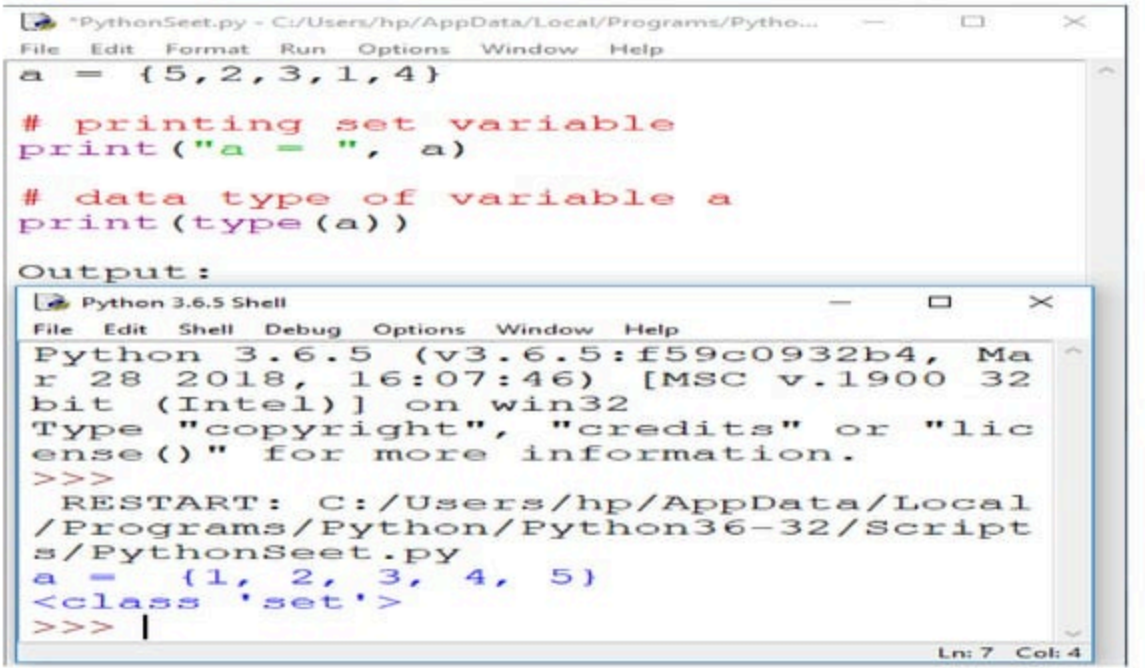
- Strings are also immutable.


PythonString.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/PythonString.py (...
File Edit Format Run Options Window Help
#python string data extraction and
#immutability of the python strings
s = 'Hello world!'
# s[4] = 'o'
print("s[4] = ", s[4])
# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])
# Generates error
# Strings are immutable in Python
s[5] ='d'

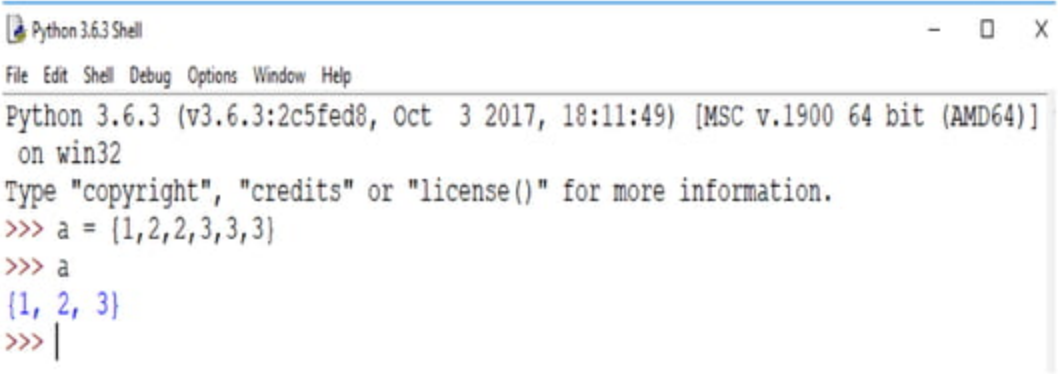

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/PythonSt
ring.py
s[4] = o
s[6:11] = world
Traceback (most recent call last):
File "C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/PythonStri
ng.py", line 14, in <module>
s[5] ='d'
TypeError: 'str' object does not support item assignment
>>>

Python Set

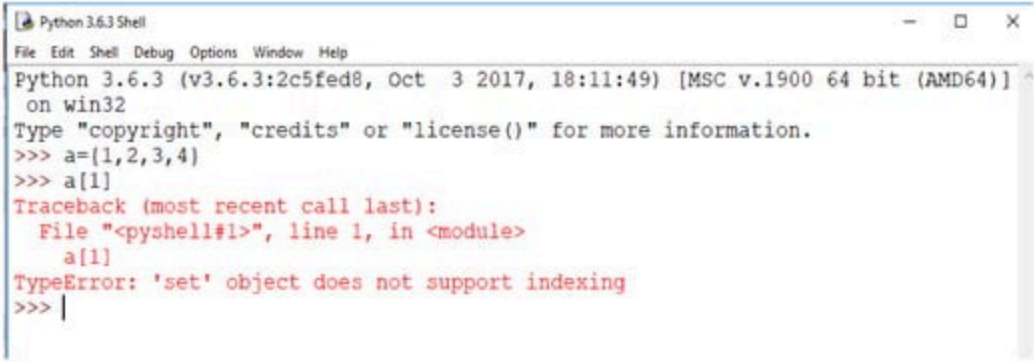
- Unordered collection of unique items
- defined by values separated by comma inside braces { }


*PythonSeet.py - C:/Users/hp/AppData/Local/Programs/Pytho...
File Edit Format Run Options Window Help
a = {5,2,3,1,4}
# printing set variable
print("a = ", a)
# data type of variable a
print(type(a))
Output:
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/PythonSeet.py
a = {1, 2, 3, 4, 5}
<class 'set'>
>>>
Ln: 7 Col: 4

- Set have unique values. They eliminate duplicates.
- Example:

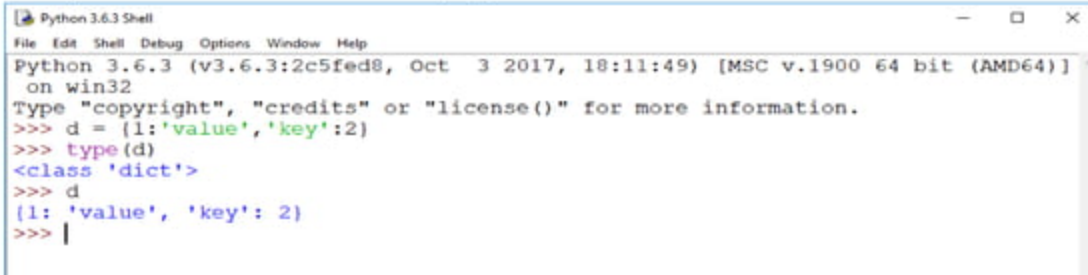


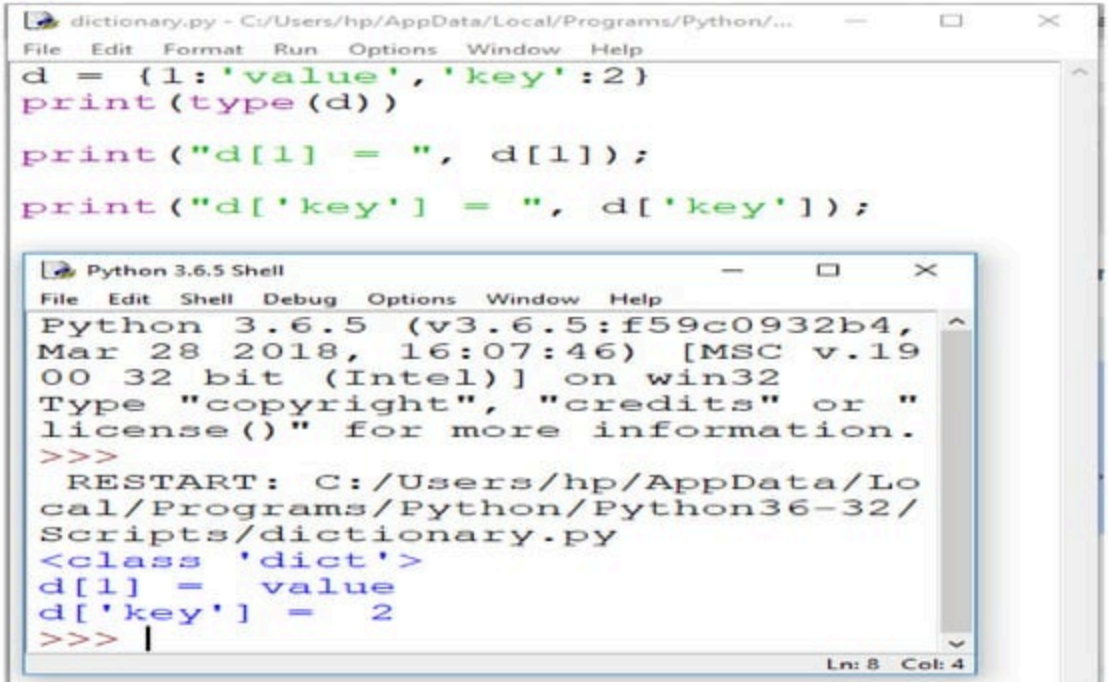
- Since, set are unordered collection, indexing has no meaning.
- Hence the slicing operator [] does not work.
- Example:

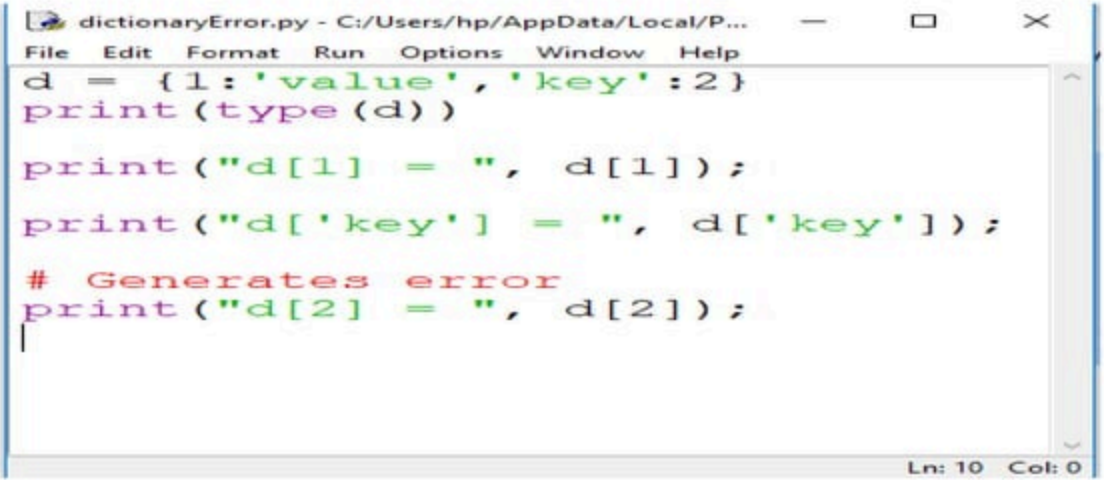


Python Dictionary

- An unordered collection of key-value pairs.
- Must know the key to retrieve the value.
- Are defined within braces {} with each item being a pair in the form key: value
- Key and value can be of any type



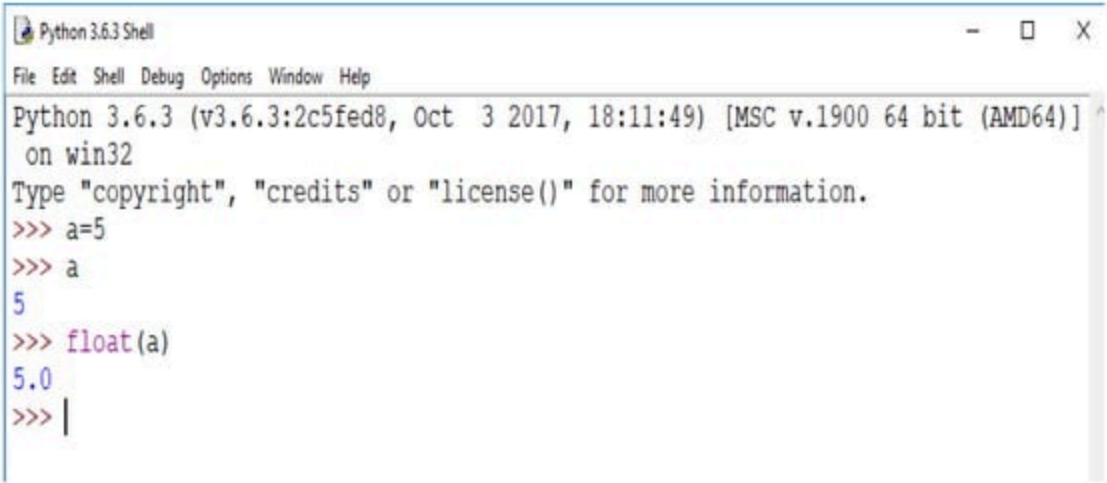

dictionary.py - C:/Users/hp/AppData/Local/Programs/Python/...
File Edit Format Run Options Window Help
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/dictionary.py
<class 'dict'>
d[1] = value
d['key'] = 2
>>>
Ln: 8 Col: 4


dictionaryError.py - C:/Users/hp/AppData/Local/P...
File Edit Format Run Options Window Help
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);
# Generates error
print("d[2] = ", d[2]);
Ln: 10 Col: 0

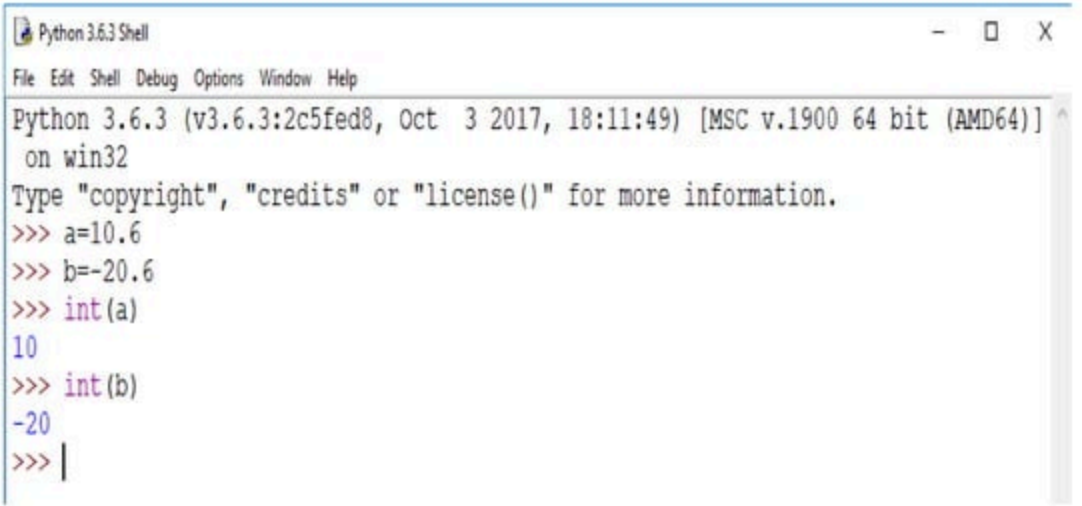
Conversion between data types

- Conversion can be done by using different types of type conversion functions like:
  - int(),
  - float(),
  - str() etc.

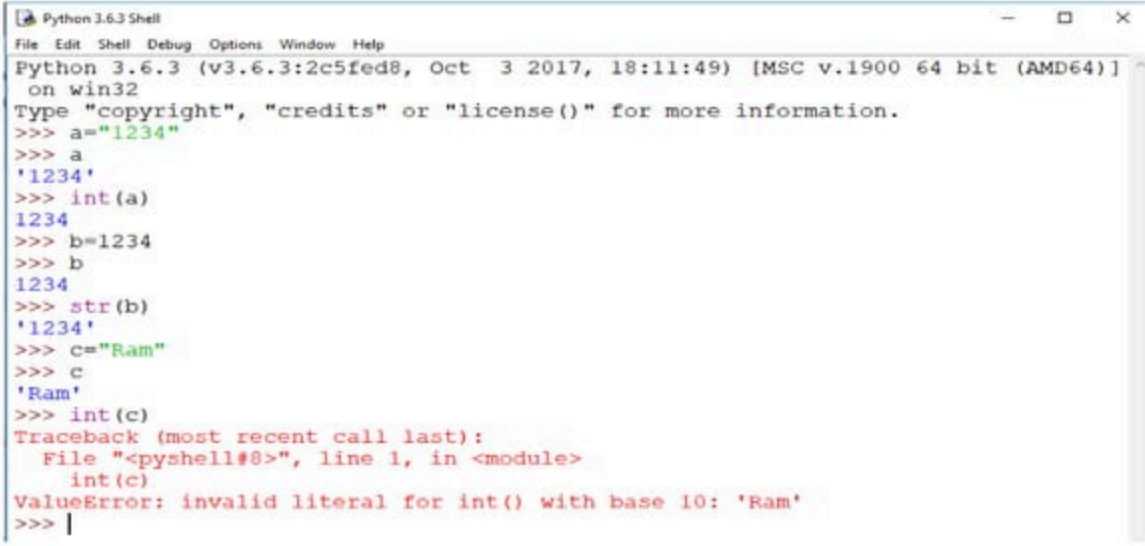
int to float


Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=5
>>> a
5
>>> float(a)
5.0
>>>

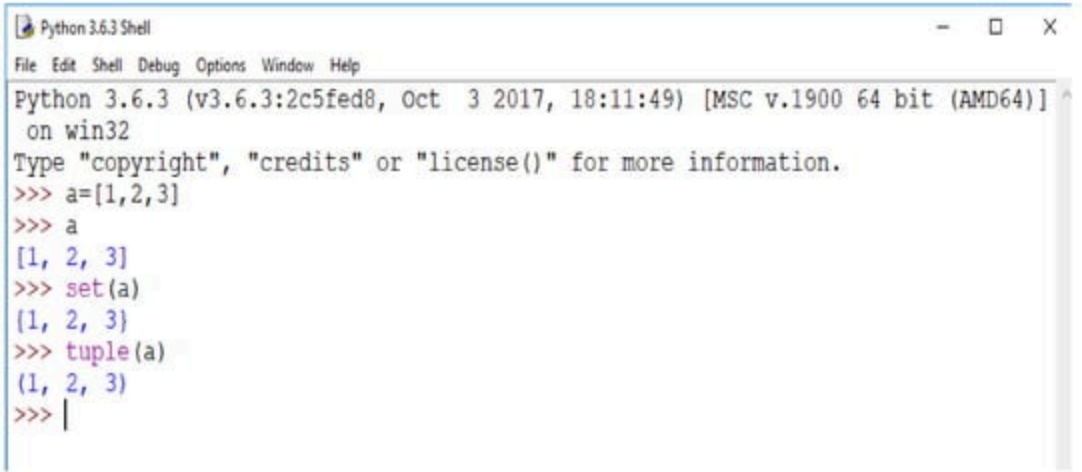
float to int


Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10.6
>>> b=-20.6
>>> int(a)
10
>>> int(b)
-20
>>>

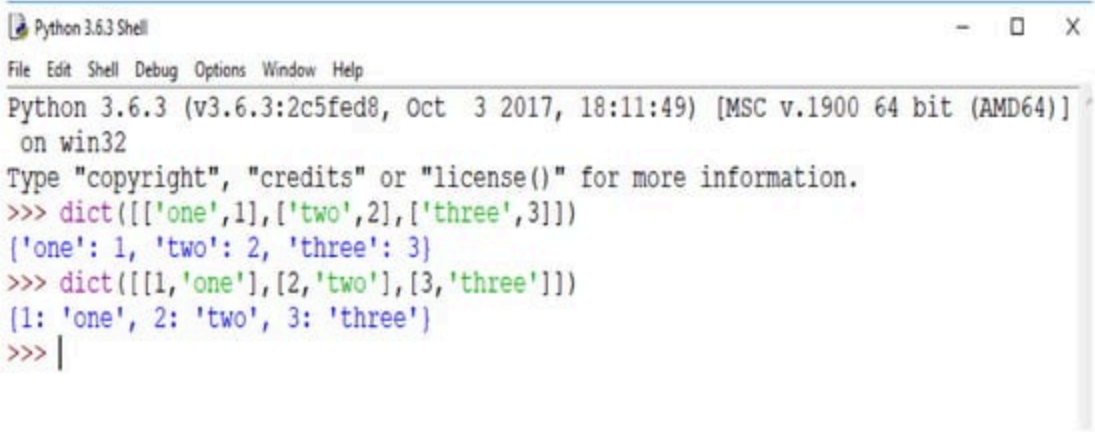
To and form string



convert one sequence to another



- To convert to dictionary, each element must be a pair:



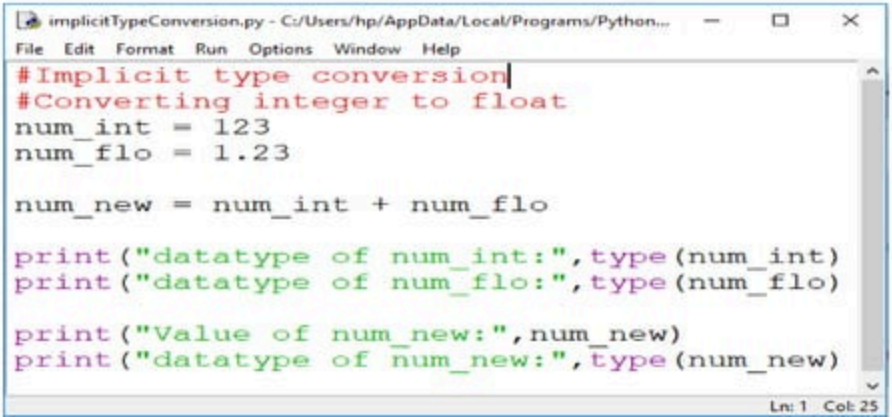
Python Type Conversion and Type Casting

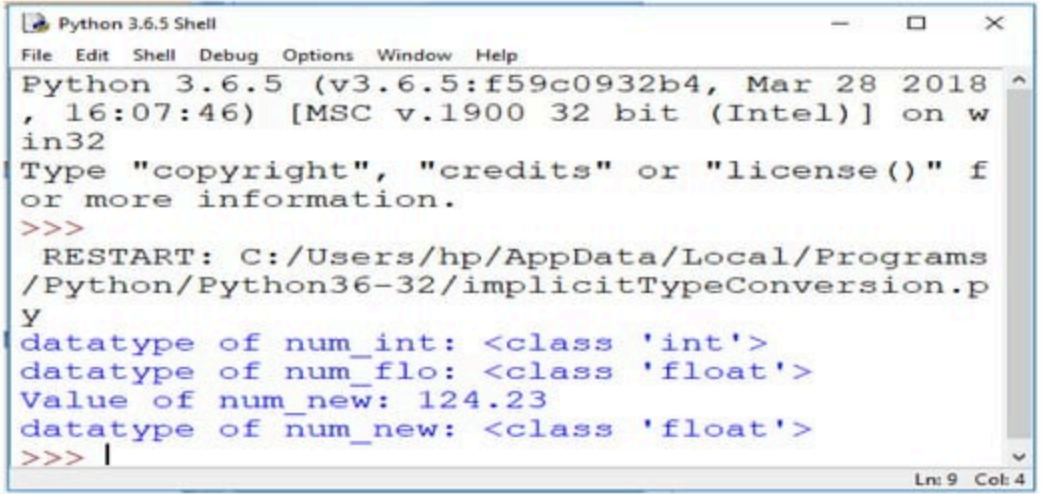
- The process of converting the value of one data type (integer, string, float, etc.) to another data type is type conversion
- Python has two types of type conversion.
  - Implicit Type Conversion
  - Explicit Type Conversion

- Type Conversion is the conversion of object from one data type to another data type
- Implicit Type Conversion is automatically performed by the Python interpreter
- Python avoids the loss of data in Implicit Type Conversion
- Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user
- In Type Casting loss of data may occur as we enforce the object to specific data type

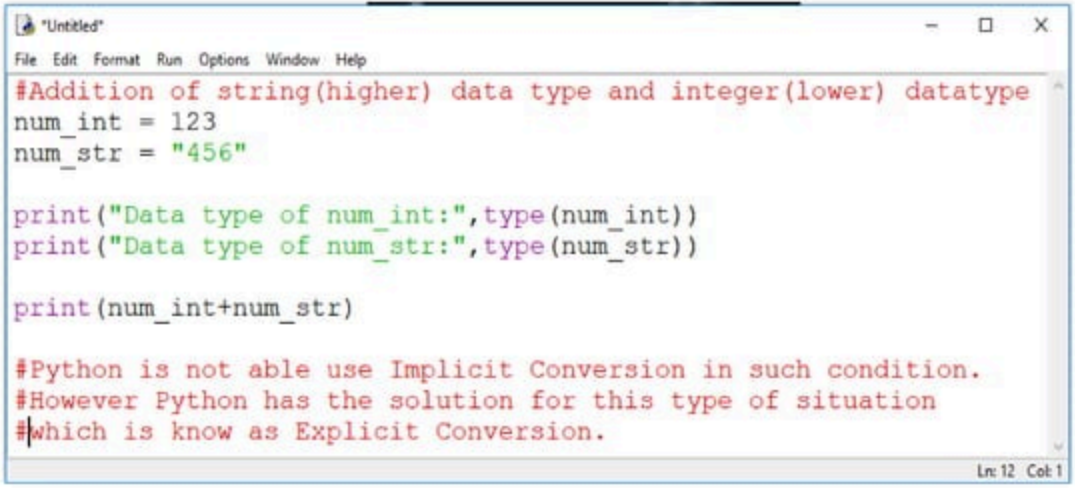
Implicit Type Conversion

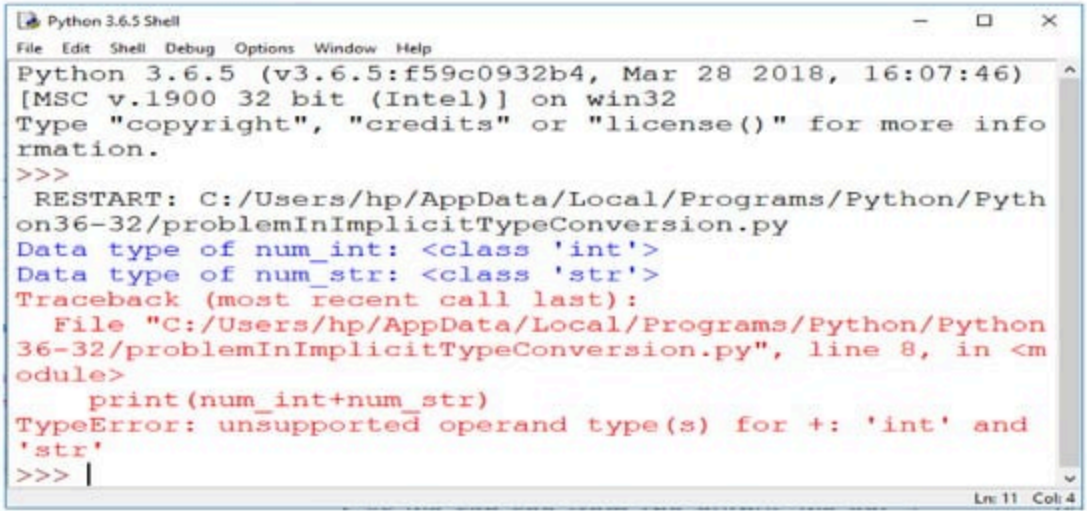
- Automatically converts one data type to another data type
- Always converts smaller data type to larger data type to avoid the loss of data




Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018
, 16:07:46) [MSC v.1900 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" f
or more information.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs
/Python/Python36-32/implicitTypeConversion.p
y
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
>>>
Ln: 9 Col: 4

Problem in implicit type conversion

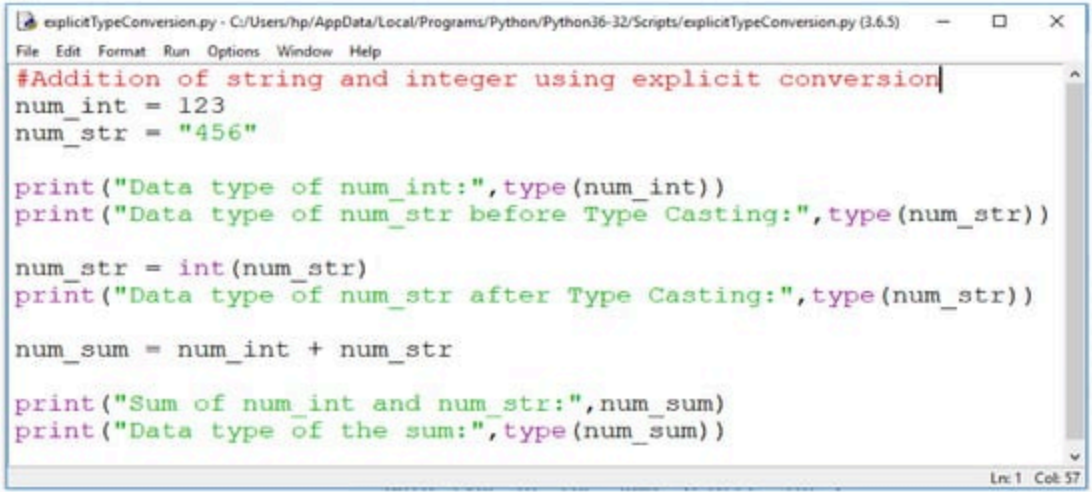


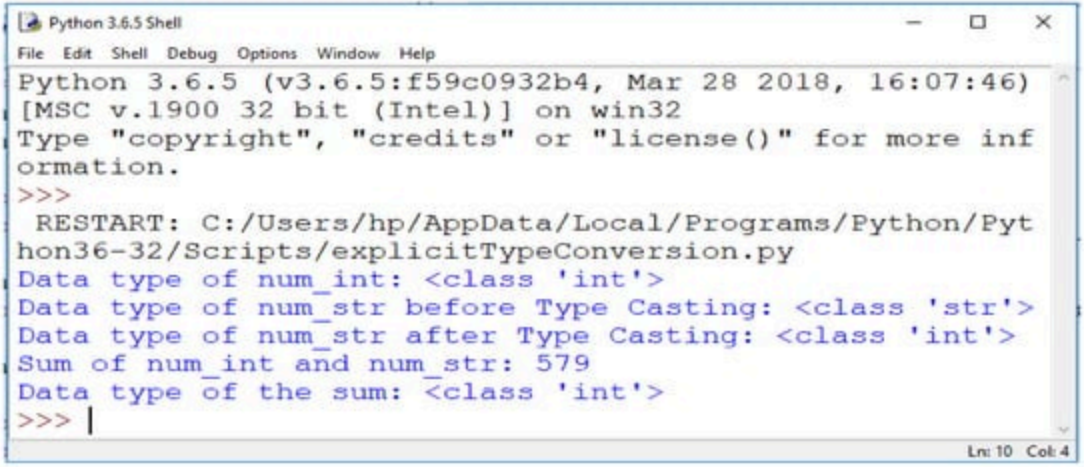

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more info
rmation.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Pyth
on36-32/problemInImplicitTypeConversion.py
Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>
Traceback (most recent call last):
File "C:/Users/hp/AppData/Local/Programs/Python/Python
36-32/problemInImplicitTypeConversion.py", line 8, in <m
odule>
print(num_int+num_str)
TypeError: unsupported operand type(s) for +: 'int' and
'str'
>>>
Ln: 11 Col: 4

Explicit Type Conversion

- Users convert the data type of an object to required data type.
- Predefined functions like int(), float(), str(), etc. are to perform explicit type conversion
- Is also called typecasting because the user casts (change) the data type of the objects.
- Syntax :
  - (required_datatype)(expression)
  - Int(b)

Example

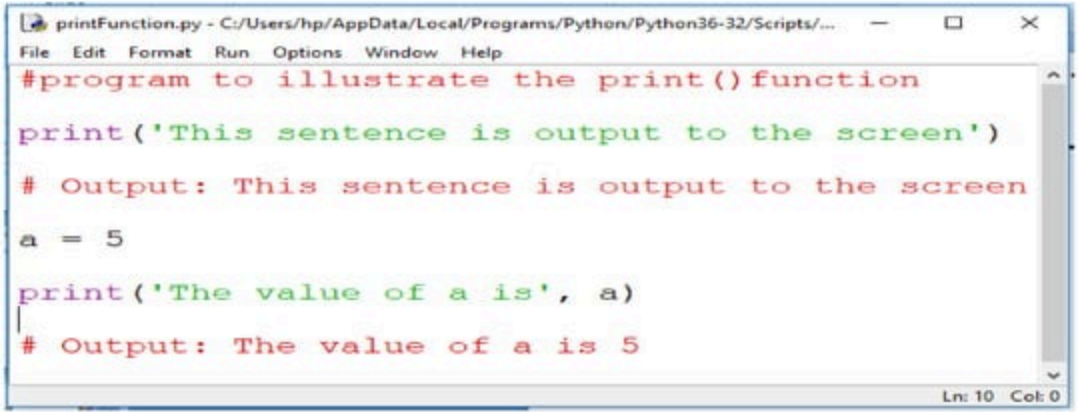

explicitTypeConversion.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/explicitTypeConversion.py (3.6.5)
File Edit Format Run Options Window Help
#Addition of string and integer using explicit conversion
num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
Ln: 1 Col: 57


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inf
ormation.
>>>
RESTART: C:/Users/hp/AppData/Local/Programs/Python/Pyt
hon36-32/Scripts/explicitTypeConversion.py
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
>>>
Ln: 10 Col: 4

Python Input, Output and Import

- Widely used functions for standard input and output operations are:
  - print() and
  - input()

Python Output Using print() function



Output formatting

- Can be done by using the str.format() method
  - Is visible to any string object

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
>>>
```

- Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

Example

formatFunction.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/formatFunc...

File Edit Format Run Options Window Help

```
#program to illustrate the format()function

print('I love {0} and {1}'.format('bread','butter'))

# Output: I love bread and butter

print('I love {1} and {0}'.format('bread','butter'))

# Output: I love butter and bread
```

Ln: 8 Col: 0

```
I love bread and butter
I love butter and bread
>>>
```

- We can even use keyword arguments to format the string

```
>>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'Ram'))
Hello Ram, Goodmorning
>>>
```

- We can even format strings like the old printf() style used in C
- We use the % operator to accomplish this

```
>>> x= 12.3456789
>>> print('the value of x is %3.2f' %x)
the value of x is 12.35
>>> print('the value of x is %3.3f' %x)
the value of x is 12.346
>>>
```

Input() function

- input() function to take the input from the user.
- The syntax for input() is :
  - input([prompt])
  - where prompt is the string we wish to display on the screen.
  - It is optional

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
>>>
```

- Entered value 10 is a string, not a number.
- To convert this into a number we can use int() or float() functions

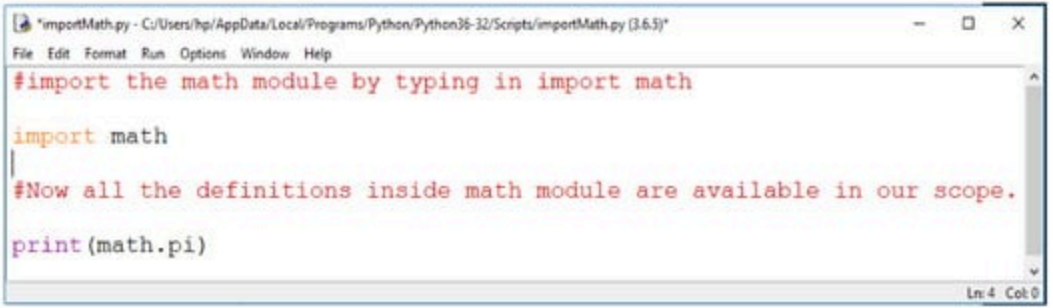
```
>>> int('10')
10
>>> float('10')
10.0
>>>
```

- This same operation can be performed using the eval(). It can evaluate expressions, provided the input is a string.

```
>>> int('2+3')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    int('2+3')
ValueError: invalid literal for int() with base 10: '2+3'
>>> eval('2+3')
5
>>>
```

Python Import

- When our program grows bigger, it is a good idea to break it into different modules.
- A module is a file containing Python definitions and statements.
- Python modules have a filename and end with the extension .py
- Definitions inside a module can be imported to another module or the interactive interpreter in Python
- We use the import keyword to do this




```
3.141592653589793
>>>
```

- import some specific attributes and functions only, using the from keyword.

```
>>> from math import pi
>>> pi
3.141592653589793
>>>
```

Python Operators

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

```
>>> 2+3
5
```

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

Arithmetic operators

Arithmetic operators in Python

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

Example

arithmeticOperator.py - C:/Users/hp/AppData/Loc...

File Edit Format Run Options Window Help

```
# Arithmetic operators in Python
x = 15
y = 4

# Output: x + y = 19
print('x + y =',x+y)

# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3
print('x // y =',x//y)

# Output: x ** y = 50625
print('x ** y =',x**y)
```

Ln: 1 Col: 32

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

Comparison operators

Comparision operators in Python

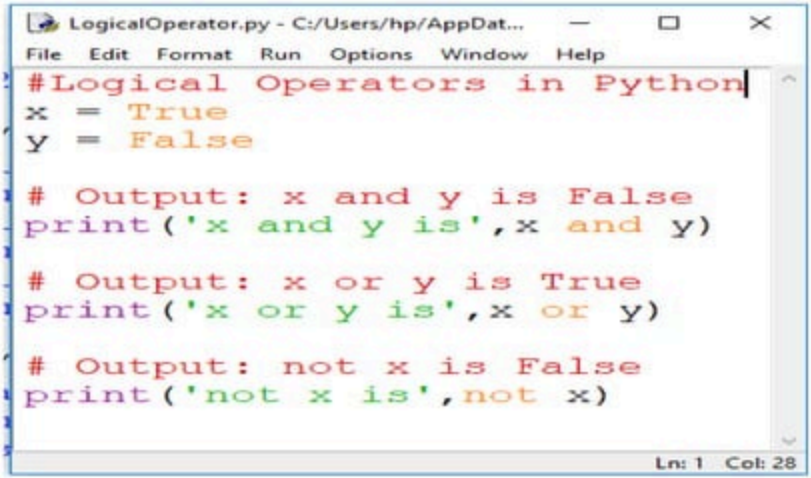
Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True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


```
x > y  is False
x < y  is True
x == y is False
x != y is True
x >= y is False
x <= y is True
```

Logical operators

Logical operators in Python

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




```
x and y is False
x or y is True
not x is False
```

Bitwise operators

- Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.
- For example, 2 is 10 in binary and 7 is 111.
- In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Bitwise operators in Python

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

Assignment operators

- Assignment operators are used in Python to assign values to variables.
- a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same.
- It is equivalent to a = a + 5.

Assignment operators in Python

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

Special operators

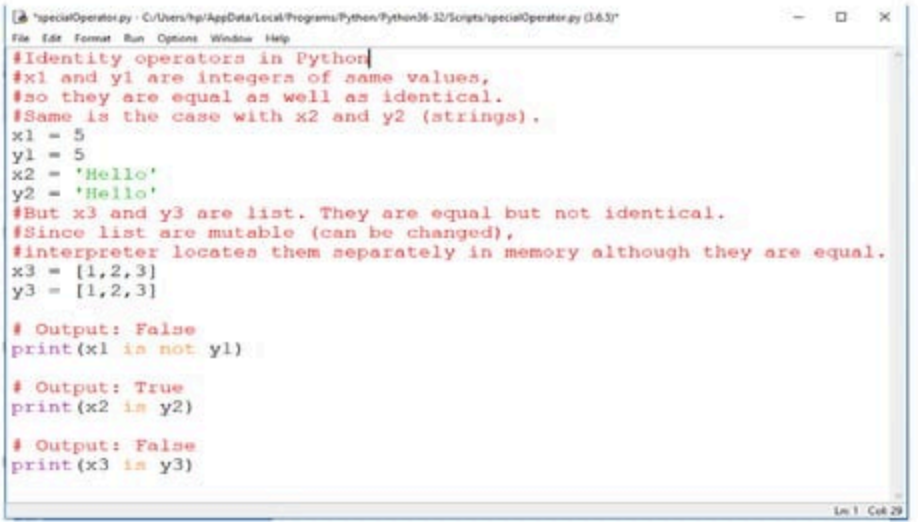
- identity operator
- membership operator

Identity operators

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

Identity operators in Python

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

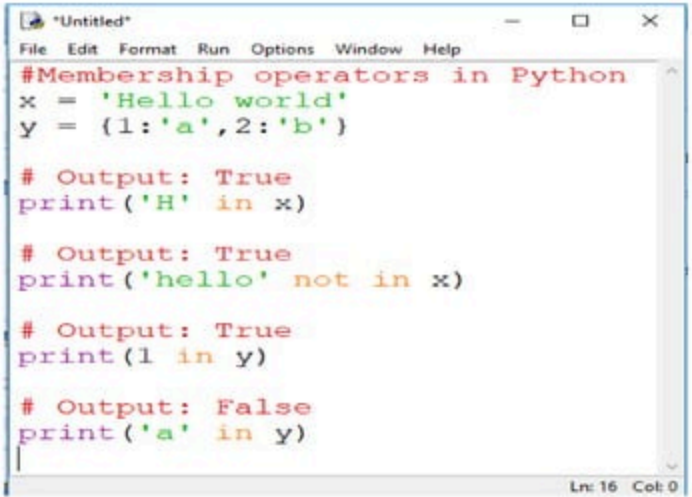



```
False
True
False
```

Membership operators

- in and not in are the membership operators in Python
- They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
- In a dictionary we can only test for presence of key, not the value

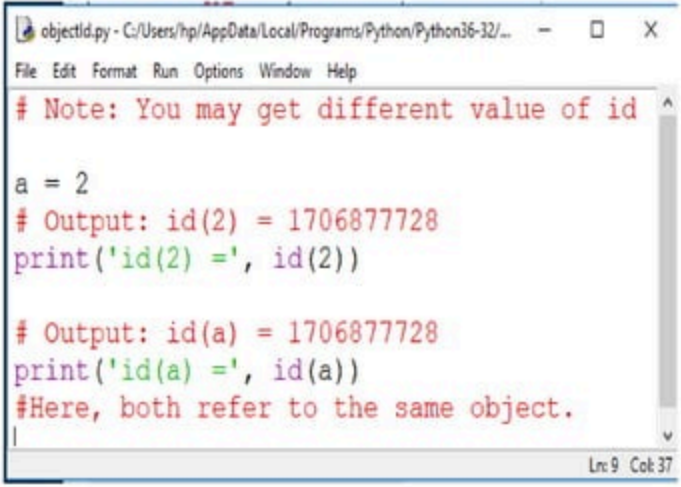
Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x



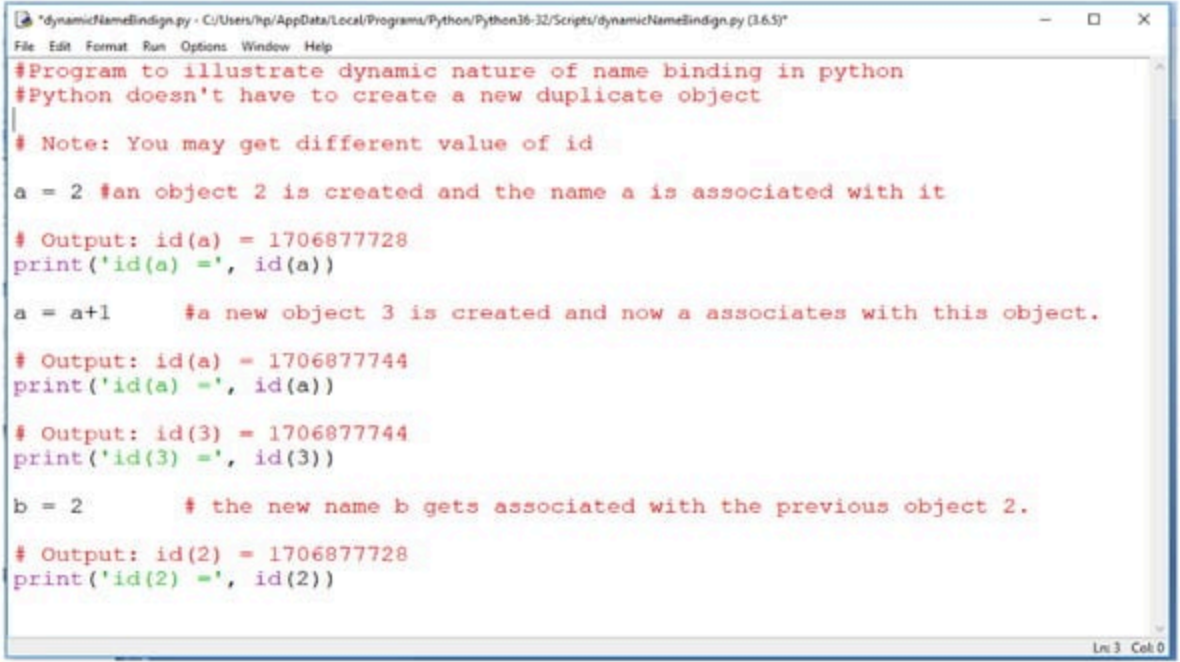

```
True
True
True
False
```

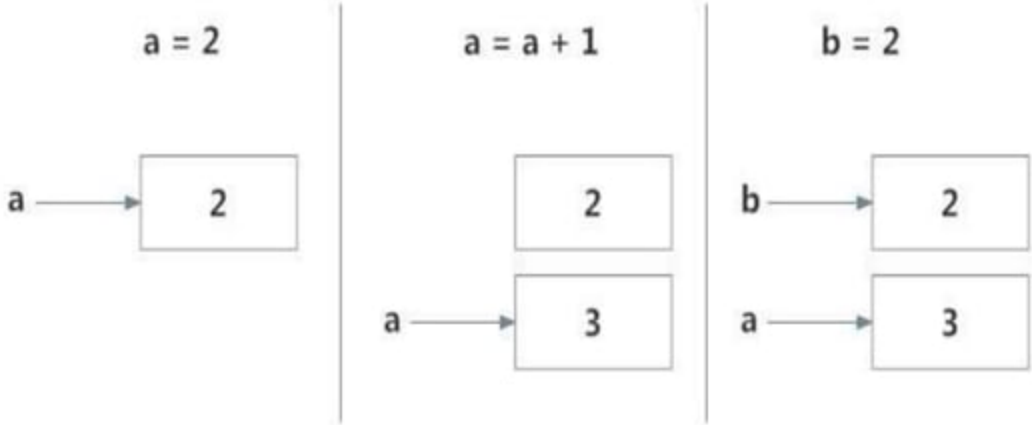
Python Namespace and Scope

- Name (also called identifier) is simply a name given to objects.
- Everything in Python is an object
- Name is a way to access the underlying object.
- For example:
  - when we do the assignment a = 2, here 2 is an object stored in memory and a is the name we associate it with
  - We can get the address (in RAM) of some object through the built-in function, id()




```
id(2) = 1706877728
id(a) = 1706877728
```


dynamicNameBindign.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/dynamicNameBindign.py (3.6.5)
File Edit Format Run Options Window Help
#Program to illustrate dynamic nature of name binding in python
#Python doesn't have to create a new duplicate object
# Note: You may get different value of id
a = 2 #an object 2 is created and the name a is associated with it
# Output: id(a) = 1706877728
print('id(a) =', id(a))
a = a+1 #a new object 3 is created and now a associates with this object.
# Output: id(a) = 1706877744
print('id(a) =', id(a))
# Output: id(3) = 1706877744
print('id(3) =', id(3))
b = 2 # the new name b gets associated with the previous object 2.
# Output: id(2) = 1706877728
print('id(2) =', id(2))
Ln: 3 Col: 0


a = 2
a
2
a = a + 1
2
a
3
b = 2
b
2
a
3

A name could refer to any type of object

```
>>> a = 5
>>> print(a)
5
>>> a ="Hello World!"
>>> print(a)
Hello World!
>>> a = [1,2,3]
>>> print(a)
[1, 2, 3]
>>>
```

Functions are objects too, so a name can refer to them as well

hello.py - C:/Users/Cab/AppData/Local/Programs/Python/Python36/hello.py (3.6.3)

File Edit Format Run Options Window Help

```
def hello_function():
    print("Hello")
hello_function()
a=hello_function
a()
```

```
Hello
Hello
```

What is a Namespace in Python?

- namespace is a collection of names.
- Help to eliminate naming collision/naming conflict

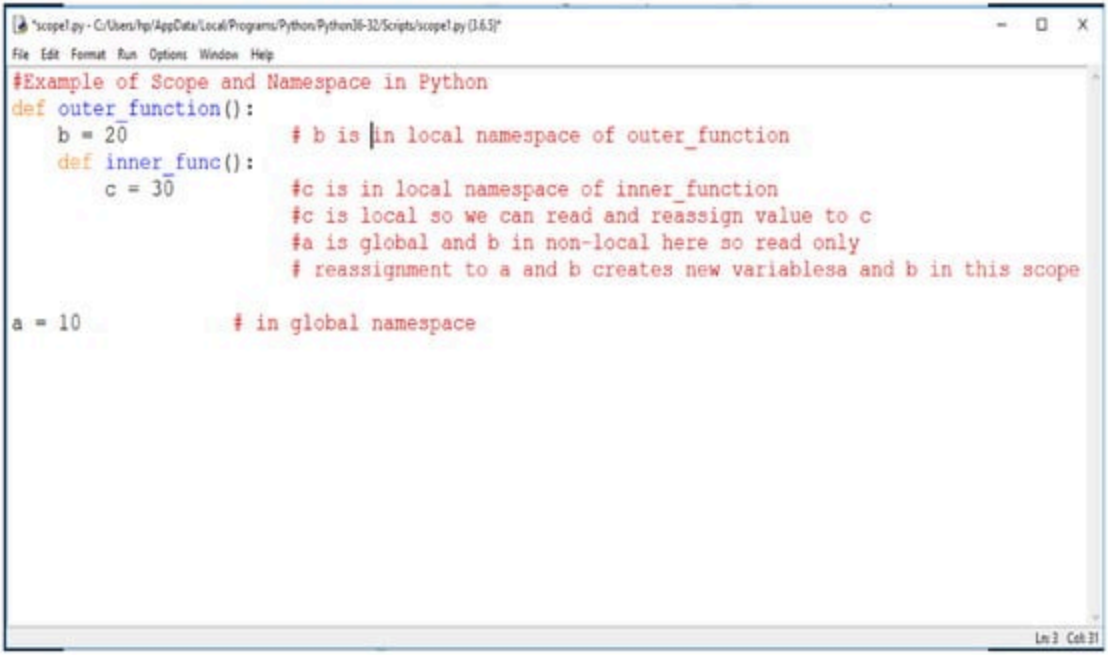
- A namespace containing all the built-in names is created when we start the Python interpreter and exists as long we don't exit
- Each module creates its own global namespace
- Modules can have various user defined functions and classes.
- A local namespace is created when a function is called, which has all the names defined in it.
- Similar, is the case with class

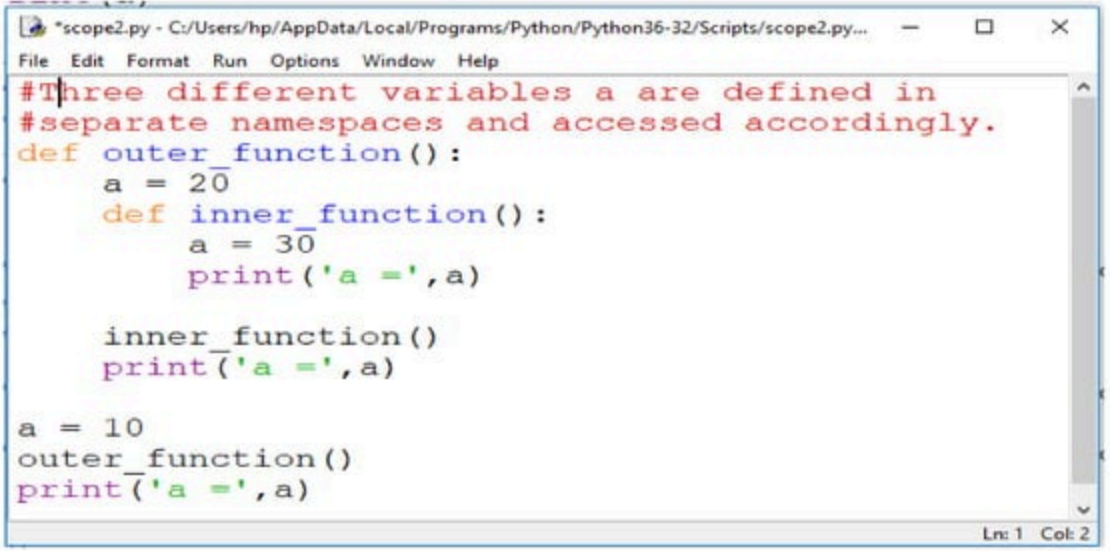

Built-in Namespace
Module: Global Namespace
Function: Local Namespace

Python Variable Scope

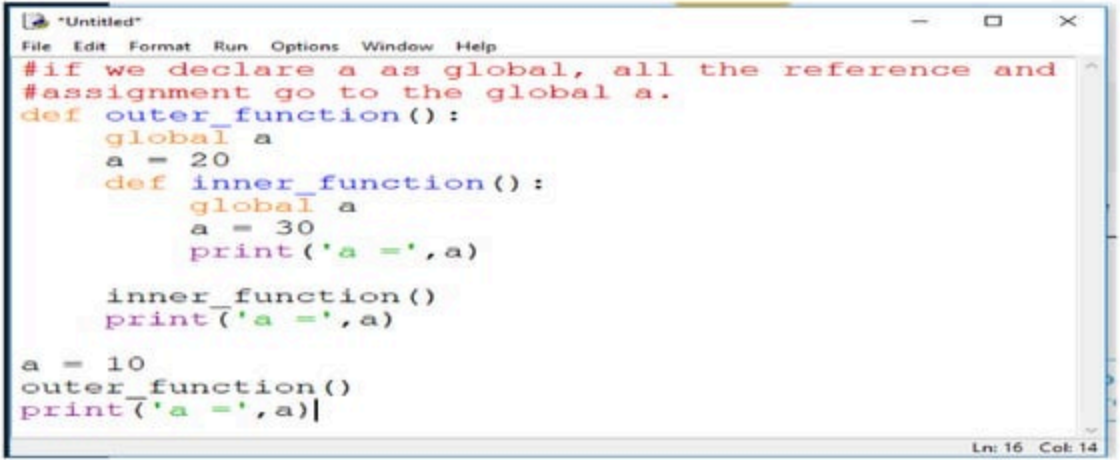
- All namespaces may not be accessible from every part of the program
- Scope is the portion of the program from where a namespace can be accessed directly without any prefix

- At any given moment, there are at least three nested scopes.
- Scope of the current function which has local names
- Scope of the module which has global names
- Outermost scope which has built-in names
- When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.
- If there is a function inside another function, a new scope is nested inside the local scope.


scope1.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/scope1.py (3.6.5)
File Edit Format Run Options Window Help
#Example of Scope and Namespace in Python
def outer_function():
b = 20 # b is in local namespace of outer_function
def inner_func():
c = 30 #c is in local namespace of inner_function
#c is local so we can read and reassign value to c
#a is global and b in non-local here so read only
# reassignment to a and b creates new variablesa and b in this scope
a = 10 # in global namespace
Ln: 3 Col: 21


*scope2.py - C:/Users/hp/AppData/Local/Programs/Python/Python36-32/Scripts/scope2.py...
File Edit Format Run Options Window Help
#Three different variables a are defined in
#separate namespaces and accessed accordingly.
def outer_function():
a = 20
def inner_function():
a = 30
print('a =',a)
inner_function()
print('a =',a)
a = 10
outer_function()
print('a =',a)
Ln: 1 Col: 2

```
a = 30
a = 20
a = 10
```


"Untitled"
File Edit Format Run Options Window Help
#if we declare a as global, all the reference and
#assignment go to the global a.
def outer_function():
global a
a = 20
def inner_function():
global a
a = 30
print('a =',a)
inner_function()
print('a =',a)
a = 10
outer_function()
print('a =',a)
Ln: 16 Col: 14

```
a = 30
a = 30
a = 30
```