Python Programming

Dive Deep

1

Index

Introduction

- Easy to learn – Beginner's programming language
- Cross-platform Interpreted Language
- Clean and Elegant Syntax
- Free and Open source
- Object-Oriented Programming Language
- Large Standard Library
- Integrated – Easily integrated with C, C++, JAVA, etc..
- Magic Methods

Monty Python are a British surreal comedy group who created the sketch comedy television show Monty Python's Flying Circus, which first aired on the BBC in 1969.

Application Areas

- **Web Development** – Django, Flask, Web2py, Bottle, Tornado

Application Areas

- **Graphical User Interfaces** – Kivy, PyQT, Tkinter, PyGUI, PySide, WxPython, pygame

Application Areas

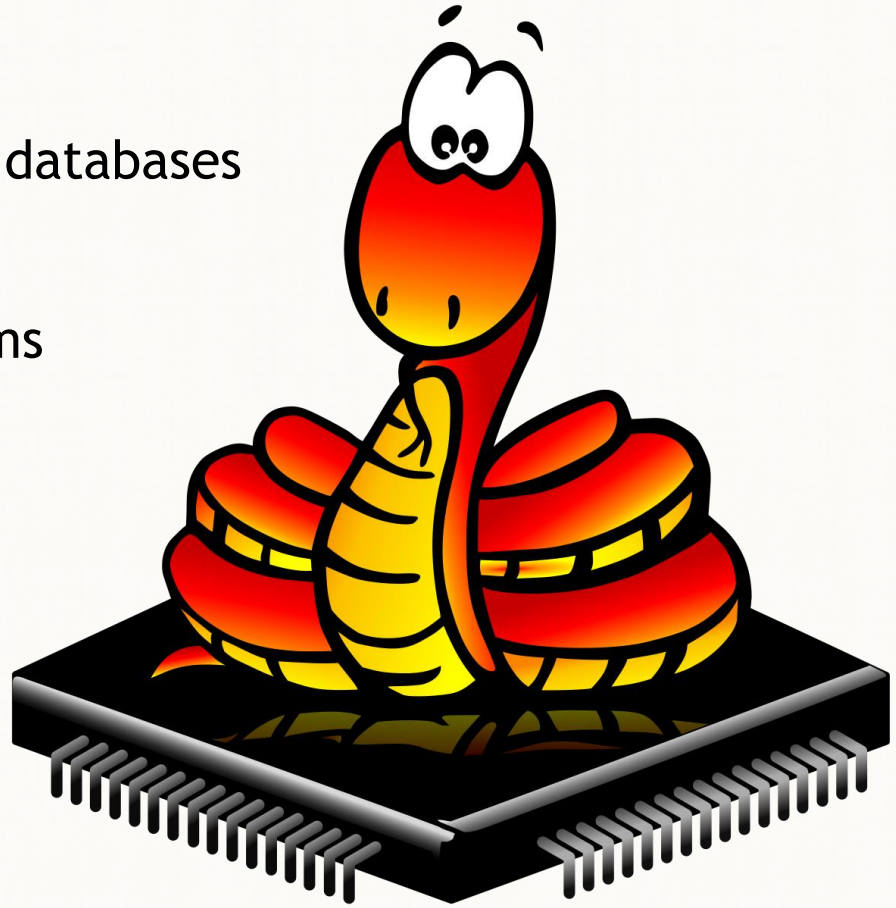
- **Multimedia Programming** – PyMedia, GStreamer, PyGame, Pyglet

Application Areas

- **Database Programming** – Supporting connectors for all databases
- **Networking** – Asyncio, Diesel, Pulsar, Twisted, NAPALM
- **Automation** – ANN, IOT, MicroPython, Embedded Systems

AsyncIO

MySQL®



Application Areas

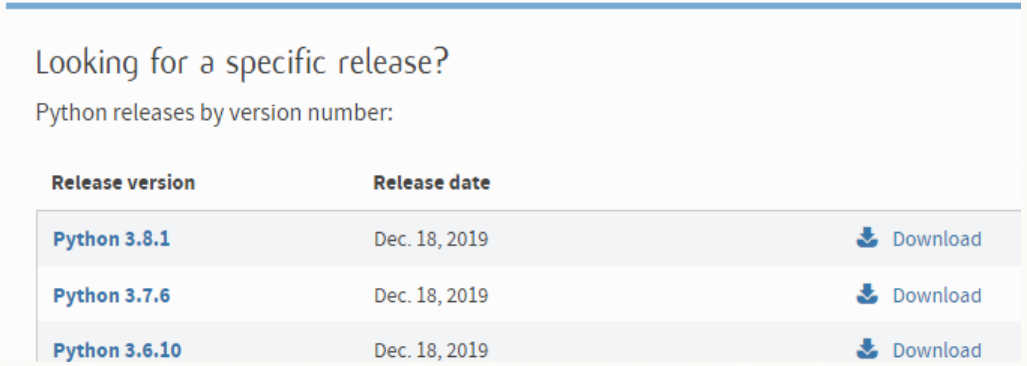
- **Web Scraping -** Beautiful Soup, Scrapy, Requests, Selenium
- **System Administration -** Fabric, Salt, Psutil, Ansible, Chef, Puppet, Blueprint, Buildout, Shinken
- **Scientific Computing -** Astrophy, Biopython, Bokeh, Cubes, Dask, Matplotlib, NetworkX, NumPy, Pandas, PyTorch, SciPy

Application Areas

- **Text Processing** – NLTK, Polyglot, TextBlob, CoreNLP, Pattern, Vocabulary, QuePy
- **Image Processing** – OpenCV, Pillow, Scikit-image
- **Machine Learning** – Scikit-learn, TensorFlow, Keras, Theano, PyTorch
- **Data Analytics** – NumPy, Pandas, Matplotlib, Seaborn, Plot.ly, SciPy
- **Data Science** – ML, DA, TensorFlow, Keras

How to get?

- Visit
  - https://www.python.org/downloads/

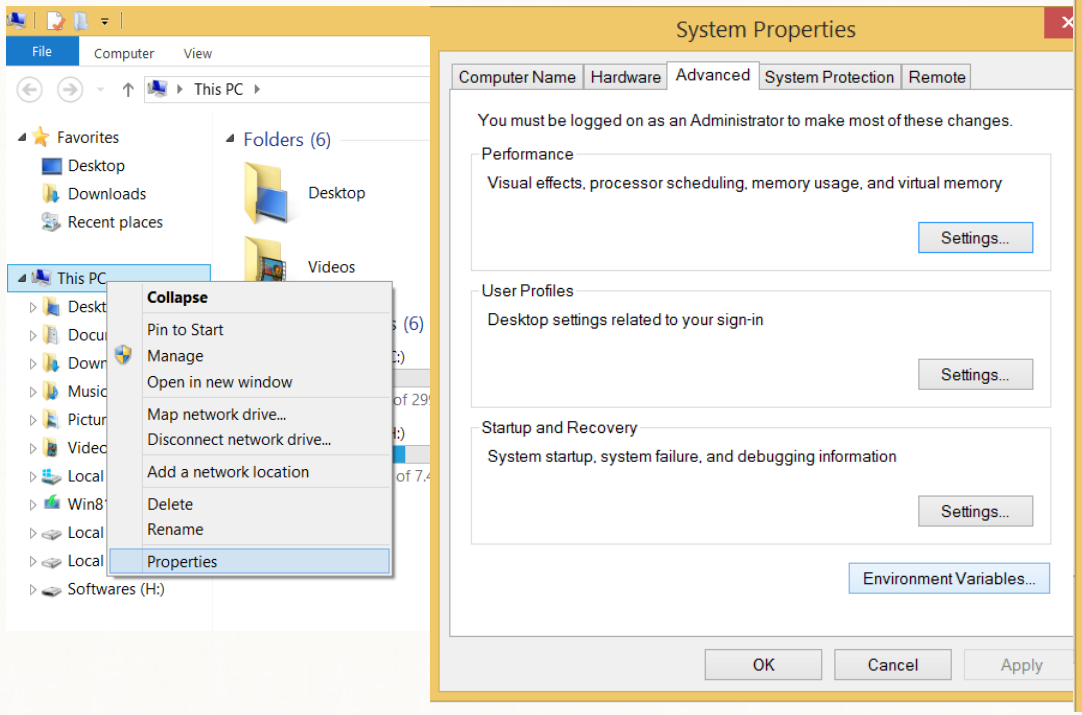


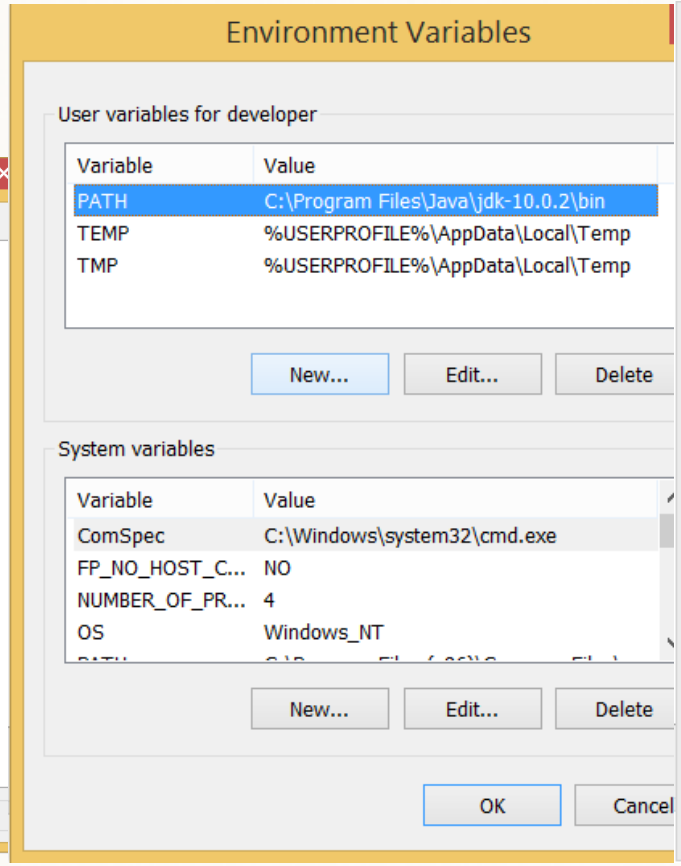
Install on Windows

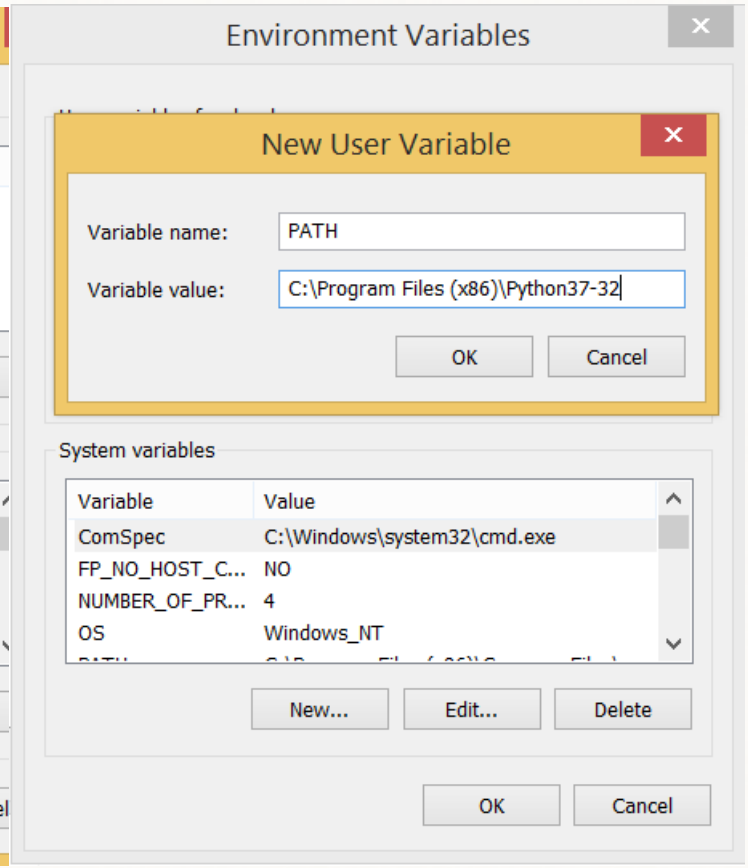
- Run Installer
- Make sure you tick on “ADD PYTHON to PATH”

Setting Up Environment

- In case you miss to add path,







Install on Linux (Ubuntu)

- sudo apt-get install python3
- sudo apt-get install idle3

Verify Installation

C:\WINDOWS\system32\cmd.exe - python

```
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Himanshu>python
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

root@javatpoint: /var/www/html

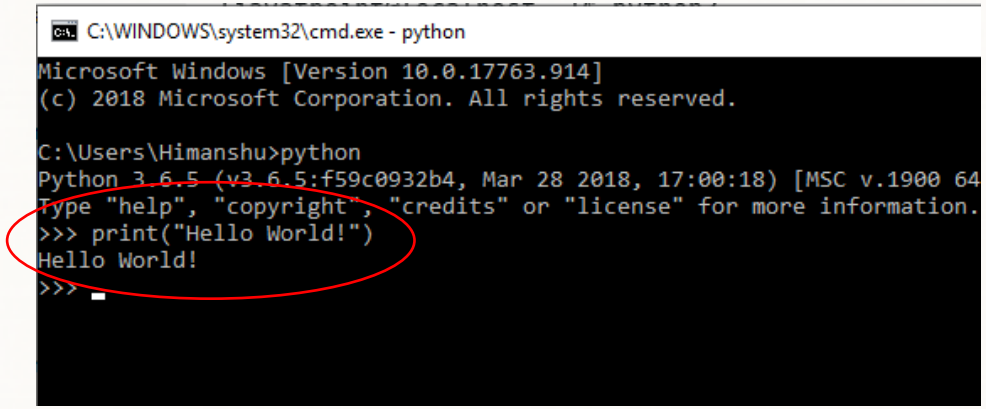
```
root@javatpoint:/var/www/html# python3
Python 3.5.2 (default, Sep 14 2017, 22:51:06)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

How to write Python Code?

- Python provides us the two ways to run a program:
  - Using Interactive interpreter prompt
  - Using a script file

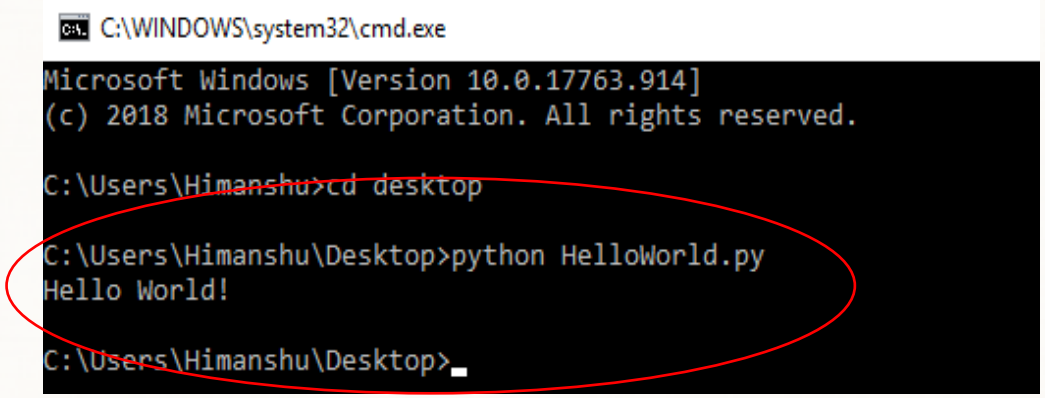
Using Interactive Interpreter Prompt

- Open Terminal/Command Prompt/Idle3
- In case of Terminal or Command Prompt
  - python or python3 (in case you have python2 and python3 both installed)
- Type
  - print("Hello World!")
  - and press ENTER key
- You will get a message in console showing
  - Hello World!



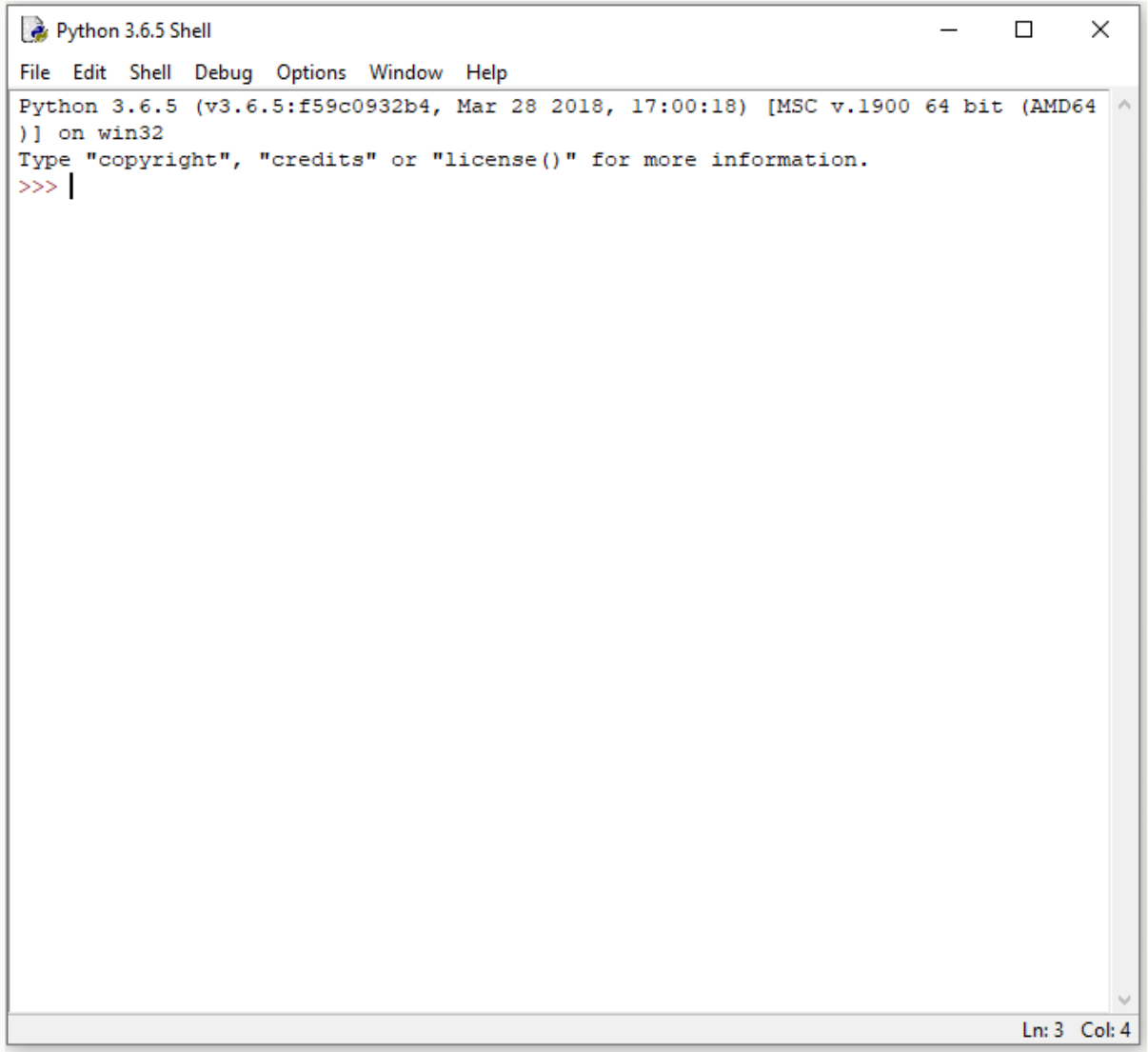
Using a script file

- Open your favorite code editor, type the following code
  - print("Hello World!")
- Save the file, lets say, HelloWorld.py (.py is python code extension, and file name can be anything with alpha-numerals)
- Now open a Terminal or Command Prompt in the very same folder of above file.
- And then type following command
  - python HelloWorld.py (or python3 HelloWorld.py)



But wait…

- Don’t get just bored with this tradition “Hello World” thing
- Lets find out, how python stand ahead of all
- Head on to your Python Terminal Window (or Idle Shell Window)



Now lets do something

- 4+5
- 8/3
- 4//2
- 19* “o”
- a = 34
- b = “good”
- print(b)
- print(a,b)

How Python Works?

- Python is an interpreted language i.e., Interpreted language requires **interpreter**, which takes the source code and **executes one instruction at a time**.
- **Now, what is an interpreter?**
- An interpreter is a program that is written in some other language and compiled into machine readable language. The interpreter itself is the machine language program and is written to read source programs from the interpreted language and interpret them.

Getting Output on Console

- The **print()** function
  - Syntax
    - print(*objects, sep=‘ ‘, end=‘\n’, file=sys.stdout, flush=False)
  - *objects – Objects to print (can be single or multiple).
  - sep – The separator string to separate two values in case of printing multiple objects.
  - end – String to be printed after all objects are printed.
  - file – The location where we want to print our output.
  - flush – Normally output to a file or the console is buffered, with text output at least until you print a newline. The flush makes sure that any output that is buffered goes to the destination.

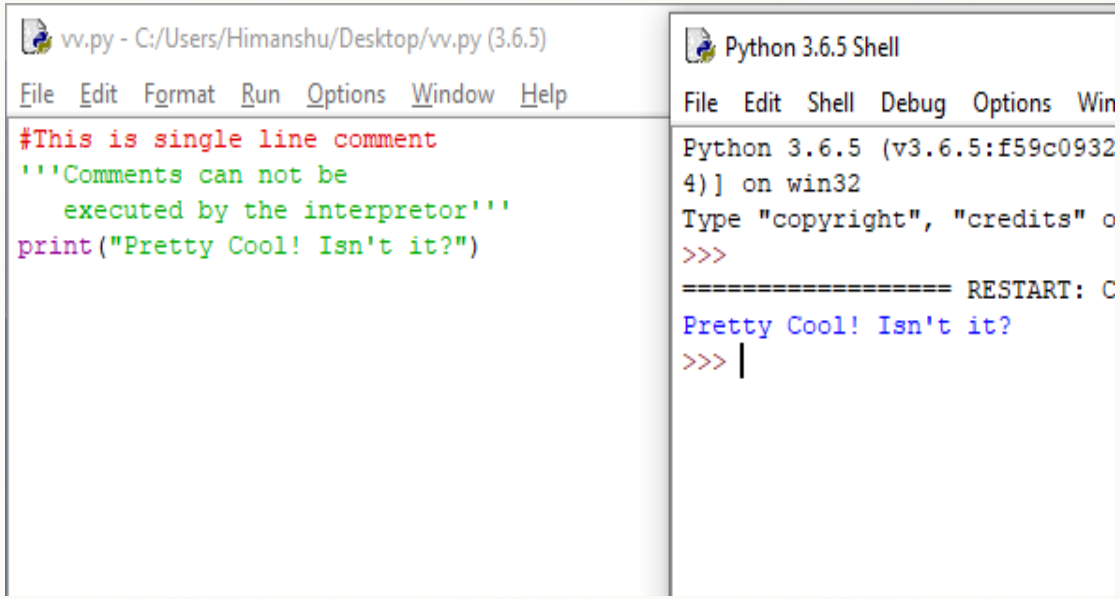
Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Comments starts with a #, and Python will ignore them

```
#This is a comment
print("Hello, World!")
print("Hello, World!") #This is a comment
```

- We can use multi line comments also using triple quoted string (although it means something else, we will discuss it later on)

```
"""This is a comment
Multiline comment"""
```



Python Variables

- Variable is a name used to refer memory locations because we can't remember memory addresses and used to hold values.
- In Python, we don't need to specify type of the variable because Python is smart enough to get variable type.
- Variables can be named using combination of letters, digits and underscore (_) . No other special symbol can be used.
- Variables can begin with letter or underscore but not with digit.
- Python is case sensitive language.
- Variables can also be called as identifiers.
- Valid identifiers:- a123, _n, n_9
- Invalid identifiers:- 1a, n%4, n 9
- The equal (=) operator is used to assign value to a variable.
- Python doesn't bound us to declare variables before use. It allows us to create variable at required time.
- Example:

```
a = 10
name = "Python Developer"
```

Python Variables contd.

- Python support Multiple Assignments

```
x=y=z=50   #(Here, x,y,z have same value i.e., 50)
x,y=20,42  #(Here, x =20 and y=42)
```

- Examples to demonstrate the variables:-

```
a=10
b,c=20,30
x=y=z=50
print(a)
print(b)
print(c)
print(x)
print(y)
print(z)
```

Python Keywords

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', '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']
>>>
```

Python Operators

- Arithmetic `+,-,*,/,//,**,%`
- Relational `>, <, ==, !=, >=, <=`
- Assignment `=, +=, -=, *=, /=, //=, **=, %=`
- Logical `and, or, not`
- Bitwise `&, |, ^, <<, >>`
- Identity `is, is not`
- Membership `in, not in`

Arithmetic Operators

Operator	Name	Example	Output
+	Addition	2+3	5
-	Subtraction	5-3	2
*	Multiplication	5*3	15
/	Division	5/2	2.5
//	Floor Division	5//2	2
%	Modulus	5%2	1
**	Exponentiation	5**2	25

Relational Operators

Operator	Example	Same As
=	X=5	X=5
+=	X+=3	X=X+3
-=	X-=3	X=X-3
*=	X*=3	X=X*3
/=	X/=3	X=X/3
%=	X%=3	X=X%3
//=	X//=3	X=X//3
**=	X**=3	X=X**3
&=	X&=3	X=X&3
|=	X|=3	X=X|3
^=	X^=3	X=X^3
>>=	X>>=3	X=X>>3
<<=	X<<=3	X=X<<3

Assignment Operators

Operator	Example	Same As
=	X=5	X=5
+=	X+=3	X=X+3
-=	X-=3	X=X-3
*=	X*=3	X=X*3
/=	X/=3	X=X/3
%=	X%=3	X=X%3
//=	X//=3	X=X//3
**=	X**=3	X=X**3
&=	X&=3	X=X&3
|=	X|=3	X=X|3
^=	X^=3	X=X^3
>>=	X>>=3	X=X>>3
<<=	X<<=3	X=X<<3

Logical Operators

Operator	Name	Example	Output
and	Logical And	2<5 and 2<10	True
or	Logical Or	2<5 or 2<1	True
not	Logical Not	not(2<5 and 2<10)	False

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of the two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Left Shift	Shift bits to left pushing zeros in right
>>	Right Shift	Shift bits to right pushing zeros in left

Identity Operators

Operator	Description	Example
is	Returns True if both variables are same object	x is y
is not	Returns True if both variables are not same object	x in not y

Membership Operators

Operator	Description	Example
in	Returns True if a sequence have the specified value	x in y
not in	Returns True if a sequence do not have the specified value	x not in y

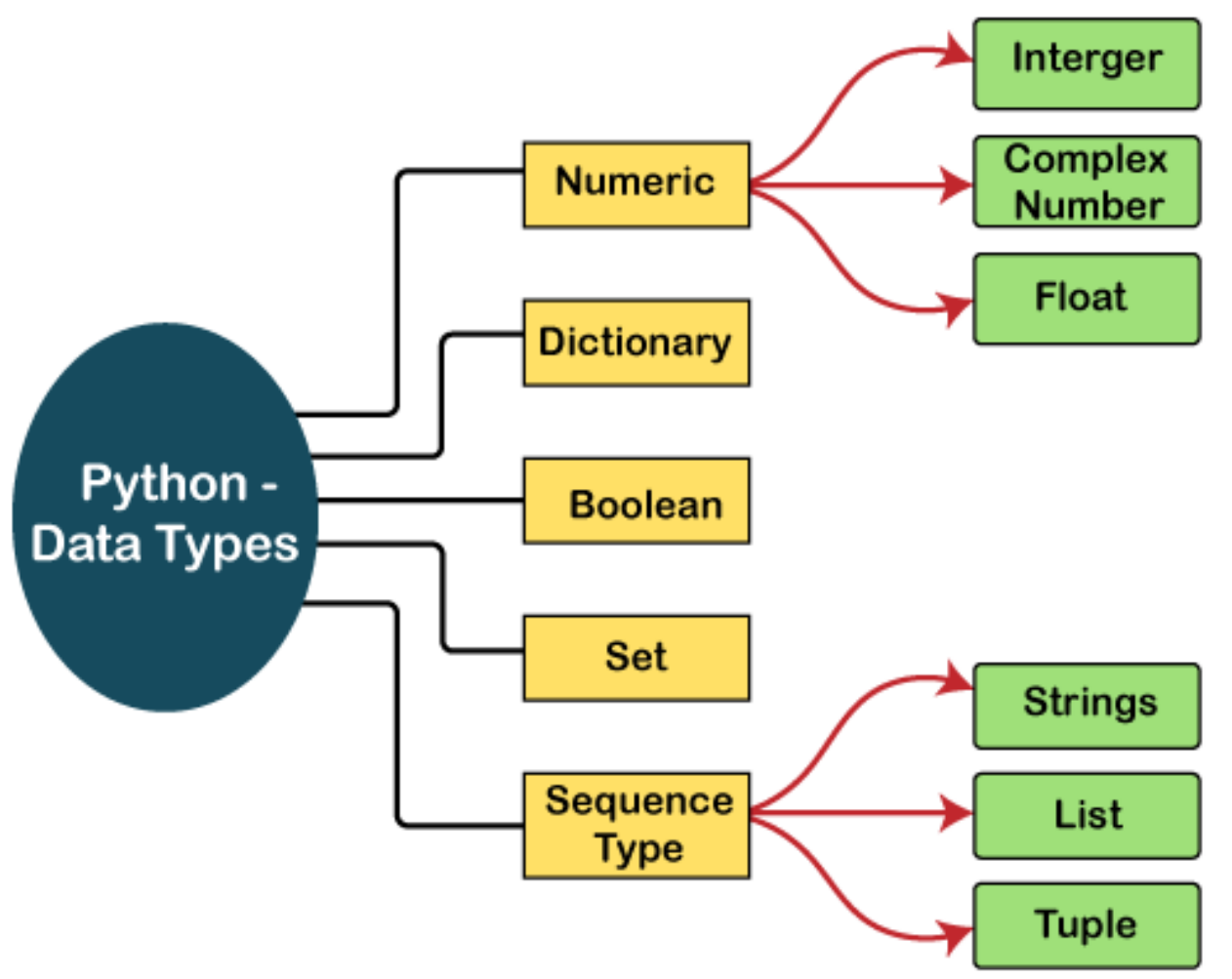
Use of Brackets in Python

- () Function Arguments, Expressions, Tuples
- {} Dictionary, Sets
- [] Lists, Indexing Slicing
- <> not used as brackets

35

Data Classes & Built-In Functions

Data Types


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple

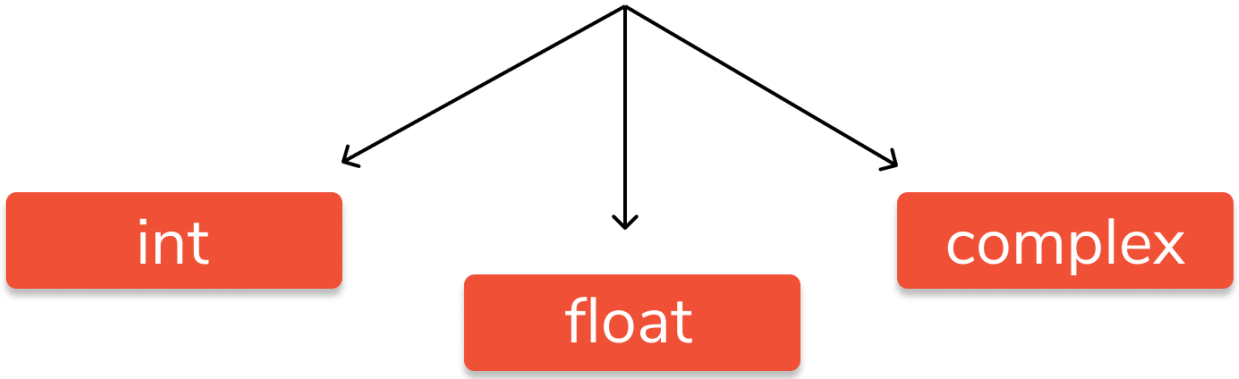
Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

38

Numeric Category

Data Classes Built-in Functions

Python Numbers



Int | Float | Complex

```
>>> a = 10
>>> print(a)
10
>>> type(a)
<class 'int'>
>>> print(type(a))
<class 'int'>
>>> a
10
>>> b = 10.3
>>> print(b)
10.3
>>> print(type(b))
<class 'float'>
>>> c = 3+4j
>>> print(c)
(3+4j)
>>> print(type(c))
<class 'complex'>
>>>
```

41

String

Data Classes Built-in Functions

string

- Strings are arrays of bytes representing Unicode characters.
- A String is a **sequence of characters**.
- A single character is simply a string with a length of 1.
- A string is always defined in between single or double quotes (‘ or “).
- Strings are **immutable**.
- To create multiline strings use triple quotes (’’’ or ”””)
- Strings can be accessed via three methods:
  - Indexing
  - Slicing
  - Loops

Python Strings contd.

- Example to create String variable :-

```
a=“This is Single line string”
b=“”””This is Multliline
                string””””
print(type(a),type(b))
print(a, ‘\n’, b)
```

```
a='This is single line string'
b="""This is multiline
string"""
print(type(a),type(b))
print(a,b,sep='\n')
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar
)] on win32
Type "copyright", "credits" or "lice
>>>
================== RESTART: C:\Users
<class 'str'> <class 'str'>
This is single line string
This is multiline
string
>>>
```

Accessing String using Indexing

- Index of strings starts with 0 and ends with (n-1). Here, n stands for the total number of characters in the string i.e., length of string.

"Hello World!"

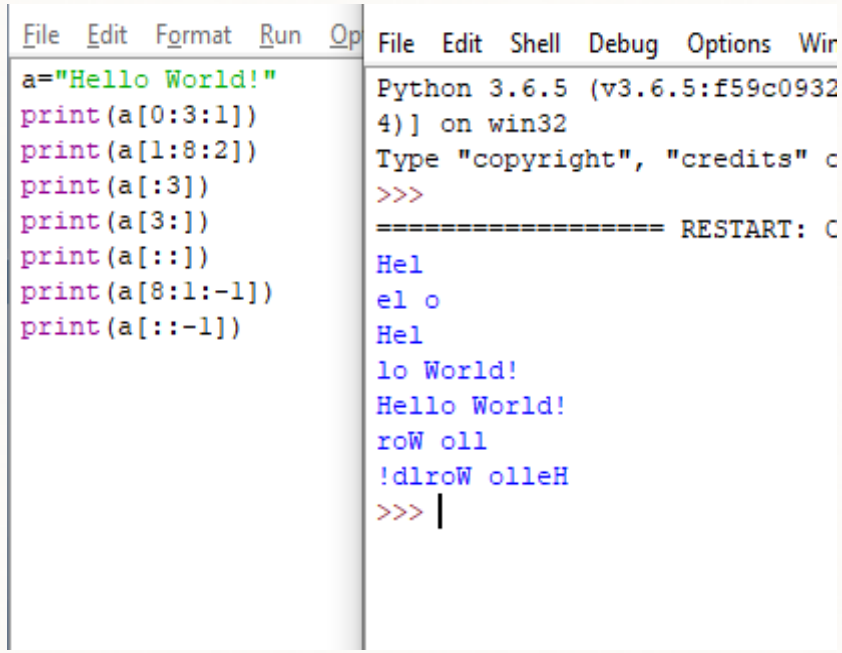
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o		W	o	r	l	d	!

- To access string character by character use index number enclosed by square braces.
- Example:
  - `a="Hello World!"`
  - `print(a[0])`
  - `print(a[6])`

Accessing String using Slicing

- Syntax:
  - str[start:stop:step]
    - start – start index number (included)
    - stop – stop index number (excluded)
    - step – spacing between values and direction
- Example:

```
a="Hello World!"
print(a[0:3:1])
print(a[1:8:2])
print(a[:3])
print(a[3:])
print(a[::])
print(a[8:1:-1])
print(a[::-1])
```



Python String Operations

- Repeating the String

```
>>>
>>> a = 'good'
>>> a * 3
'goodgoodgood'
>>>
```

- Concatenation

```
>>>
>>> a = 'hello'
>>> b = 'bye'
>>> a + b
'hellobye'
>>>
```

String Escape Sequences

- \<newline> Ex:- print(“Hello \
  World”)
- \\ Backslash Ex:- print(“2020\\01\\01”)
- \’ Single quote Ex:- print(“Monty\’s Python”)
- \” Double quote Ex:- print(“The \” for\” loop”)
- \a ASCII Bell Ex:- print(“\a”)
- \b ASCII Backspace Ex:- print(“Hello \b World”)
- \n ASCII Linefeed Ex:- print(“Hello\nWorld”)
- \r Carriage Return Ex:- print(“Hello\rWorld“)
- \t Horizontal TAB Ex:- print(“Hello\tWorld”)
- \v Vertical TAB Ex:- print(“Hello\vWorld”)

```
print("Hello\
World")
print("2020\\01\\01")
print("Monty\'s Python")
print("The \"for\" loop")
print("\a")
print("Hello \b World")
print("Hello\nWorld")
print("Hello\rWorld")
print("Hello\tWorld")
print("Hello\vWorld")
```

```
=================
HelloWorld
2020\01\01
Monty's Python
The "for" loop
▯
Hello ▯ World
Hello
World
HelloWorld
Hello    World
Hello▯World
>>>
```

Python String Functions

- capitalize()
  - Return a copy of the string with its first character capitalized and the rest lowercased.
- center(width[,fillchar])
  - Return centered in a string of length width. Padding is done using the specified fillchar.
- count(sub[,sub[,end]])
  - Return number of occurrences of substring sub in range[start,end].
- endswith(sub[,start[,end]])
  - Return True if string ends with specified substring sub in range[start,end].
- find(sub,[,start[,end]])
  - Return lowest index in the string where substring sub is found in range[start,end] else return -1.
- rfind(sub,[,start[,end]])
  - Return highest index in the string where substring sub is found in range[start,end] else return -1.
- index(sub,[,start[,end]])
  - Return lowest index in the string where substring sub is found in range[start,end] else raise ValueError.
- rindex(sub,[,start[,end]])
  - Return highest index in the string where substring sub is found in range[start,end] else raise ValueError.

Python String Functions

- lower()
  - Convert all upper case letters to lower case in the string.
- split(sep[,maxsplit])
  - Return a list of substrings of the string splitted on the basis of sep and length of maxsplit.
- join(words)
  - Concatenate a list or tuple of words separating with sep.
- lstrip()
  - Return a string with leading white spaces removed.
- rstrip()
  - Return a string with trailing white spaces removed.
- strip()
  - Return a string with leading and trailing both white spaces removed.
- swapcase()
  - Return copy of string after converting lowercase letters to uppercase letters and vice versa.
- upper()
  - Return copy of string after converting all characters to uppercase letters.
- replace(old, new)
  - Return a copy of string with all occurrences of substring old replaced by new.
- len(string)
  - Return length of the string i.e., number of characters in the string.

```python
print("hello world".capitalize())
print("hello".center(50))
print("hello".center(50,'%'))
print("This is the example of counting".count('h'))
print("The Monty Python's Flying Circus".endswith('cus'))
print("The Monty Python's Flying Circus".find('y'))
print("The Monty Python's Flying Circus".rfind('y'))
print("The Monty Python's Flying Circus".lower())
print("The Monty Python's Flying Circus".upper())
print("The Monty Python's Flying Circus".swapcase())
print("Hello World".replace('World','Python'))
print("     Hello World     ".lstrip())
print("     Hello World     ".rstrip())
print("     Hello World     ".strip())
print("The Monty Python's Flying Circus".split("'"))
print('-'.join(['2020','01','01']))
```

```
Hello world
                        hello
%%%%%%%%%%%%%%%%%%%%%%%hello%%%%%%%%%%%%%%%%%%%%%%%
2
True
8
21
the monty python's flying circus
THE MONTY PYTHON'S FLYING CIRCUS
tHE mONTY pYTHON'S fLYING cIRCUS
Hello Python
Hello World
     Hello World
Hello World
['The Monty Python', 's Flying Circus']
2020-01-01
>>>
```

51

List

Data Classes Built-in Functions

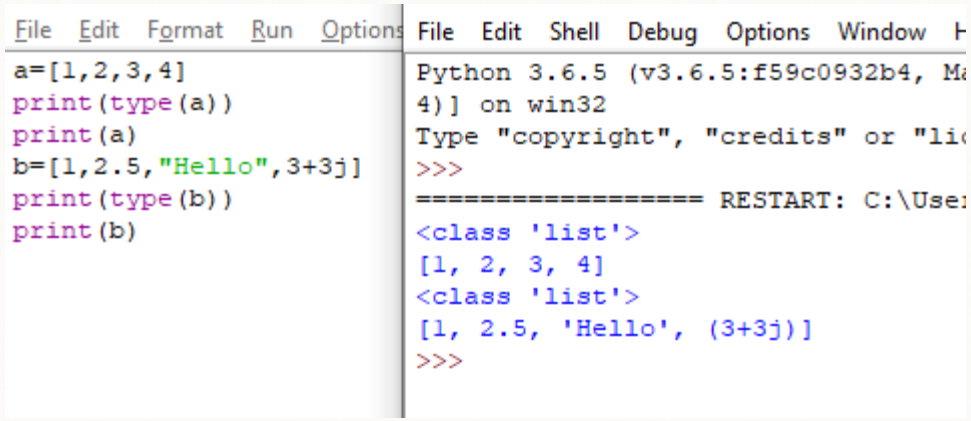
list

- Lists are just like the arrays.
- Lists may contain Homogenous or Heterogeneous data i.e., A list may contain integer data as well as float, string, list or any other type of data unlike arrays.
- Lists are **mutable** objects.
- A List is always defined in between squared braces [ ]
- Lists can be accessed via three methods:
  - Indexing
  - Slicing
  - Loops

Python Lists contd.

- Example to create a List object

```
a=[1,2,3,4]
print(type(a)
print(a)
b=[1,2.5,”Hello”,3+3j]
print(type(b))
print(b)
```



Accessing Lists using Indexing

- Index of lists starts with 0 and ends with (n-1). Here, n stands for the total number of elements in the list i.e., length of list.

[1,2.5,'Hello',3+3j]

0	1	2	3
1	2.5	'Hello'	3+3j

- To access list element by element use index number enclosed by square braces.
- Example
  - `a= [1,2.5,'Hello',3+3j]`
  - `print(a[0])`
  - `print(a[-2])`
  - `print(a[2][3])`

Accessing List using Slicing

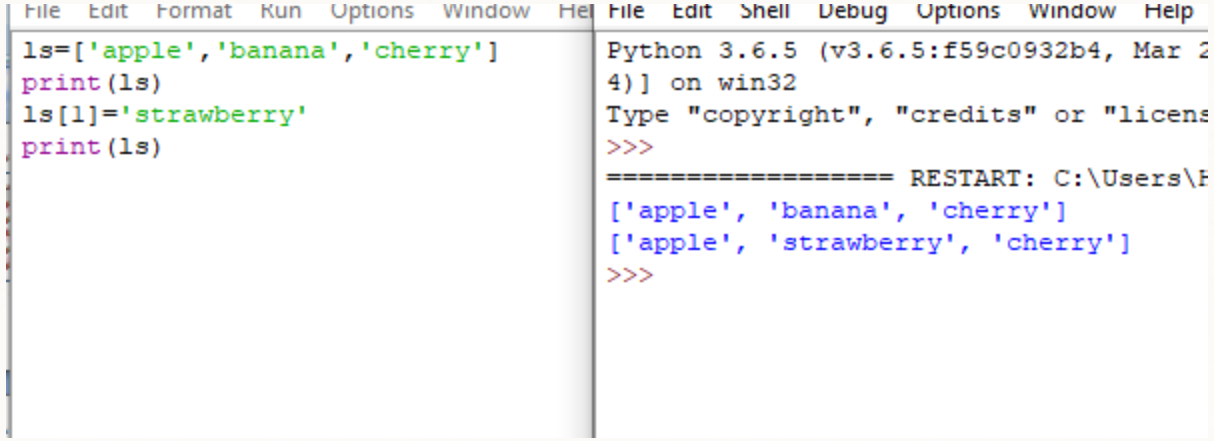
- Syntax :-
  - list[start:stop:step]
    - start – start index number (included)
    - stop – stop index number (excluded)
    - step – spacing between values and direction
- All rules of slicing we have seen in strings apply on lists also.
- Example

```
a= [1j, 2, 4, 4, “Python is easy!”,3, 3.5, 3, 6, 5]
print(a[0:3:1])
print(a[1:8:2])
print(a[:3])
print(a[-1:-4:-1])
```

Python List Operations

- Change Value of Item in a List
  - To change the value of specific item in a list, use index number and assign a value to it.
  - Example:

```
ls=[‘apple’, ‘banana’, ‘cherry’]
print(ls)
ls[1]=‘strawberry’
print(ls)
```



Python Lists Operations contd.

- Deleting a Item in List:

```
>>> a = ['apple', 'banana', 'cherry']
>>> del a[2]
>>> a
['apple', 'banana']
>>>
```

- Check if a item is in List:

```
>>> a = ['apple', 'banana', 'cherry']
>>> 'apple' in a
True
>>> 'mango' not in a
True
>>> 'papaya' in a
False
>>>
```

Python Lists Operations contd.

- Repeating Items of List

```
>>> a = ['apple', 'banana', 'cherry']
>>> a*2
['apple', 'banana', 'cherry', 'apple', 'banana', 'cherry']
>>>
```

- Concatenating Lists (Also used for insertion)

```
>>> a = ['apple']
>>> a += ['banana']
>>> a
['apple', 'banana']
>>> a += 'cherry'
>>> a
['apple', 'banana', 'c', 'h', 'e', 'r', 'r', 'y']
>>>
```

Python List Functions

- len(list)
  - Return length i.e., number of elements in a list.
- append(x)
  - Adds an item (x) to the end of the list.
- extend(iterable)
  - Extend the list by appending all the items from the iterable.
- insert(i,x)
  - Inserts an item (x) at a given index i.
- remove(x)
  - Removes the first item from the list that has a value of x. Returns an error if there is no such item.
- pop([i])
  - Remove and return the item at the given index in the list. If no index is specified, pop() removes and returns the last item in list.
- clear()
  - Removes all items from the list.
- index(x[,start[,end]])
  - Returns the position of the first list item that has a value of x. Raise a ValueError if there is no such item. The start and end arguments are used to limit the search for the item in a particular subsequence. Returned index is computed relative to the beginning of the full sequence.

Python List Functions

- count(x)
  - Returns the number of times x appears in the list.
- sort(key=None,reverse=False)
  - Sorts the items of the list in place. key specifies a function of one argument that is used to extract a comparison key from each list element. The default value is None (compares the elements directly). reverse Boolean value. If set to True, then the list elements are stored as if each comparison were reversed.
- reverse()
  - Reverses the elements of the list in place.
- copy()
  - Returns a shallow copy of the list.
- **PYTHON Built-in**
  - list([iterable])
    - list() constructor returns a mutable list of elements.
  - max(iterable,*,key,default)
    - Returns the largest item in inerrable or largest of two arguments.
  - min(iterable,*,key,default)
    - Returns the smallest item in inerrable or smallest of two arguments.

```python
a = ["bee", "moth"]
print(len(a))
print(a)
a.append("ant")
print(a)
a.extend(["ant", "fly"])
print(a)
a.insert(1, "wasp")
print(a)
a.remove("moth")
print(a)
a.pop()
print(a)
a.pop(1)
print(a)
a.clear()
print(a)
a = ["bee", "ant", "moth", "ant"]
print(a.index("ant"))
print(a.index("ant", 2))
print(a.count("ant"))
print(a.count("wasp"))
a.sort()
print(a)
a.sort(reverse=True)
print(a)
a.sort(key=len)
print(a)
a.reverse()
print(a)
b=a
b.append("ant")
print(a)
print(b)
b = a.copy()
b.append("ant")
print(a)
print(b)
print(max(b))
print(min(b))
```

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
4)] on win32
Type "copyright", "credits" or "license()" fo
>>>
==================
2
['bee', 'moth']
['bee', 'moth', 'ant']
['bee', 'moth', 'ant', 'ant', 'fly']
['bee', 'wasp', 'moth', 'ant', 'ant', 'fly']
['bee', 'wasp', 'ant', 'ant', 'fly']
['bee', 'wasp', 'ant', 'ant']
['bee', 'ant', 'ant']
[]
1
3
2
0
['ant', 'ant', 'bee', 'moth']
['moth', 'bee', 'ant', 'ant']
['bee', 'ant', 'ant', 'moth']
['moth', 'ant', 'ant', 'bee']
['moth', 'ant', 'ant', 'bee', 'ant']
['moth', 'ant', 'ant', 'bee', 'ant']
['moth', 'ant', 'ant', 'bee', 'ant']
['moth', 'ant', 'ant', 'bee', 'ant', 'ant']
moth
ant
>>>
```