

Ethan's Python Training



54- Hours Classroom Training

52- Python Hand-on Exercises

45 - Assignments

4- Mini Projects

1- Major Student Project (End to End)

100% - Practical Training

Certification Training

Prerequisites:

No eligibility, course start right from installation

Lab:

45 hours' lab sessions, 5 projects
Project Based Training

After the classes:

Students will easily crack Python interview and have advance knowledge of Python Automation and Data Science with Python

Syllabus

Module-1: Python Introduction

- What is Python and history of Python?
- Why Python and where to use it?
- Discussion about Python 2 and Python 3
- Set up Python environment for development
- Demonstration on Python Installation
- Discuss about IDE's like IDLE, Pycharm and Enthought Canopy
- Discussion about unique feature of Python
- Write first Python Program
- Start programming on interactive shell.
- Using Variables, Keywords
- Interactive and Programming techniques
- Comments and document interlude in Python
- Practical use cases using data analysis

Module 2 – Core Objects and Built-in Functions

- Python Core Objects and builtin functions
- Number Object and operations
- String Object and Operations
- List Object and Operations
- Tuple Object and operations
- Dictionary Object and operations
- Set object and operations
- Boolean Object and None Object
- Different data Structures, data processing

Module 3 – Conditional Statements and Loops

- What are conditional statements?
- How to use the indentations for defining if, else, elif block
- What are loops?
- How to control the loops
- How to iterate through the various object
- Sequence and iterable objects

Module 4 – UDF Functions and Object Functions

- What are various type of functions
- Create UDF functions
- Parameterize UDF function, through named and unnamed parameters
- Defining and calling Function
- The anonymous Functions - Lambda Functions
- String Object functions
- List and Tuple Object functions
- Dictionary Object functions

Module 5 – File Handling with Python

- Process text files using Python
- Read/write and Append file object
- File object functions
- File pointer and seek the pointer
- Truncate the file content and append data
- File test operations using os.path

Module-6 – Python Modules and Packages

- Python inbuilt Modules
- os, sys, datetime, time, random, zip modules
- Create Python UDM – User Defined Modules
- Define PYTHONPATH
- Create Python Packages
- init File for package initialization

Module 7 – Exceptional Handling and Object Oriented Python

- Python Exceptions Handling
- What is Exception?
- Handling various exceptions using try....except...else
- Try-finally clause
- Argument of an Exception and create self exception class
- Python Standard Exceptions
- Raising an exceptions, User-Defined Exceptions
- Object oriented features
- Understand real world examples on OOP
- Implement Object oriented with Python
- Creating Classes and Objects, Destroying Objects
- Accessing attributes, Built-In Class Attributes
- Inheritance and Polymorphism
- Overriding Methods, Data Hiding
- Overloading Operators

Syllabus

Module-8– Debugging, Framework & Regular expression

- Debug Python programs using pdb debugger
- Pycharm Debugger
- Assert statement for debugging
- Testing with Python using UnitTest Framework
- What are regular expressions?
- The match and search Function
- Compile and matching
- Matching vs searching
- Search and Replace feature using RE
- Extended Regular Expressions
- Wildcard characters and work with them

Module-9– Database interaction with Python

- Creating a Database with SQLite 3,
- CRUD Operations,
- Creating a Database Object.
- Python MySQL Database Access
- DML and DDL Operations with Databases
- Performing Transactions
- Handling Database Errors
- Disconnecting Database

Module 10 –Package Installation, Windows spreadsheet parsing and webpage scrapping

- Install package using Pycharm
- What is pip, easy_install
- Set up the environment to install packages?
- Install packages for XLS interface and XLS parsing with Python
- Create XLS reports with Python
- Introduction to web scraping and beautiful soup

Module 11 – Machine Learning with Python

- Understanding Machine Learning?
- Areas of Implementation of Machine Learning,
- Why companies prefer ML
- Major Classes of Learning Algorithms
- Supervised vs Unsupervised Learning, Learning
- Why Numpy?
- Learn Numpy and Scipy,
- Basic plotting using Matplotlib
- Algorithms using Skikit learn

Module 12 – Data Analysis with Pandas

- What is Pandas
- Creating Series
- Creating Data Frames,
- Grouping, Sorting
- Plotting Data/Create Graphs and Analysis
- Data analysis with data set

Module 13 – Hadoop dataprocessing with Python

- Introduction of Big Data
- Why Big Data
- Hadoop Eco system
- Understanding problem statements
- Market data Analysis with Python
- HDFS file system
- Cloudera Cluster of single node
- Map Reduce using Python
- Introduction to MrJob Package

Module 14 - Parallelism

- GIL
- Multithreading in Python
- Create Threads with parameters
- Create Daemon and Non Daemon processes
- Multiprocessing in Python

Getting Started with Python



Python Introduction

- Why Python and where to use it?
- What is Python and history of Python?
- Discussion about Python 2 and Python 3
- Set up Python environment for development
- Discuss about IDE's like IDLE, Pycharm and Enthought Canopy
- Demonstration on Python Installation
- Discussion about unique feature of Python

Python Programming

- Write first Python Program
- Start programming on interactive shell.
- Using Variables, Keywords
- Interactive and Programming technique
- Comments and document interlude in Python

What is Scripting Language?

A scripting language is a “wrapper” language that integrates OS functions.

The interpreter is a layer of software logic between your code and the computer hardware on your machine.

Wiki Says:

"Scripts" are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or bytecode.

The “program” has an executable form that the computer can use directly to execute the instructions.

The same program in its human-readable source code form, from which executable programs are derived (*e.g., compiled*)

Python is scripting language, fast and dynamic.



History of Python

- ✓ Python was developed by **Guido van Rossum** in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science (*Centrum Wiskunde & Informatica aka CWI*) in the Netherlands.
- ✓ Python is inherited from ABC programming and has many features including Interactive programming, object oriented, exceptional handling and many more.
- ✓ Python was named on 'Monty Python's Flying Circus' a comedy series created by the comedy group Monty Python.
- ✓ Python Version 2.0 was released in 2000, with many major new features including a full garbage collector and support for unicode.
- ✓ A Java-based version of Python exists in Jython and used to work with Java code. Similarly Iron Python, a C# version exists for the .Net.
- ✓ Python is called 'scripting language' because of it's scalable interpreter, but actually it is much more than that.



Why Python?

Easy to read

- ✓ Python scripts have clear syntax, simple structure and very few protocols to remember before programming.

Easy to Maintain

- ✓ Python code is easily to write and debug. Python's success is that its source code is fairly easy-to-maintain.

Portable

- ✓ Python can run on a wide variety of Operating systems and platforms and providing the similar interface on all platforms.

Broad Standard Libraries

- ✓ Python comes with many prebuilt libraries apx. 21K

High Level programming

- ✓ Python is intended to make complex programming simpler. Python deals with memory addresses, garbage collection etc internally.

Interactive

- ✓ Python provide an interactive shell to test the things before implementation. It provide the user the direct interface with Python.

Database Interfaces

- ✓ Python provides interfaces to all major commercial databases. These interfaces are pretty easy to use.

GUI programming

- ✓ Python supports GUI applications and has framework for Web. Interface to tkinter, WXPYthon, Django in Python make it

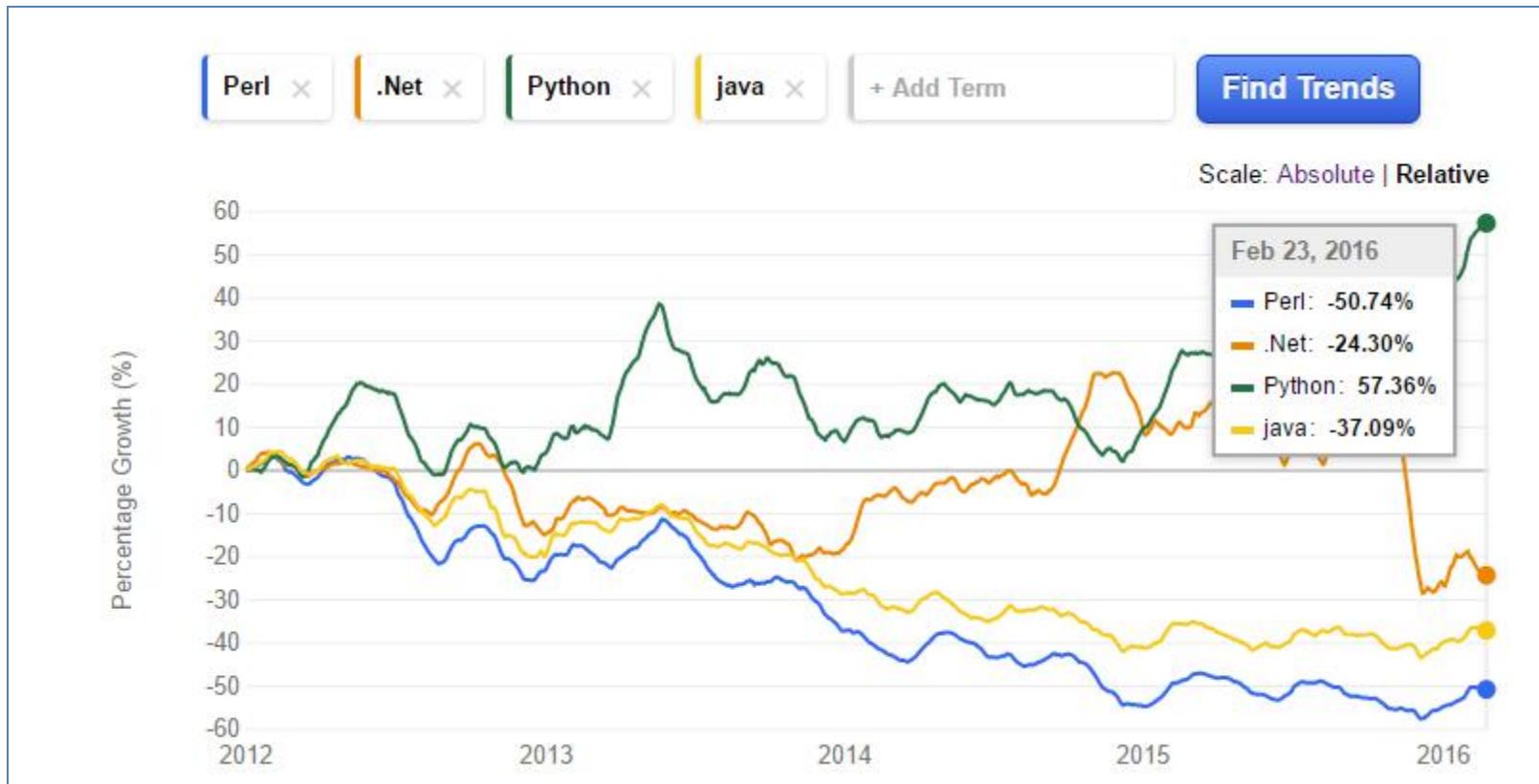
Python Features

- ✓ High Level language
- ✓ Interactive and efficient
- ✓ No compilation (Interpreter language)
- ✓ No type declarations (Hold anything)
- ✓ Automatic memory management
- ✓ High-level data types and operations
- ✓ Object-oriented programming
- ✓ Automated garbage collection
- ✓ Fewer restrictions and rules
- ✓ Wide portability
- ✓ Extendible and customizable
- ✓ Easy Debugging Techniques
- ✓ Many editors are available for programming



Who uses Python?

- ✓ Google search engine and many other products in Google makes extensive use of Python and it also employs Python's creator Guido van Rossum.
- ✓ The popular YouTube video sharing service is largely written in Python.
- ✓ Disney uses Python in many of their creative processes.
- ✓ Mozilla uses Python to explore their extensive code base and releases tons of open source packages built in python.
- ✓ Intel, Juniper, Cisco, Hewlett-Packard, Seagate, Qualcomm use Python for automated hardware testing.
- ✓ Dropbox file hosting service is implemented using Python, Guido van Rossum now working here.
- ✓ Morgan Stanley, BNP, JP Morgan, Citibank apply Python for financial market forecasting.
- ✓ NASA, Los Alamos, JPL, use Python for scientific programming tasks.
- ✓ The NSA uses Python for cryptography and intelligence analysis.
- ✓ Linux Weekly News is published using a Web application written in Python using the Quixote framework.
- ✓ The Red Hat Linux distribution uses Python for its installer (anaconda) and configuration utilities



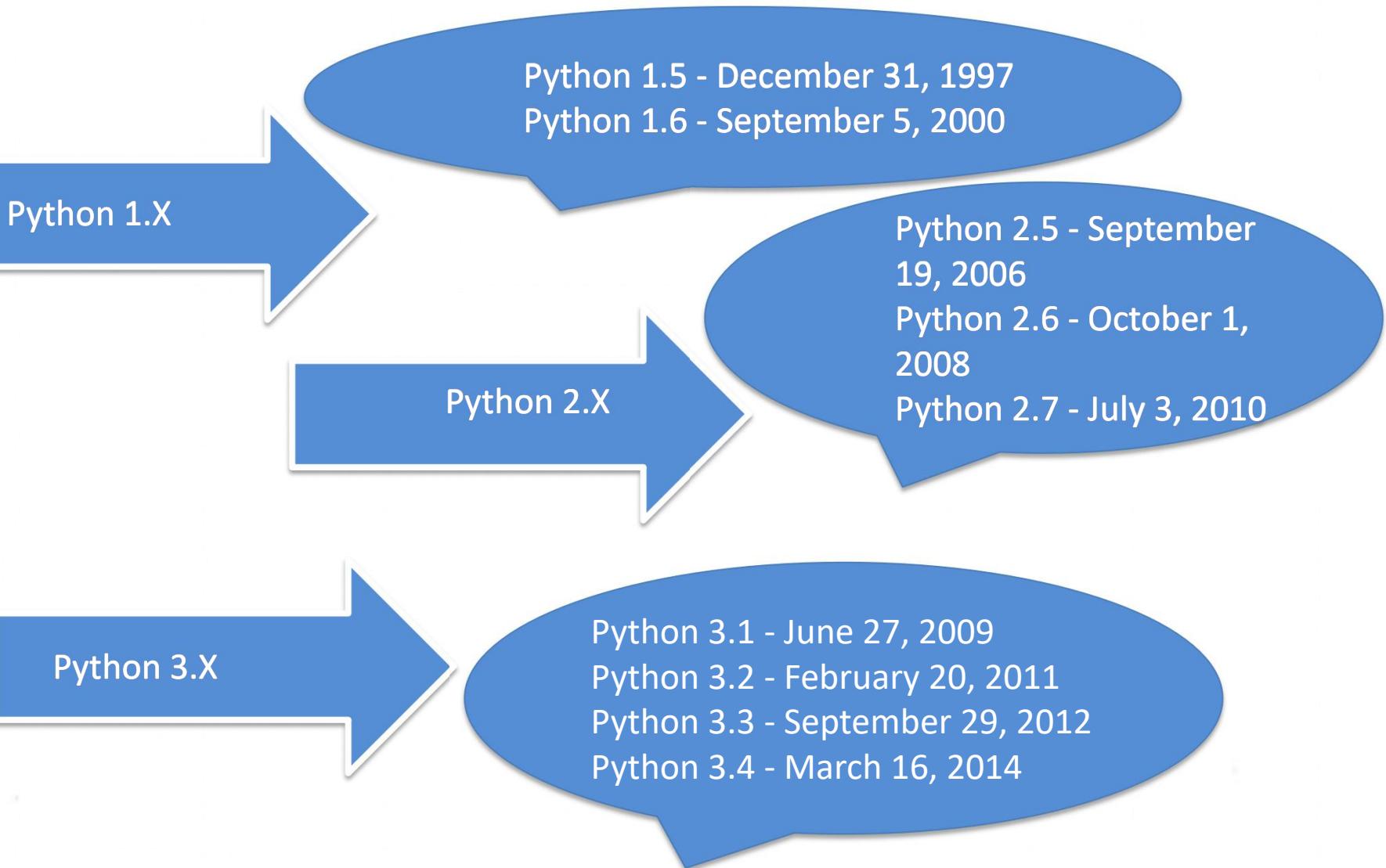
Per the indeed.com, percentage growth of Python is 500 times more than it's peer Languages.
<http://www.indeed.com/jobtrends?q=Perl%2C+.Net%2C+Python%2Cjava&l=&relative=1>

Job In Big Data space

Skill	% of Big Data Jobs Mentioning This Skill Set (multiple responses allowed)	% Growth in Demand For This Skill Set Over the Previous Year
Java	6.62%	63.30%
Structured query language	5.86%	76.00%
Apache Hadoop	5.45%	49.10%
Software development	4.70%	60.30%
Linux	4.10%	76.60%
Python	3.99%	96.90%
NoSQL	2.74%	34.60%
Data warehousing	2.73%	68.80%
UNIX	2.43%	61.90%
Software as a Service	2.38%	54.10%

Source: <http://www.forbes.com/sites/louis columbus/2014/12/29/where-big-data-jobs-will-be-in-2015/>

Python time line



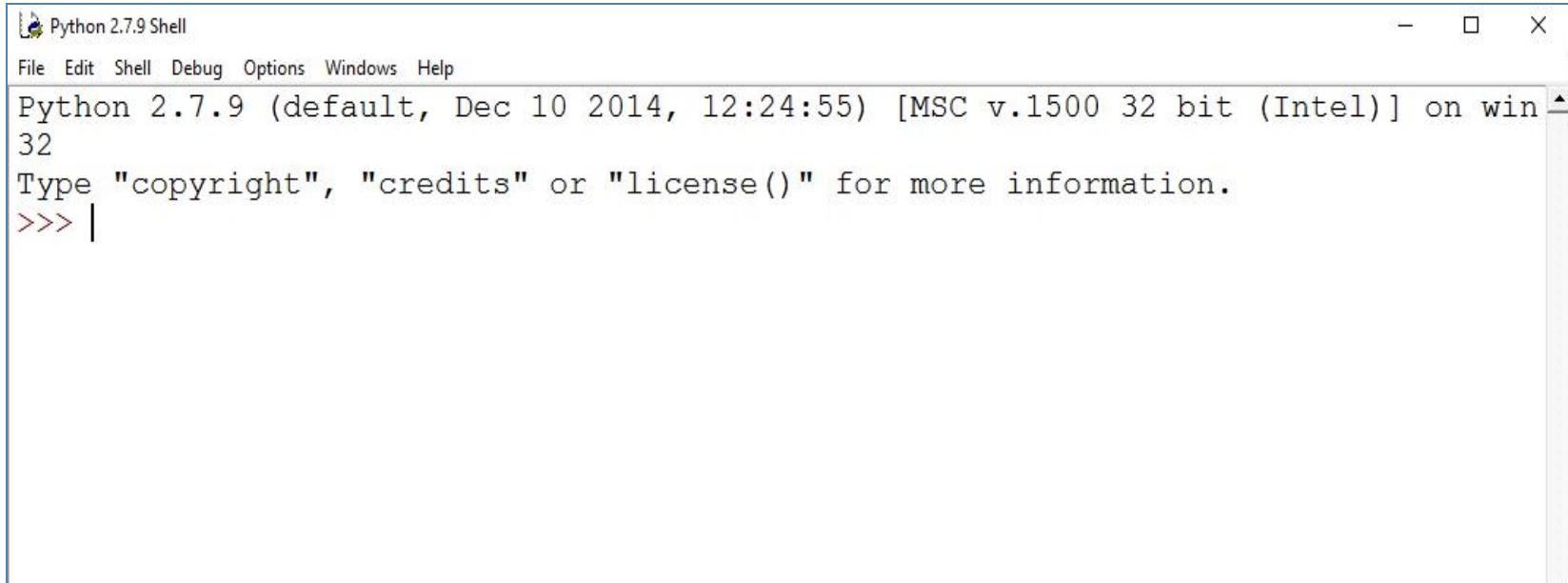
- ✓ There are couple of difference in Python2 and Python3, starting from syntax to added functionality.
- ✓ Syntax Difference example:
 - In Python 2, print is a function but doesn't required mandatory parenthesis.
 - In Python 3, print() is a function and has mandatory parentheses.
- ✓ Logic difference example:
 - In Python 2, division with integer returns integer, ex: 5/2 return 2 not 2.5
 - In Python 3, implicit division work accordingly, the above division return 2.5 as expected.
- ✓ In this Python course, we are going to use Python 2.7.X prefer to have version 2.7.9.
- ✓ There are couple of reasons to use Python 2 instead of Python 3, few of them are:
 - ✓ The library support for new Version 3.x are not adequate.
 - ✓ Most of the companies software's are using version 2.x and therefore it is better to say Python2 is most widely used version.

How to install Python?

- ✓ Python is pre-installed on almost every Unix systems, including Linux and MAC.
- ✓ In case if you need latest version , you need to update the package installation of Python with the package Installation commands of the specific OS. (In case of any help ,student can take necessary assistance from Absolute classes)
- ✓ On Windows, Python version 2.7.X need to installed explicitly.
- ✓ Latest versions are available at <https://www.python.org/downloads/>
- ✓ Installation is done as per the instruction given by the installer.
- ✓ One Python is installed on Windows user need to modify the PATH variable to get it referred on DOS prompt.
- ✓ To check whether the installation is done run the command python –V. for

```
C:\Ethans>python -V  
Python 2.7.9
```

IDLE Session

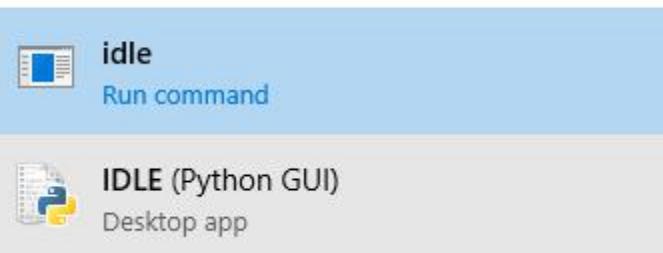


Python 2.7.9 Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Best match



Interactive shell

```
C:\Ethans>python
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)]
n32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Starting Python

```
C:\Users\jatin> python
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

Hello, World!

Interactive Mode Programming

```
>>> print "Hello, Python!"
Hello, Python!
```

Scripting Mode Programming

```
#!/usr/bin/python
print "Hello, Python!"
```

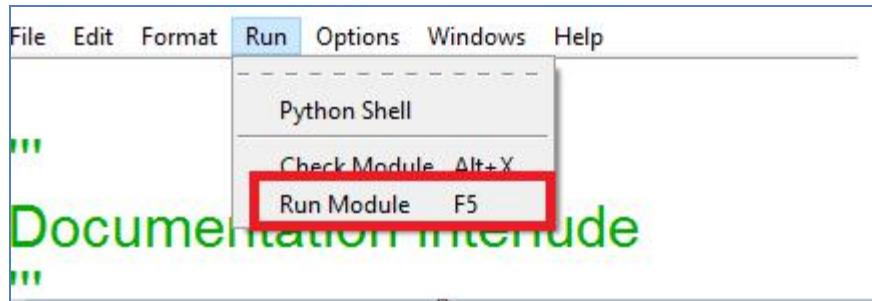
Executing program

```
$ chmod +x hello.py
$ python hello.py
```

Write first Program

```
"""\nDocumentation interlude\n"""\n# Comment 1\n\nprint 'Hello World' # comment 2
```

```
"""\nMultiline comment\nMultiline comment\n"""
```



Python Variables

- Variables starts with a letter capital A to Z or small a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Invalid Identifiers → special characters such as @, \$ and %
- Should not starts with number
- Should not names as pre-defined reserved keywords
- Variables are unlimited in length. Case is significant.

Pre-defined reserved keywords

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python Indentations

- Python blocks of code are formed by line indentation.
- The number of spaces in the indentation in block should be same with others statements within the block

Correct Example

```
A = 10
if A:
    print "True"
else:
    print "False"
```

Incorrect Example

```
A = 10
if A:
    print "Answer"
        print "True"
else:
    print "Answer"
    print "False"
```

Command and multiline statement

- Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.
- Comments in Python is created with # sign
- Python doesn't support multiline command though an another way of doing it with documentation interlude
- Statements contained within the [], {} or () brackets do not need to use the line continuation character.

```
>>> help('print')
```

The "print" statement

```
*****
```

```
print_stmt ::= "print" ([expression ("," expression)* [","]]  
    | ">>" expression [(", " expression)+ [","]])
```

"print" evaluates each expression in turn and writes the resulting object to standard output (see below). If an object is not a string, it is first converted to a string using the rules for string conversions. The (resulting or original) string is then written. A space is written before each object is (converted and) written, unless the output system believes it is positioned at the beginning of a line. This is the case (1) when no characters have yet been written to standard output, (2) when the last character written to standard output is a whitespace character except " " ", or (3) when the last

- Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.
- Comments in Python is created with # sign
- Python doesn't support multiline command though an another way of doing it with documentation interlude
- Statements contained within the [], {} or () brackets do not need to use the line continuation character.

Quiz:

1 – From the following, what are the invalid variables?

- a. myVariable
- b. Var1
- c. X
- d. X
- e. _x
- f. 1var
- g. my#Variable

2 – What will be the output of below program?

```
print 'One' ,  
print 'Two'  
# print 'Threee'  
"  
print 'Four'  
""
```

Facts at a glance

- **Python syntax is line oriented and space sensitive rather than free format**
- **Python's built-in types include numeric primitives and container types**
- **Integers hold arbitrary precision values**
- **Both floating-point and complex numbers supported in the language**
- **Python has expected set of operators**

Interesting Fact

- Monty Python's Flying Circus is a British sketch comedy series created by the comedy group Monty Python and broadcast by the BBC from 1969 to 1974.
- The first episode was recorded on 7 September and broadcast on 5 October 1969 on BBC One, with 45 episodes airing over four series from 1969 to 1974, plus two episodes for German TV.
- The members of Monty Python were highly educated.
- Terry and Michael are Oxford graduates.
- Eric Idle, John Cleese, and Graham Chapman attended Cambridge University.
- American-born member Terry Gilliam is an Occidental College graduate.

Source - Wiki

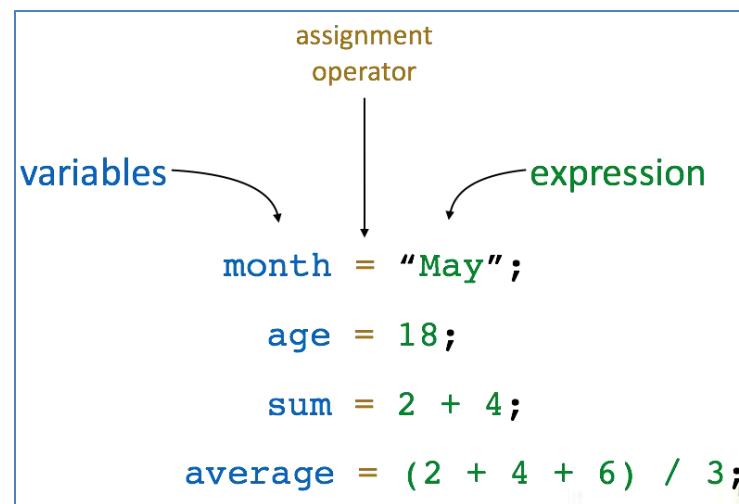
Core Objects and built in functions

- Python Core Objects and builtin functions
- Number Object and operations
- String Object and Operations
- List Object and Operations
- Tuple Object and operations
- Dictionary Object and operations
- Set object and operations
- Boolean Object and None Object
- Different data Structures, data processing

- Numbers
 - ex: 1, 1.1, 3 + 4j and 200L
- Strings
 - ex: 'Ethans' "Tech"
- List
 - ex: ['Ethans', "Tech"]
- Tuple
 - ex: ('Ethans', "Tech")
- Dictionary -
 - ex: {'Institute': 'Ethans'}
- Set
 - ex: set([1,2,3,4])
- Files
 - ex: file = open('file.txt')
- None
 - ex: None
- Boolean
 - ex: True, False

Assignment Operator

```
>>> number = 10
>>> name = 'Ethan'
>>> location = 'Pune'
>>> number1 = number2 = number3 = 1
>>> # Multiple assignment
>>> number1, number2 = 10, 20
```



- ✓ Integers
- ✓ Float
- ✓ Long
- ✓ Complex

```
>>> number = 10
>>> # Number is an object of interger class
>>>
>>> number = 10.1
>>> # Number is an object of float class
>>>
>>> number = 3 + 4j
>>> # Number is an object of complex class
>>>
>>> number = 200987654323456789876543L
>>> # Number is an object of long class
```

type function

```
>>> number_int = 10
>>> type(number_int)
<type 'int'>
>>>
>>> number_int = 2147483647
>>> type(number_int)
<type 'int'>
>>> number_int = 2147483648
>>> type(number_int)
<type 'long'>
>>> number_comp = 3 + 4j
>>> type(number_comp)
<type 'complex'>
>>> number_float = 3.7
>>> type(number_float)
<type 'float'>
```

Builtin functions

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

```
>>> help('abs')
```

Help on built-in function abs in module __builtin__:

abs(...)

abs(number) -> number

Return the absolute value of the argument.

```
>>> abs(-10)
```

10

```
>>> abs(-10.1)
```

10.1

```
>>> abs(-1000L)
```

1000L

```
>>> abs(3+4j)
```

5.0

```
>>> help('__builtin__')
```

Help on built-in module __builtin__:

NAME

__builtin__ - Built-in functions, exceptions, and other objects.

FILE

(built-in)

DESCRIPTION

Noteworthy: None is the 'nil' object; Ellipsis represents '...' in slices.

CLASSES

Number functions

```
>>> int('1010', 2)
10
>>> float(10)
10.0
>>> long(12)
12L
>>> complex(1, 2)
(1+2j)
```

```
>>> cmp(1,2)
-1
>>> cmp(3,2)
1
>>> cmp(2,2)
0
```

```
>>> isinstance(1, int)
True
>>> isinstance(1.1, float)
True
>>> isinstance(1L, float)
False
>>> isinstance(complex(1,2), complex)
True
```

Operation on numbers

```
>>> 10 + 2  
12  
>>> 10 -2  
8  
>>> 10 * 2  
20  
>>> 10 / 2  
5  
>>> 10 * 2 ** 2 #operator precedence  
40  
>>> (10 * (2 + (6 * 7)) * 2) # operator associativity  
880
```

Precedence Table



Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //=- += *= **=	Assignment operators
is, is not	Identity operators
in, not in	Membership operators
and, or, not	Logical operators



Operation on numbers

```
>>> -5 ** 2  
-25  
>>> (-5) ** 2  
25  
>>> 5 -- 2  
7  
>>> bin(10)  
'0b1010'  
>>> bin(-10)  
'-0b1010'  
>>> bin(11)  
'0b1011'  

```

```
>>> 10 << 2  
40  
>>> 10 >> 2  
2  
>>> 10 & 2  

```

```
>>> name = 'Ethans'  
>>> location = "Pune" # Double quotes  
>>> expertTraining = """  
Python  
Hadoop  
Selenium  
DevOps  
Informatica  
ETL  
"""
```

String Functions

```
>>> type(name)
<type 'str'>
>>> str(10)
'10'
>>> repr(10)
'10'
>>> `10`
'10'
>>> isinstance('Ethan', str)
True
>>> isinstance('Ethan', basestring)
True
>>> ord('a')
97
>>> chr(65)
'A'
```

```
>>> len(name)
6
```

String Operations

```
>>> name
'Ethans'
>>> lname = 'Technologies'
>>>
>>> print name + lname
EthansTechnologies
>>> print "***" * 20, "\n" + name + lname + "\n", "***" * 20
*****
EthansTechnologies
*****
```

String Indexing

```
>>> name[0]
'E'
>>> name[-1]
's'
>>> name[3]
'a'
>>> name[4]
'n'
>>> name[5]
's'
>>> name[len(name)]
```

Traceback (most recent call last):

```
  File "<pyshell#101>", line 1, in <module>
    name[len(name)]
```

IndexError: string index out of range

String Slicing

```
>>> # Slicing Syntax
>>> # string[START:STOP:STEP]
>>>
>>> name[0:5:1]
'Ethan'
>>> name[0:]
'Ethans'
>>> name[::]
'Ethans'
>>> name[:9]
'Ethans'
>>> name[::-2]
'Ehn'
```

```
>>> name[-1:-5]
"
>>> name[-1:-5:-1]
'snah'
>>> name[::-1]
'snahtE'
>>> name[::-2]
'sat'
```

String Formatting

```
>>> print "We at %s turning %d today" %('Ethans', 2)
We at Ethans turning 2 today
>>> name, age = 'Ethans', 2
>>> print "We at %s turning %d today" %(name, age)
We at Ethans turning 2 today
>>> print "We at %s turning %s today" %(name, age)
We at Ethans turning 2 today
>>> print "We at %r turning %r today" %(name, age)
We at 'Ethans' turning 2 today
>>>
>>> print 'This is my File name: C:\name\test'
This is my File name: C:
ame      est
>>> print r'This is my File name: C:\name\test'
This is my File name: C:\name\test
```

The list is another datatype in Python which can be written with comma-separated values (items) in a square brackets.

Items in a list should be of same type or different type.

```
>>> names = ['Ethans', 'Ethan', 'Ethan Tech']
>>> names[0]
'Ethans'
>>> names[-1]
'Ethan Tech'
>>> names[-2]
'Ethan'
>>> type(names)
<type 'list'>
```

List Functions

```
>>> names
['Ethans', 'Ethan', 'Ethan Tech']
>>> len(names)
3
>>> any(names)
True
>>> all(names)
True
>>> names + names
['Ethans', 'Ethan', 'Ethan Tech', 'Ethans', 'Ethan', 'Ethan Tech']
>>> names * 2
['Ethans', 'Ethan', 'Ethan Tech', 'Ethans', 'Ethan', 'Ethan Tech']
```

List Slicing and range function

```
>>> names
['Ethans', 'Ethan', 'Ethan Tech']
>>> # names[START:STOP:STEP]
>>>
>>> names[0:]
['Ethans', 'Ethan', 'Ethan Tech']
>>> names[0:2]
['Ethans', 'Ethan']
>>> names[-1:-3]
[]
>>> names[-1:-3:-1]
['Ethan Tech', 'Ethan']
```

```
>>> # range(START:STOP:STEP)
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10,2)
[1, 3, 5, 7, 9]
```

List Data Structure

```
>>> # List of List, 2 dimensional
>>> names_age = [ ['Jatin', 35], ['Rahul', 16], ['Vijay', 30]]
>>> names_age[0]
['Jatin', 35]
>>> names_age[0][0]
'Jatin'
>>> names_age[-1]
['Vijay', 30]
>>> names_age[-1][1]
30
```

Tuple is another datatype in Python which can be written with comma-separated values (items) in a brackets.

Items in a tuple should be of same type or different type doesn't matter.

```
>>> names = ('Ethans', 'Ethan', 'Ethan Tech')
>>> type(names)
<type 'tuple'>
>>> names = ('Ethans')
>>> type(names)
<type 'str'>
>>> names = ('Ethans',)
>>> type(names)
<type 'tuple'>
```

Difference between Tuple and List

```
>>> names_tuple = ('Ethans', 'Ethan', 'Ethan Tech')
>>> names_list = ['Ethans', 'Ethan', 'Ethan Tech']
>>>
>>> names_list[0] = 'ETHANS'
>>> names_tuple[0] = 'ETHANS'
```

Traceback (most recent call last):

```
  File "<pyshell#225>", line 1, in <module>
    names_tuple[0] = 'ETHANS'
TypeError: 'tuple' object does not support item assignment
>>>
>>> # Mutable and immutable property of an object
```

Tuple Data Structure

```
>>> names_tuple = (('Ethans', 'Ethan'), ('Ethan Tech',))  
>>> len(names_tuple)  
2  
>>> names_tuple[0]  
('Ethans', 'Ethan')  
>>> names_tuple[0][1]  
'Ethan'
```

```
>>> names_tuple_list =(['Ethans', 'Ethan'], ['Ethan Tech'])  
>>> names_tuple[0][1]  
'Ethan'  
>>> names_tuple[0][1] = 'ETHANS'
```

Traceback (most recent call last):

```
  File "<pyshell#241>", line 1, in <module>  
    names_tuple[0][1] = 'ETHANS'
```

TypeError: 'tuple' object does not support item assignment

```
>>>
```

```
>>> names_tuple_list[0][1] = 'ETHANS'
```

Dictionary

Another builtin datatype in Python where each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.

```
>>> details = {'Name':'Ethans', 'Age': 2, 'Location':'Pune'}  
>>> details[0]
```

Traceback (most recent call last):

```
  File "<pyshell#250>", line 1, in <module>  
    details[0]  
KeyError: 0
```

```
>>> details['Name']  
'Ethans'  
>>> print details # Random order  
{'Age': 2, 'Name': 'Ethans', 'Location': 'Pune'}
```

Add and Remove Dictionary Item

```
>>> details
{'Age': 2, 'Name': 'Ethans', 'Location': 'Pune'}
>>> details['Name'] = 'Ethans Tech'
>>> details['Technology'] = 'Python' # Adding New Element
```

```
>>> details
{'Age': 2, 'Technology': 'Python', 'Name': 'Ethans Tech', 'Location': 'Pune'}
>>> del details['Age']
>>> details
{'Technology': 'Python', 'Name': 'Ethans Tech', 'Location': 'Pune'}
```

Dictionaries and List, Tuple

```
>>> print tuple(details)
('Technology', 'Name', 'Location')
>>> print list(details)
['Technology', 'Name', 'Location']
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict([('two', 2), ('one', 1), ('three', 3)])
>>> a == b == c
True
```

Dictionaries data Structure

```
>>> emp = {'Names':['Ethan', 'Ethans'], 'Location':['Pune', 'Bangalore']} # DoL
>>> emp = {'Names':('Ethan', 'Ethans'), 'Location':('Pune', 'Bangalore')} # DoT
>>> emp = {'Names':{'1':'Ethan', 2:'Ethans'}, 'Location':{1:'Pune',2:'Bangalore'}} # DoD
>>> emp
{'Names': {'1': 'Ethan', 2: 'Ethans'}, 'Location': {1: 'Pune', 2: 'Bangalore'}}
>>> emp['Names'][1]
'Ethan'
>>> emp['Location'][1]
'Pune'
```

Set Object

```
>>> managers = set(['Aakash', 'Rahul', 'Bob'])
>>> engineers = set(['Rahul', 'Vijay'])
>>> type(managers)
<type 'set'>
>>> # Get the intersection
>>> managers & engineers
set(['Rahul'])
>>> managers - engineers # elements available in managers not in engineers
set(['Bob', 'Aakash'])
>>> managers | engineers # elements available in both
set(['Bob', 'Vijay', 'Aakash', 'Rahul'])
```

Boolean and None Object

```
>>> yes = True
>>> type(yes)
<type 'bool'>
>>> no = False
>>> type(no)
<type 'bool'>
>>>
>>> Null = None
>>> type(Null)
<type 'NoneType'>
```

True and False are the pre defined objects in Python, when comparing the value or doing comparison operation Python returns either True or False.

None is another object in Python. It is similar as NULL in database.

Membership Operators

Python membership operators test for membership in a sequence or an iterable, such as strings, lists, or tuples.

in

Evaluates to `true` if it finds a variable in the specified sequence and `false` otherwise.

not in

Evaluates to `true` if it does not find a variable in the specified sequence and `false` otherwise.

```
>>> 'is' in 'This is a string'  
True  
>>> 'IS' not in 'This is a string'  
True  
>>> 1 in range(10)  
True  
>>> 10 not in range(10)  
True
```

Python Identity Operator

is

Evaluates to **true** if the variables on either side of the operator point to the same object and **false** otherwise.

is not

Evaluates to **false** if the variables on either side of the operator point to the same object and **true** otherwise.

```
>>> number1 = 10
>>> number2 = 300
>>>
>>> number1 is number2
False
>>> number1 is not number2
True
>>> number1 is 10
True
>>> number2 is 300
False
```

Python Identity Operator

is

Evaluates to `true` if the variables on either side of the operator point to the same object and `false` otherwise.

is not

Evaluates to `false` if the variables on either side of the operator point to the same object and `true` otherwise.

```
>>> name = 'Ethan'  
>>> name is 'Ethan'  
True  
>>> names = ['Ethans', 'Ethan', 'Ethan Tech']  
>>> names2 = names  
>>> name is names  
False  
>>> names2[0] = 'ETHANS'  
>>> names  
['ETHANS', 'Ethan', 'Ethan Tech']
```

Quiz:

1 – What will be the output of below program?

```
>>> sum([1,2,3,4,5])
```

2 – What will be the output of below program?

```
>>> name = 'Ethans Technologies'  
>>> name[0:6]
```

3 – What will be the output of below program?

```
>>> name = 'Ethans Technologies'  
>>> name[-1:0]
```

Interesting Fact

Raspberry Pi is a card-sized, inexpensive microcomputer that is being used for a surprising range of exciting do-it-yourself stuff such as robots, remote-controlled cars, and video game consoles.

With Python as its main programming language, the Raspberry Pi is being used even by programmers to build radios, cameras, arcade machines, and pet feeders!

With Raspberry Pi mania on the uptrend, there are countless DIY projects, tutorials, and books to choose from online.

These will help you branch out from your “hello world” starter programs to something you can truly be proud of.

Conditional Statements and Loops

- What are conditional statements?
- How to use the indentations for defining if, else, elif block
- What are loops?
- How to control the loops
- How to iterate through the various object
- Sequence and iterable objects

Choices in Real life

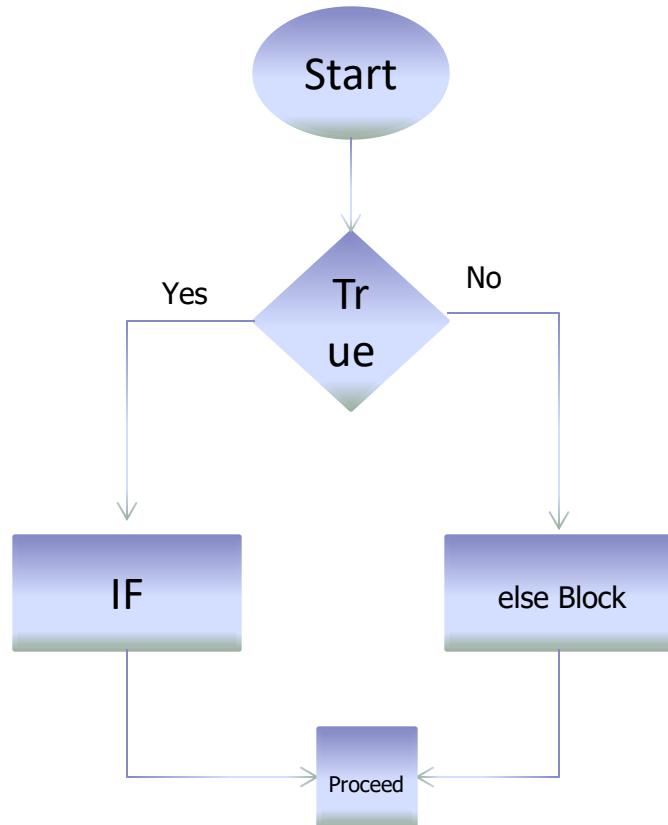
Honey, Shall
we go for
movie today
evening?



Condition statements



Only, **If** I
come back
from office
by 5 or **else**
we shall go
tomorrow.



- Variable alone doesn't support conditions.
- If-elif-else like clauses used to make conditions based on some pre-conditions.
- In Python, we have the reserved keywords like if, else, elif and unless for conditions.
- The statements to be executed are enclosed within braces indentation block
- All defined objects like numbers, string, list, tuple, dict etc returns true when they are defined with true value.

In module 2, we have listed the condition operators.

In Python, we have following operators which evaluate either True or False

Type	Operators
Conditional Operator	<code>==</code> (equals), <code>!=</code> (Not equal), <code>></code> (greater than), <code><</code> (less than), <code><=</code> (less then or equals), <code>>=</code> (greater than or equals)
Membership Operator	<code>In</code> , <code>not in</code>
Identity Operator	<code>Is</code> , <code>is not</code>

If - Else Example

```
number = 10
```

```
if number: # Not condition, return True or False based on Value
    print "Got a true expression value ",
    print number
```

```
number = 0
```

```
if number: # Check the indentation of the block
    print "2 - Got a true expression value ",
    print number
```

```
else:
```

```
    print "2 - Got a False expression value ", number
```

Python conditions is either True or False.

Following are certain example of evaluated False value in the conditions.

- 0, 0.0, 0L - Numbers
- '' – An Empty String
- [] – An Empty List
- {} - An Empty dictinoary
- () - An Empty tuple
- None
- False
- Set() – An Empty set

```
number = 10
if number == 100:
    print "1 - The number is equals to 100",
    print number
elif number == 150:
    print "2 - The number is equals to 150",
    print number
elif number == 200:
    print "3 - The number is equals to 200",
    print number
else:
    print "4 - Got a different value"
```

```
number1, number2, number3 = 10, 20, 0
```

```
if number1 and number2: # If both number1 and number2 are True  
    print "number1 and number2 are True"
```

```
else:
```

```
    print "Either number1 is not true or number2 is not true"
```

```
if number1 or number2:
```

```
    print "Either number1 is true or number2 is true or both are True"
```

```
else:
```

```
    print "Both number1 and number2 are False"
```

Repeated Action



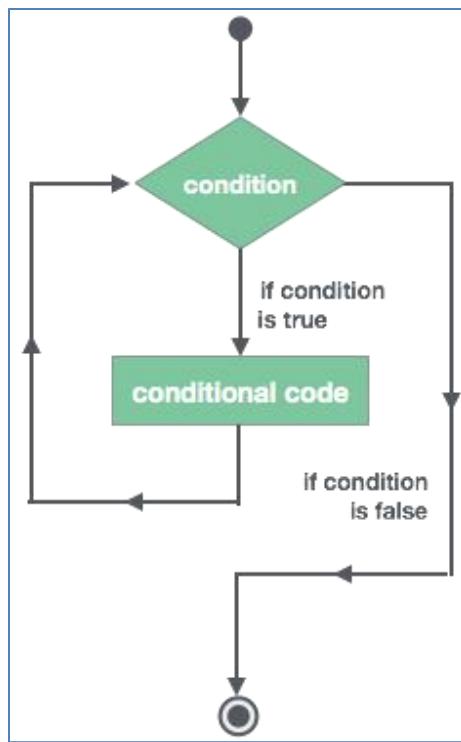
Loops are the repetitive action/s to perform any task.

They are also called as iterative constructs statements.

Python provide us below reserved keywords for repetitive actions

- While loop
- For loop

- while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.



Syntax

```
while expression:  
    statement(s)
```

While Loop - Example

```
count = 0
```

```
# Indentation is required for while loop
while count < 9:
    print 'The count is:', count
    count = count + 1
```

- A loop becomes infinite loop if a condition never becomes false.
- This results in a loop that never ends. Such a loop is called an *infinite loop*.
- Example - client/server programming.

Example

```
var = 1
while var == 1 : # This constructs an infinite loop
    num = raw_input("Enter a number :")
    print "You entered: ", num

print "Good bye!"
```

Note - It will go in an infinite loop and use CTRL+C to come out of the program.

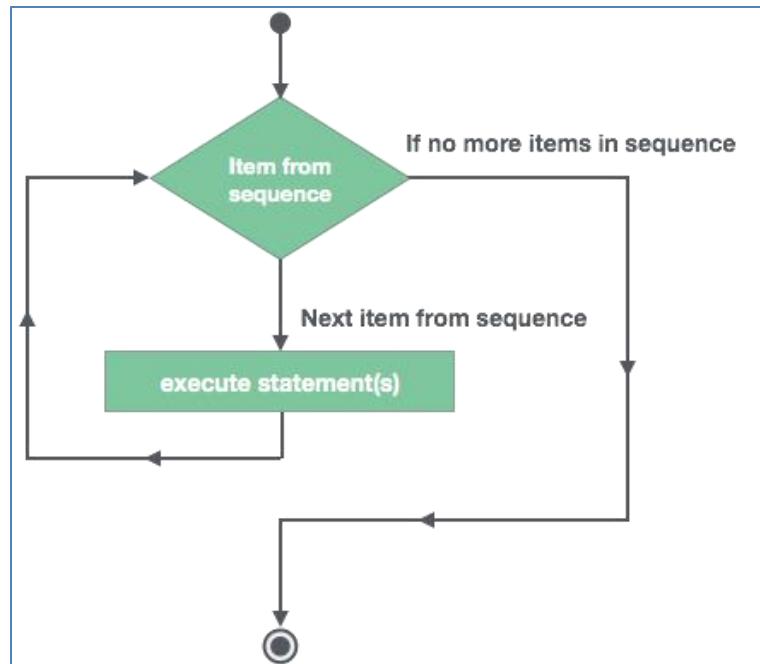
- Python supports to have an else statement associated with a loop statement.
- When the else statement is used with a while loop, the else statement is executed when the condition becomes false.

```
count = 0
```

```
# Indentation is required for while loop
while count < 9:
    print 'The count is:', count
    count = count + 1
else:
    print 'While loop completed'
```

For loop

- The for loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.



Syntax

```
for iterating_var in sequence:  
    statements(s)
```

```
for letter in 'Python':    # First Example
    print 'Current Letter :', letter
```

```
fruits = ['banana', 'apple', 'mango']
for fruit in fruits:        # Second Example
    print 'Current fruit :', fruit
```

```
for index in range(len(fruits)): # Third Example
    print 'Current fruit :', fruits[index]
```

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following loop control statements –
 - `break` statement
 - `continue` statement
 - `pass` statement

- The `break` statement in Python terminates the current loop and resumes execution at the next statement.
 - The `break` statement can be used in both `while` and `for` loops.
-
- The `continue` statement in Python returns the control to the beginning of the `while` or `for` loop.
 - The `continue` statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
-
- The `pass` statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
 - The `pass` statement is a null operation; nothing happens when it executes.

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. It is built over the Simple DirectMedia Layer (SDL) library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games (mainly the graphics part) can be abstracted from the game logic, making it possible to use a high-level programming language, such as Python, to structure the game.

Pygame was built to replace PySDL after its development stalled.[4] Pygame was originally written by Pete Shinners and is released under the open source free software GNU Lesser General Public License. It has been a community project since 2004 or 2005.

UDF Functions and Object Functions

- What are various type of functions
- Create UDF functions
- Parameterize UDF function, through named and unnamed parameters
- Defining and calling Function
- The anonymous Functions - Lambda Functions
- String Object functions
- List and Tuple Object functions
- Dictionary Object functions

What are Functions?

- A function is a set of instructions organized, which is used to perform a single, related actions.
- Functions provide better modularity to a program and a high degree of freedom to reuse existing codes.

We can have different function in Python, most common functions are:

- Builtin functions
- Object functions
- User defined functions
- Module function

To take the help on function

```
>>> help('NAME OF THE FUNCTION')
```

Help on built-in function

User defined functions are common Python function which we create::

Here how we create UDF:

- Function blocks begin with the keyword **def** followed by the function name and parentheses.
- Input parameters or arguments should be placed within the parentheses. Parameters can also be defined inside these parentheses.
- The first non executable statement of a function can be an optional statement – the documentation string of the function or **docstring**.
- The code block within every function starts with a colon (:) and is indented.
- A return statement with no arguments returns None.

UDF Functions Example

```
def NoParam():
    """
    DocString of the function
    """

    print 'First Executable Statement'
    return

def OneParam(a):
    print 'First Executable Statement'
    print a
    return a

# Call function
NoParam()
value = OneParam(10)
print value
```

Pass by Reference or value

- All the parameters in Python are passed by reference as all datatypes are Objects
- If you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
def OneParam(a):
    a.append(4)
    return a

# Call function
a = OneParam([1,2,3])
print a
```

How to pass parameters

You can call a function by using the following types of arguments:

- Mandatory arguments
- Optional or Default arguments
- Any number of unnamed arguments
- Any number of named arguments

Mandatory Arguments

```
# Accept only one arguments
def OneParam(a):
    print a

# Call function
OneParam([1,2,3])

# If we pass more than one arguments it will raise an issue
OneParam([1,2,3], 10)

# Accept only two arguments
def twoParam(a, b):
    print a, b

# Calling with two arguments
twoParam ([1,2,3], 10)
```

Mandatory Arguments

```
# Accept only one arguments
def OneParam(a):
    print a

# Call function
OneParam(a = [1,2,3]) # Fine

# If we pass more than one arguments it will raise an issue
OneParam(a = [1,2,3], 10) # Error

# Accept only two arguments
def twoParam(a, b):
    print a, b

# Calling with two arguments
twoParam (b = [1,2,3], a = 10) # Fine
twoParam (b = [1,2,3], 10)     # Error
```

Default Arguments

```
# Accept one default arguments
def OneParam(a=10):
    print a

# Call function
OneParam(a = [1,2,3]) # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Error

# Accept only two arguments
def twoParam(a, b=20):
    print a, b

# Calling with two arguments
twoParam (b = [1,2,3], a = 10) # Fine
twoParam (10) # Fine
```

N unnamed Arguments

```
# Accept n number of default arguments
def OneParam(*a):
    print a

# Call function
OneParam([1,2,3]) # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Fine

# Accept one mandatory and any arguments
def twoParam(a, *b):
    print a, b

# Calling with two arguments
twoParam (a = 10, 20, 30) #Error
twoParam (10, 20, 30, 40) # Fine
```

N named Arguments

```
# Accept n number of default arguments
def OneParam(**a):
    print a

# Call function
OneParam(c=[1,2,3]) # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Error
OneParam(b=10, c=20) # Fine

# Accept one mandatory and any arguments
def twoParam(a, **b):
    print a, b

# Calling with two arguments
twoParam (a = 10, b=20, c=30) #Fine
twoParam (10, 20, 30, 40) # Error
```

Lambda Functions

- lambda keyword to create one liner anonymous functions.
- These functions are called anonymous because they are not declared in the standard manner by using the def keyword.
- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- Anonymous functions cannot contain commands or multiple expressions.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Lambda Functions

```
# Accept n number of default arguments
def OneParam(a):
    print a + 10

# Function using Lambda
oneParam = lambda a : a + 10
print oneParam(10)
```

String functions

```
>>> string = 'ethans'

>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize',
 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index',
 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

String functions help

```
>>> help(string.capitalize)
```

Help on built-in function capitalize:

capitalize(...)

S.capitalize() -> string

Return a copy of the string S with only its first character capitalized.

```
>>> help(string.upper)
```

Help on built-in function upper:

upper(...)

S.upper() -> string

Return a copy of the string S converted to uppercase.

String functions executions

```
>>> string.capitalize()
```

```
'Ethans'
```

```
>>> string.center(10, 'X')
```

```
'XXethansXX'
```

```
>>> string.count('e')
```

```
1
```

```
>>> string.encode('base64')
```

```
'ZXRoYW5z\n'
```

```
>>> 'ZXRoYW5z\n'.decode('base64')
```

```
'ethans'
```

```
>>> string.endswith('ns')
```

```
True
```

```
>>> string.find('a')
```

```
1
```

String functions executions

```
>> 'I am {0} and I am {1} years old'.format('Ethans', 2)
'I am Ethans and I am 2 years old'
>>> string.index('a')
3
>>> string.isalnum()
True

>>> string.isalpha()
True
>>> string.islower()
True
>>> string.isspace()
False
>>> string = 'This is the sample string'
>>> allWords = string.split(' ')
>>> allWords
['This', 'is', 'the', 'sample', 'string']
>>> '|'.join(allWords)
'This|is|the|sample|string'
```

List functions

```
>>> allWords
```

```
['This', 'is', 'the', 'sample', 'string']
```

```
>>> dir(allWords)
```

```
'__add__', '__class__', '__contains__', '__delattr__', '__delitem__',  
'__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',  
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',  
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',  
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',  
'__rmul__', '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__',  
'__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',  
'reverse', 'sort']
```

List functions help

```
>>> help(allWords.append)
```

Help on built-in function append:

append(...)

L.append(object) -- append object to end

```
>>> help(allWords.extend)
```

Help on built-in function extend:

extend(...)

L.extend(iterable) -- extend list by appending elements from the iterable

List functions Executions

```
>>> allWords.append('New Word')
>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word']

>>> allWords.count('is')
1
>>> allWords.extend(['New List1', 'New List2'])

>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2']

>>> allWords.index('is')
1
```

List functions Executions

```
>>> allWords.insert(10, 'New List3')
>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2', 'New
List3']

>>> allWords.pop()
'New List3'

>>> allWords.remove('is')

>>> allWords
['This', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2']

>>> allWords.reverse()

>>> allWords
['New List2', 'New List1', 'New Word', 'string', 'sample', 'the', 'This']
```

Sorting

```
>>> help(allWords.sort)
```

Help on built-in function sort:

```
sort(...)  
L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;  
cmp(x, y) -> -1, 0, 1
```

```
>>> numbers = [1,2,11,3,44,5,22,6,8,9]  
>>> numbers.sort()  
>>> numbers  
[1, 2, 3, 5, 6, 8, 9, 11, 22, 44]
```

```
>>> allWords.sort()  
>>> allWords  
['New List1', 'New List2', 'New Word', 'This', 'sample', 'string', 'the']  
# ASCII based sorting
```

Sorting

```
>>> names = ['Amit', 'aakash', 'Ajay', 'Ashish']
>>> names.sort()
>>> names
['Ajay', 'Amit', 'Ashish', 'aakash']
>>> names.sort(key=str.upper)
>>> names
['aakash', 'Ajay', 'Amit', 'Ashish']
```

```
>>> names_age = [['Jatin', 35], ['Rahul', 20], ['Vijay', 47]]
>>> names_age.sort(key=lambda x:x[1], reverse=True)
>>> names_age
[['Vijay', 47], ['Jatin', 35], ['Rahul', 20]]

>>> def age_sort(list1):
        return list1[1]
>>> names_age.sort(key=age_sort, reverse=True)
>>> names_age
[['Vijay', 47], ['Jatin', 35], ['Rahul', 20]]
```

Tuple Functions

```
>>> names = ('Rahul', 'Vijay', 'Aakash')
>>> type(names)
<type 'tuple'>
>>> dir(names)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__getslice__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__',
 '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
>>>
```

Tuple Functions

```
>>> help(names.count)
```

Help on built-in function count:

count(...)

T.count(value) -> integer -- return number of occurrences of value

```
>>> names.count('Rahul')
```

1

```
>>> names.index('Rahul')
```

0

Dictionary functions

```
>>> dict1 = {1:2, 3:4}
>>> dir(dict1)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
 '__getitem__', '__gt__', '__hash__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
 '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'has_key',
 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop',
 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys',
 'viewvalues']
```

Dictionary functions example

```
>>> help(dict1.clear)
Help on built-in function clear:

clear(...)
    D.clear() -> None. Remove all items from D.
```

Example:

```
>>> dict1.clear()
>>> dict1
{}
>>> dict1 = {1:2, 3:4, 5:6}
>>> dict2 = dict1.copy()
>>> dict2
{1: 2, 3: 4, 5: 6}
>>> dict2[1] = 10
>>> print dict1[1]
2
```

Dictionary functions example

```
>>> dict1 = {1: 20, 3: 4, 5: 6}
>>> dict1.fromkeys(dict1, 10)
{1: 10, 3: 10, 5: 10}
>>> dict1.get(1)
20
>>> dict1.get(2, 10)
10
>>> dict1.get(1, 10)
20
>>> dict1.get(4)
>>> dict1.has_key(1)
True
>>> dict1.has_key(10)
False
```

Dictionary functions example

```
>>> help(dict1.items)
```

Help on built-in function items:

items(...)

D.items() -> list of D's (key, value) pairs, as 2-tuples

```
>>> employee = {'Jatin':3500, 'Rahul':1500, 'Aakash':1000}
```

```
>>> employee1 = employee.items()
```

```
>>> print employee1
```

```
>>> [('Jatin', 3500), ('Rahul', 1500), ('Aakash', 1000)]
```

```
>>> for i, j in employee.iteritems():
```

```
    print i, j
```

```
Aakash 1000
```

```
Jatin 3500
```

```
Rahul 1500
```



Dictionary functions example

```
>>> for i in employee.itervalues():
    print i
1000
3500
1500
>>> for i in employee.iterkeys():
    print i
Aakash
Jatin
Rahul

>>> employee.keys()
['Aakash', 'Jatin', 'Rahul']
>>> employee.values()
[1000, 3500, 1500]
>>> employee.pop('Aakash')
1000
>>> employee.popitem()
('Jatin', 3500)
```

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. It is built over the Simple DirectMedia Layer (SDL) library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games (mainly the graphics part) can be abstracted from the game logic, making it possible to use a high-level programming language, such as Python, to structure the game.

Pygame was built to replace PySDL after its development stalled.[4] Pygame was originally written by Pete Shinners and is released under the open source free software GNU Lesser General Public License. It has been a community project since 2004 or 2005.

File Handling with Python

- Process text files using Python
- Read/write and Append file object
- File object functions
- File pointer and seek the pointer
- Truncate the file content and append data
- File test operations using os.path

Open function

Open is the built-in function in Python, which is used to interact with the flat files in all the operating systems.

Help on the function is bellow:

```
>>> help(open)
```

Help on built-in function open in module `__builtin__`:

`open(...)`

`open(name[, mode[, buffering]]) -> file object`

Open a file using the `file()` type, returns a file object. This is the preferred way to open a file. See `file.__doc__` for further information.

Open mode

Mode	What it do
r	Open the file in read mode
w	Open the file in write mode
a	Open the file in append mode
r+	Open the file in read and write mode
w+	Open the file in write and read mode
a+	Open the file in append and read mode
rb	Open the file in read binary mode
wb	Open the file in write binary mode
ab	Open the file in append binary mode

Open file

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt')

>>> dir(infile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
 '__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
 'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write',
 'writelines', 'xreadlines']
```

- Default mode is read
- Default buffering is enable
- "r" – is used to make file path raw
- Dir(object) – display all file object functions

Open file

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt')

>>> dir(infile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
 '__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
 'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write',
 'writelines', 'xreadlines']
```

- Default mode is read
- Default buffering is enable
- "r" – is used to make file path raw
- Dir(object) – display all file object functions

Open file and read data

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
```

```
>>> help(infile.read)
```

Help on built-in function read:

```
read(...)
```

read([size]) -> read at most size bytes, returned as a string.

If the size argument is negative or omitted, read until EOF is reached.

Notice that when in non-blocking mode, less data than what was requested may be returned, even if no size parameter was given.

```
>>> print infile.read()
```

This is line number 1 in the file

This is line number 2 in the file

This is line number 3 in the file

File Pointer operations

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
```

```
>>> print infile.read(5)
```

This

```
>>> print infile.read(5)
```

is li

```
>>> print infile.read()
```

ne number 1 in the file

This is line number 2 in the file

This is line number 3 in the file

File Pointer operations with tell

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')

>>> print infile.tell()
0
>>> print infile.read(5)
This
>>> print infile.tell()
5
>>> print infile.read()
is line number 1 in the file
This is line number 2 in the file
This is line number 3 in the file
>>> print infile.tell()
103
```

File Pointer operations with seek

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
```

```
>>> print infile.read(5)
```

This

```
>>> help(infile.seek)
```

Help on built-in function seek:

seek(...)

seek(offset[, whence]) -> None. Move to new file position.

Argument offset is a byte count. Optional argument whence defaults to 0 (offset from start of file, offset should be ≥ 0); other values are 1 (move relative to current position, positive or negative), and 2 (move relative to end of file, usually negative, although many platforms allow seeking beyond the end of a file). If the file is opened in text mode, only offsets returned by tell() are legal. Use of other offsets causes undefined behavior.

Note that not all file objects are seekable.

readline and readlines functions

```
>>> help(infile.readline)  
readline(...)  
    readline([size]) -> next line from the file, as a string.
```

Retain newline. A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then).
Return an empty string at EOF.

```
>>> help(infile.readlines)  
  
readlines(...)  
    readlines([size]) -> list of strings, each a line from the file.
```

Call readline() repeatedly and return a list of the lines so read.
The optional size argument, if given, is an approximate bound on the total number of bytes in the lines returned.

readline and readlines functions

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')

>>> print infile.readline()
This is line number 1 in the file

>>> print infile.readlines()
['This is line number 2 in the file\n', 'This is line number 3 in the file']

>>> print infile.tell()
103

>>> infile.seek(0,0)

>>> print infile.readlines()
['This is line number 1 in the file\n', 'This is line number 2 in the file\n', 'This is line
number 3 in the file']
```

Write file

```
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.closed
False
>>> dir(outfile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
 '__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
 'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate',
 'write', 'writelines', 'xreadlines']
```

Data descriptors

closed

True if the file is closed

encoding

file encoding

errors

Unicode error handler

mode

file mode ('r', 'U', 'w', 'a', possibly with 'b' or '+' added)

name

file name

newlines

end-of-line convention used in this file

write function

```
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.write('This is line number 1 in the file\n')
>>> # Content is not yet written in the file, we need to flush or close object
```

```
>>> help(outfile.write)
```

Help on built-in function write:

write(...)

 write(str) -> None. Write string str to file.

Note that due to buffering, flush() or close() may be needed before the file on disk reflects the data written.

flush and close function

```
>>> help(outfile.flush)
```

Help on built-in function flush:

flush(...)

flush() -> None. Flush the internal I/O buffer.

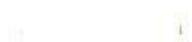
```
>>> help(outfile.close)
```

Help on built-in function close:

close(...)

close() -> None or (perhaps) an integer. Close the file.

Sets data attribute .closed to True. A closed file cannot be used for further I/O operations. close() may be called more than once without error. Some kinds of file objects (for example, opened by popen()) may return an exit status upon closing.



writelines function

```
>>> help(outfile.writelines)
```

Help on built-in function writelines:

```
writelines(...)
```

writelines(sequence_of_strings) -> None. Write the strings to the file.

Note that newlines are not added. The sequence can be any iterable object producing strings. This is equivalent to calling write() for each string.

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
>>> outfile = open(r'C:\Users\jatin\Desktop\Ethans\test1.txt', 'w')
>>> outfile.write(infile.readlines())
>>> outfile.close()
```

This module provides a portable way of using operating system dependent functionality

```
>>> import os
>>> dir(os)
['abort', 'access', 'altsep', 'chdir', 'chmod', 'close', 'closerange', 'curdir', 'defpath', 'devnull',
'dup', 'dup2', 'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve',
'execvp', 'execvpe', 'extsep', 'fdopen', 'fstat', 'fsync', 'getcwd', 'getcwdu', 'getenv', 'getpid',
'isatty', 'kill', 'linesep', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir',
'path', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'remove',
'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'spawnl', 'spawnle', 'spawnv', 'spawnve',
'startfile', 'stat', 'stat_float_times', 'stat_result', 'statvfs_result', 'strerror', 'sys', 'system',
'tempnam', 'times', 'tmpfile', 'tmpnam', 'umask', 'unlink', 'unsetenv', 'urandom', 'utime',
'waitpid', 'walk', 'write']
```

<https://docs.python.org/2/library/os.html>

os module functions help

```
>>> help(os.getenv)
```

Help on function getenv in module os:

```
getenv(key, default=None)
```

Get an environment variable, return None if it doesn't exist.

The optional second argument can specify an alternate default.

os module functions

```
>>> os.getenv('OS')
'Windows_NT'

>>> os.getcwd()
'C:\\Python27'

>>> os.getcwdu()
u'C:\\Python27'

>>> os.getpid()
10124

>>> os.chdir('..')
>>> os.getcwd()
'C:\\'

>>> os.listdir('.')
['Anaconda2', 'Ethans', 'Perl64', 'Program Files', 'Program Files (x86)', 'Python27', 'sqlite',
 'swapfile.sys', 'ubuntu', 'Users', 'Windows', 'Windows.old', 'wubildr', 'wubildr.mbr']
```

os module functions

```
>>> os.chdir('C:\\Users\\jatin\\Desktop\\Ethans')

>>> os.getcwd()
'C:\\\\Users\\\\jatin\\\\Desktop\\\\Ethans'

>>> os.mkdir('DemoDir')

>>> os.remove('file.txt')

>>> os.stat('test.txt')
nt.stat_result(st_mode=33206, st_ino=0L, st_dev=0, st_nlink=0, st_uid=0, st_gid=0,
st_size=103L, st_atime=1470715035L, st_mtime=1470715057L, st_ctime=1470715035L)

>>> os.stat('test.txt')[6]
103L

>>> os.system('cls')
>>> file = os.popen('dir')
>>> file.read() # return all data of dir command
```

os.path module functions

This module implements some useful functions on pathnames. To read or write files see `open()`, and for accessing the filesystem see the `os` module.

```
>>> dir(os.path)
['__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '_abspath_split',
'_getfullpathname', 'abspath', 'altsep', 'basename', 'commonprefix', 'curdir', 'defpath',
'devnull', 'dirname', 'exists', 'expanduser', 'expandvars', 'extsep', 'genericpath', 'getatime',
'getctime', 'getmtime', 'getsize', 'isabs', '.isdir', '.isfile', 'islink', 'ismount', 'join', 'lexists',
'normcase', 'normpath', 'os', 'pardir', 'pathsep', 'realpath', 'relpath', 'sep', 'split', 'splitdrive',
'splitext', 'splitunc', 'stat', 'supports_unicode_filenames', 'sys', 'walk', 'warnings']
```

os.path module functions

```
>>> os.path.isdir('DemoDir')
True
>>> os.path.isfile('test.txt')
True
>>> os.path.getsize('test.txt')
103L
>>> os.path.exists('blabla')
False
>>> os.path.basename(r'C:\Users\jatin\Desktop\Ethans\test.txt')
'test.txt'
>>> os.path.getmtime('test.txt')
1470715057.379443
```

Traverse a Directory Tree in Python

```
import os
for dirName, subdirList, fileList in os.walk('.'):
    print 'Found directory: %s' % dirName
    for fname in fileList:
        print '\t%s' % fname
```

""

dirName: The next directory it found.

subdirList: A list of sub-directories in the current directory.

fileList: A list of files in the current directory.

""

Python Modules and Packages

- Python inbuilt Modules
- os, sys, datetime, time, random, zip modules
- Create Python UDM – User Defined Modules
- Define PYTHONPATH
- Create Python Packages
- init File for package initialization

The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

More info: <https://docs.python.org/2/library/>

Common Modules

os — Miscellaneous operating system interfaces

os.path - Common pathname manipulations

pickle — Python object serialization

getopt — C-style parser for command line options

logging — Logging facility for Python

subprocess — Subprocess management

socket — Low-level networking interface

timeit — Measure execution time of small code snippets

datetime — Basic date and time types

math — Mathematical functions

shutil — High-level file operations

zipfile — Work with ZIP archives

sqlite3 — DB-API 2.0 interface for SQLite databases

re — Regular expression operations

smtplib — SMTP protocol client

smtpd — SMTP Server

telnetlib — Telnet client

More info: <https://docs.python.org/2/library/>



```
>>> import datetime

>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '__doc__', '__name__', '__package__', 'date', 'datetime',
'datetime_CAPI', 'time', 'timedelta', 'tzinfo']

>>> dir(datetime.datetime)
['__add__', '__class__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__',
'__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__', '__rsub__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', 'astimezone',
'combine', 'ctime', 'date', 'day', 'dst', 'fromordinal', 'fromtimestamp', 'hour',
'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min', 'minute', 'month',
'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timetuple',
'timetz', 'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow',
'utcoffset', 'utctimetuple', 'weekday', 'year']
```

Construct date

```
>>> t = datetime.time(1, 2, 3) # Construct a time
>>> t.hour # returns 1
>>> t.minute # returns 2

>>> d = datetime.datetime(2015, 6, 6)
>>> d.year # return 2015
```

Date Arithmetic

```
>>> today = datetime.date.today()          # Print current server date
2016-08-11
>>> one_day = datetime.timedelta(days=1)
>>> print today + one_day
```

Compare dates

```
>>> tomorrow = today + one_day
>>> tomorrow > today
True
```

datetime formatting

```
import datetime

format = "%a %b %d %H:%M:%S %Y"

today = datetime.datetime.today()
print 'ISO    :', today

s = today.strftime(format)
print 'strftime:', s

d = datetime.datetime.strptime(s, format)
print 'strptime:', d.strftime(format)
```

Construct time

```
>>> import time
>>> print 'The time is:', time.time()
The time is: 1470889543.66
>>> print 'The time is:', time.ctime()
The time is: Thu Aug 11 09:56:05 2016

>>> print 'gmtime :', time.gmtime()
gmtime :time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=4,
tm_min=29, tm_sec=41, tm_wday=3, tm_yday=224, tm_isdst=0)
>>> print 'localtime:', time.localtime()
localtime: time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=9,
tm_min=59, tm_sec=52, tm_wday=3, tm_yday=224, tm_isdst=0)
```

Arithmetic on time

```
>>> print '15 minutes later: ', time.ctime(time.time() + 15)
15 minutes later: Thu Aug 11 09:58:53 2016
```

Construct time

```
>>> import time
>>> print 'The time is:', time.time()
The time is: 1470889543.66
>>> print 'The time is:', time.ctime()
The time is: Thu Aug 11 09:56:05 2016

>>> print 'gmtime :', time.gmtime()
gmtime :time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=4,
tm_min=29, tm_sec=41, tm_wday=3, tm_yday=224, tm_isdst=0)
>>> print 'localtime:', time.localtime()
localtime: time.struct_time(tm_year=2016, tm_mon=8, tm_mday=11, tm_hour=9,
tm_min=59, tm_sec=52, tm_wday=3, tm_yday=224, tm_isdst=0)
```

Arithmetic on time

```
>>> print '15 minutes later: ', time.ctime(time.time() + 15)
15 minutes later: Thu Aug 11 09:58:53 2016
```

time formatting

```
import time

now = time.ctime()
print now

parsed = time.strptime(now)
print "%d-%d-%d" %(parsed[0],parsed[1], parsed[2])

print time.strftime("%a %b %d %H:%M:%S %Y", parsed)
```

Thu Aug 11 10:02:38 2016

2016-8-11

Thu Aug 11 10:02:38 2016

random – Pseudorandom number generators

```
>>> import random  
>>> random.random()  
0.9047586792455821  
  
>>> print random.randint(1, 100)  
60
```

```
import random  
  
with open('words.txt', 'r') as f:  
    words = f.readlines()  
    words = [ w.rstrip() for w in words ]  
  
for w in random.sample(words, 5):  
    print w
```

zip module

```
zf = zipfile.ZipFile('zipfile_write.zip')
for filename in [ 'LICENSE.txt']:
    data = zf.read(filename)
    print filename, ':'
    print repr(data)
```

```
import zipfile

print 'creating archive'
zf = zipfile.ZipFile('zipfile_write.zip', mode='w')
print 'adding README.txt'
zf.write('LICENSE.txt')
zf.close()

print zipfile.is_zipfile('zipfile_write.zip')
```

- You may want to split your program into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.
- A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.
- A module can contain executable statements as well as function definitions

Syntax:

```
import module1[, module2[,... moduleN]]
```

There are (at least) three kinds of modules in Python:

- modules written in Python (.py);
- modules written in C and dynamically loaded (.dll, .pyd, .so, .sl, etc);
- modules written in C and linked with the interpreter.

To find the location of the file:

- `>>> import math and >>> math.__file__` (Error as here no file is available)
- `>>> import os, os.__file__ # 'C:\\Python27\\lib\\os.pyc'`
- `>>> import random, >>> random.__file__`, `usr/lib/python2.4/random.pyc'`

For example:

```
# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

Do following:

- Create a simple script/program with .py extension.
- Import that program to your namespace and called its object.
- Write a program to create Fibonacci module and print the first 20 Fibonacci numbers.

Python search the PYTHONPATH for the module folder to get them imported to the namespace. We will discuss in details in next slides.

For Now if we want to import any of the module. We append the sys.path by the path we required.

For Example:

```
>>> sys.path.append('C:/Users/Jatin/Desktop/Python Scripts')
```

```
>>> import studentMarks2
```

Jatin has average marks: 75

- Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module.
- You can also import the specific methods of modules as well.

Syntax:

`from module import function1, function2`

Now you can call that function directly:

`function1(500)`

`from module import *`

Import all methods and variable except all stating with _ (Avoid it)

- When a Python file is run directly, the special variable "`__name__`" is set to "`__main__`". Therefore, it's common to have the boilerplate if `__name__ == ...` shown above to call a `main()` function when the module is run directly, but not when the module is imported by some other module.

```
#!/usr/bin/python

# import modules used here -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

    # Standard boilerplate to call the main() function to begin
    # the program.
if __name__ == '__main__':
    main()
```

- The interpreter first searches for a built-in module with that name.
- The directory containing the input script (or the current directory).
- PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH).
- The installation-dependent default.

These search paths would be in a list of directories given by the variable sys.path

```
set PYTHONPATH=C:\\\\Users\\\\Sony\\\\ Desktop\\\\Python Batch 29
Oct;C:\\Users\\Sony\\Desktop\\Python Batch 29 Oct\\Day 3
```

- What is package?
- Packages are a way of structuring Python's module namespace.

How it works?

- Suppose you want to design a collection of modules (a “package”) for the uniform handling of employee files and department data.
- When importing the package, Python searches through the directories on `sys.path` looking for the package subdirectory.
- The `__init__.py` files are required to make Python treat the directories as containing packages, this is done to prevent directories with a common name.

__init__.py

__init__.py is the mandatory file available in the directory to make it package

```
root\
    system1\
        __init__.py
        utilities.py
        main.py
        other.py
    system2\
        __init__.py
        utilities.py
        main.py
        other.py
    system3\
        __init__.py
        myfile.py
# Here or elsewhere
# Your new code here
```

import everything

- What happens when the user writes
`from root.system1 import *`?

- The import statement uses the following convention: if a package's `__init__.py` code defines a list named `__all__`, it is taken to be the list of module names that should be imported when `from package import *` is encountered.

Usually `__init__.py` contains the following code:

```
__all__ = ["echo", "surround", "reverse"]
```

Some imp functions

➤ filter(function, sequence)

returns a sequence consisting of those items from the sequence for which function(item) is true.

If sequence is a string or tuple, the result will be of the same type; otherwise, it is always a list. For example, to compute primes up to 25:

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
```

...

```
>>> filter(f, range(2, 25))
```

```
[5, 7, 11, 13, 17, 19, 23]
```

➤ map(function, sequence)

calls function(item) for each of the sequence's items and returns a list of the return values. For example, to compute some cubes:

```
>>> def cube(x): return x*x*x
```

```
>>> map(cube, range(1, 11))
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Exception Handling

- Python Exceptions Handling
- What is Exception?
- Handling various exceptions using try....except...else
- Try-finally clause
- Argument of an Exception and create self exception class
- Python Standard Exceptions
- Raising an exceptions, UD Exceptions

OOPS Python

- Object oriented features
- Understand real world examples on OOP
- Implement Object oriented with Python
- Creating Classes and Objects, Destroying Objects
- Accessing attributes, Built-In Class Attributes
- Inheritance and Polymorphism
- Overriding Methods, Data Hiding
- Overloading Operators

Common Errors

- Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python.
- Runtime errors, are the errors/exception the program received because of incorrect logic or unexpected scenarios in the programs.
- Most runtime exceptions are not handled by programs, so the programs terminate unexpectedly.

For Example:

```
>>> 20 * (10/0)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

```
>>> 7 + test*3
Traceback (most recent call last): File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
```

Exceptions

What is exception?

- An exception is a Python object that represents an error. Python script encounters a situation that it can't cope with, it raises an exception.
- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them.
 - 1) Exception Handling
 - 2) Assertion
- When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate and come out.
- There are number of built-in exception Python provides, some of them are listed in next slide.

Built-in Exceptions

```
+-- Exception
    +-- StopIteration
    +-- StandardError
    |   +-- BufferError
    |   +-- ArithmeticError
    |   |       +-- FloatingPointError
    |   |       +-- OverflowError
    |   |       +-- ZeroDivisionError
    |   +-- AssertionError
    |   +-- AttributeError
    |   +-- EnvironmentError
    |   |       +-- IOError
    |   |       +-- OSSError
    |   |           +-- WindowsError (Windows)
    |   |           +-- VMSError (VMS)
    |   +-- EOFError
    |   +-- ImportError
    |   +-- LookupError
    |   |       +-- IndexError
    |   |       +-- KeyError
    |   +-- MemoryError
    |   +-- NameError
    |   |       +-- UnboundLocalError
    |   +-- ReferenceError
    |   +-- RuntimeError
    |   |       +-- NotImplementedError
    |   +-- SyntaxError
    |   |       +-- IndentationError
    |   |       +-- TabError
```

Try-except

- If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

```
try:  
    You do your operations here;  
    .....  
except ExceptionI:  
    If there is ExceptionI, then execute this block.  
except ExceptionII:  
    If there is ExceptionII, then execute this block.  
    .....  
else:  
    If there is no exception then execute this block.
```

About Try-except

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection
- `exc_info` is the method in the `sys` module, provide the explanation of the exception

Else with try

try...else

- You can use an else block with a try block
- Else will execute when no exception is raised by try block.

Syntax

```
try:  
    piece of code will be written here  
    .....  
else:  
    This block will execute when there is no  
    exception  
    .....
```

Finally with try

try...finally

- You can use an finally block with a try block
- Finally will execute, in both the cases

Syntax

```
try:  
    piece of code will be written here  
    .....  
finally:  
    This block will execute when there is exception or no excp  
    .....
```

Raise exceptions

Raise Exception

You can also raise exceptions by using the raise statement.

Syntax

```
raise [Exception [, args]]
```

- Exception is the type of exception
- args is a value for the exception argument. It is an optional argument; if not supplied, the exception argument is None.
- args will catch while create an object of Exception class, shown in demo

Raise exceptions

- The raise statement allows the programmer to force a specified exception to occur. For example:

```
>>> raise NameError('HiThere')
```

```
Traceback (most recent call last): File "<stdin>", line 1, in ?
```

```
NameError: HiThere
```

```
>>> try: ...
```

```
    raise NameError('HiThere')
```

```
... except NameError: ...
```

```
    print 'An exception flew by!'
```

```
>>> try: ...
```

```
    raise Exception('spam', 'eggs') ...
```

```
except Exception as inst:
```

```
    print type(inst) # the exception instance
```

```
    print inst.args # arguments stored in .args
```

User Defined Excp

Programmers can also have their own exceptions by creating a new exception class Exceptions should typically be derived from the Exception class, either directly or indirectly.

For Example:

```
>>> class MyError(Exception):
```

```
...     def __init__(self, value):  
...         self.value = value
```

```
>>> try:
```

```
...     raise MyError(2*2)
```

```
except MyError as e:
```

```
    print 'My exception occurred, value:', e.value ...
```

```
My exception occurred, value: 4
```

→ Computer languages are called as object oriented programming language when they follow the object oriented principles. These principles are the pillars of object oriented programming.

Encapsulation:

It can also be called as Data hiding principle, where user do not need to bother about how the things

implemented rather than knowing how to use them. Encapsulation is the packaging of data and functions

together. It also allow selective hiding of attributes in an object to protect the code from accidental corruption.

Inheritance:

It is the principle of code re-usability, when the object or class inherit/borrow the properties from other class to

minimize the implementation of similar property in a class.

Polymorphism:

This principle refers to a programming language's ability to process objects differently depending on their data type or class.

What is object?

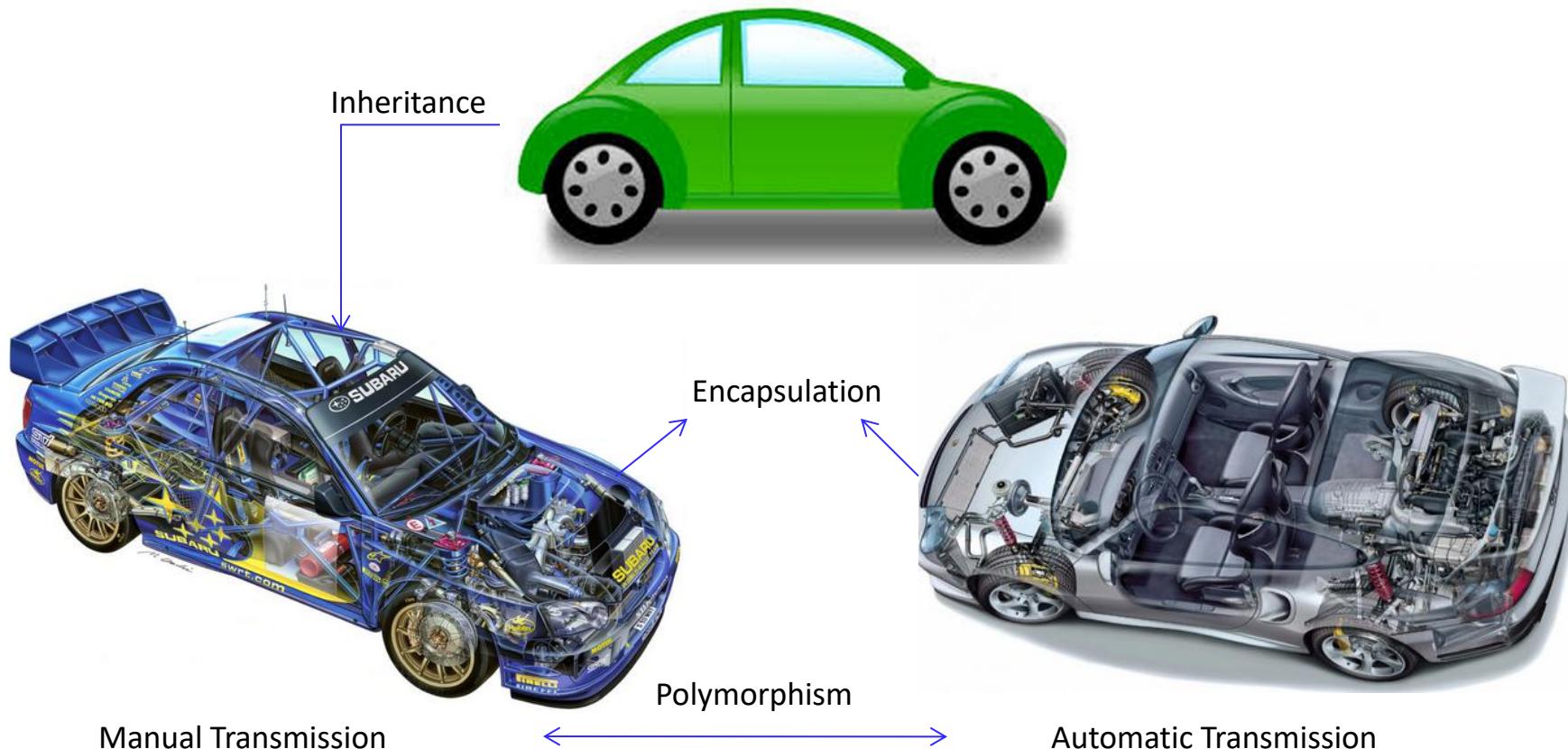
An object refers as an instance of the class, which is used to manipulate the attributes of the class.

An object can characterized as the real world object which has same characteristics, state and behavior. Cars have state (name, color, model) and behavior (turning, reverse moving, moving at particular speed). So any car model in the real world is the object of car class. As we can have many models of car, similar there can be many object of the car class.

What is class?

A class is the prototype or the blueprint from which objects are created. In the real world, you often find the many objects of the single class. Every class has the attributes and functions, like car class has the function of manual gear. Car will move back if the reverse gear start operating. These attributes are access with the object of the class.

OOPS Concept



The simplest form of python class definition looks like:

Syntax:

```
class ClassName:  
    <statement-1>  
    ... <statement-N>
```

To Note:

Similar as function definitions ([def](#) statements) class definition must be executed before they have any effect.

Python Class Object

Can be used for:

- 1) Initialization
- 2) Attribute reference.

Initialization:

x = MyClass()

It creates a new *instance* of the class (or new object) and assigns this object to the local variable x. Though it created an empty variable.

Attribute reference:

x.f()

x.FunctionName()

Class Example

Creating Class

```
class exampleClass:
```

```
... firstVar = 'This is the firstvar in the class'
```

```
... secondVar = 'This is the secondVar in the class'
```

```
def classMethod(self)
```

```
... print 'I am in the classMethod'
```

Creating Object

```
classObj = exampleClass()
```

```
classObj.firstVar
```

```
classObj.classMethod()
```

Getter and Setter

Creating Class

```
class exampleClass:  
...   firstVar = 'This is the firstvar in the class'  
      def setName(self, name)  
        ...   self.name=name  
      def getName(self):  
        ...   self.name
```

Creating Object

```
classObj = exampleClass()  
classObj.firstVar  
classObj. setName('Ethans')  
classObj. getName()
```

Class Constructor

Constructor is the special method in the class which perform certain actions when you create an object.

Creating Class with Constructor

```
class exampleClass:  
... firstVar = 'This is the firstvar in the class'  
    def __init__(self, name)  
        ... print "Let's say I am in initialization phase"  
        ... self.name=name
```

Creating Object

```
classObj1 = exampleClass('Jatin')  
classObj2 = exampleClass('Educum')
```

```
classObj1.name
```

Class and instance variable

```
class Dog:

    kind = 'canine'          # class variable shared by all instances

    def __init__(self, name):
        self.name = name    # instance variable unique to each instance

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                # shared by all dogs
'canine'
>>> e.kind                # shared by all dogs
'canine'
>>> d.name                # unique to d
'Fido'
>>> e.name                # unique to e
'Buddy'
```

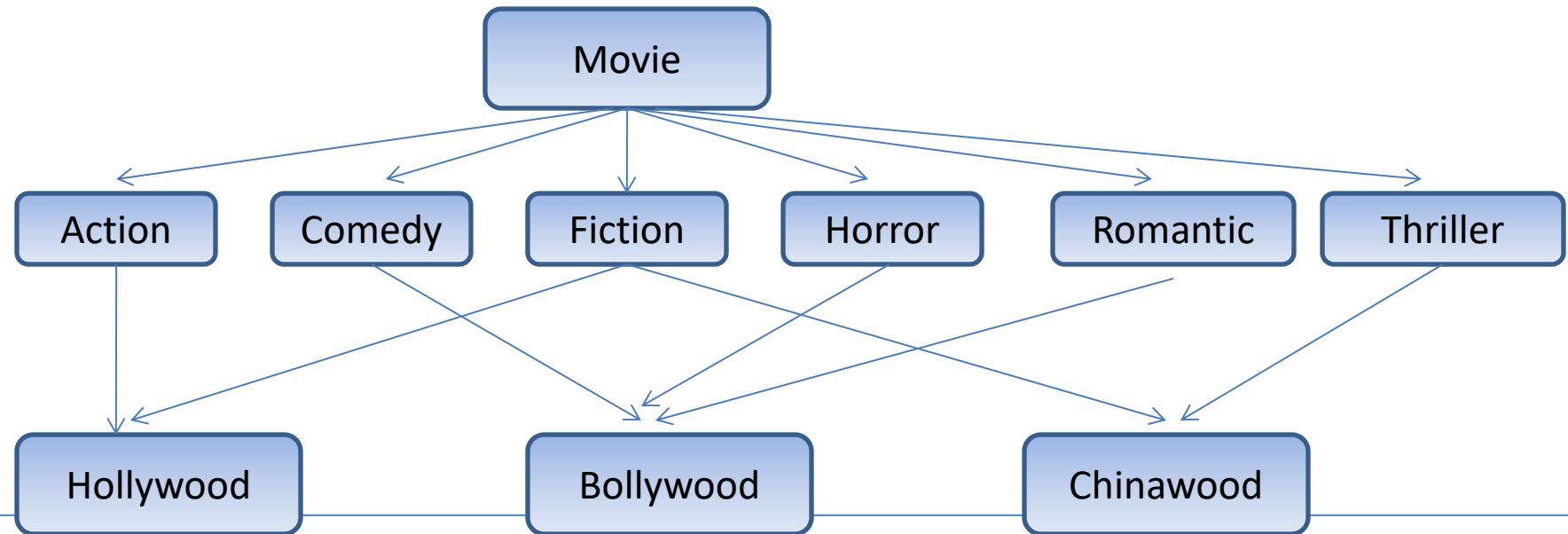
Code Snippet:

- ✓ Kind is class variable use by all object. But cannot be used inside the method.
- ✓ Name is instance variable.
- ✓ Avoid class variables as much as you can.

Inheritance

→ One of the principle of objected oriented language is inheritance.

→ The following diagram show how the inheritance works.



Inheritance

Syntax:

```
class childClass(parentClass):  
    <statement-1>  
    ... <statement-N>
```

Here child class is derived from the parent class. So all the parameters of parent class belongs to the child class. So if the child object try to access the parameter which is not available in child class it goes to parent class.

If the parameter is available at child class, child object access the child parameter. In this case parent object is overridden.

Inheritance ex

```
class parentClass:
```

```
...     firstVar = 'This is my firstVar in the parent Class'  
...     secondVar = 'This is my secondVar in the parent Class'
```

```
class childClass(parentClass):
```

```
...     pass
```

```
Pobj= parentClass()
```

```
Cobj= childClass()
```

```
Pobj.firstVar
```

```
Cobj.firstVar
```

Inheritance with multiple classes ex

```
class parentClass1:
```

```
...     firstVar = 'This is my firstVar in the parent Class'  
...     secondVar = 'This is my secondVar in the parent Class'
```

```
class parentClass2:
```

```
...     thirdVar = 'This is my thirdVar in the parent Class'  
...     secondVar = 'This is my secondVar in the parent Class2'
```

```
class childClass(parentClass1, parentClass2):
```

```
...     pass
```

```
Cobj= childClass()
```

```
Cobj.firstVar
```

```
Cobj.secondVar
```

New Style Classes

```
class parentClass1(object):
...     def func(self):
        print 'This is my func function in the parent Class1'
```

```
class parentClass2(object):
...     def func(self):
        print 'This is my func function in the parent Class2'
```

```
class childClass(parentClass1, parentClass2):
...     def func(self):
        super(childClass, self).func()
```

```
Cobj= childClass()
Cobj.func()
```

Class Functions

- The getattr(obj, name[, default]) : to access the attribute of object
- The hasattr(obj, name) : to check if an attribute exists or not.
- The setattr(obj, name, value) : to set an attribute. If attribute does not exist, then it would be created.
- The delattr(obj, name) : to delete an attribute.

For example:

- hasattr(emp1, 'age') # Returns true if 'age' attribute exists
- getattr(emp1, 'age') # Returns value of 'age' attribute
- setattr(emp1, 'age', 8) # Set attribute 'age' at 8
- delattr(empl, 'age') # Delete attribute 'age'

Function Overloading

```
class overloading():

    def __init__(self, var1, var2):
        self.var1 = var1
        self.var2 = var2
        print "Value of var1 and var2 is %d and %d" %(var1, var2)

    def __str__():
        return 'Returning a string from it'

    def __len__():
        return self.var1

    def __add__(self, var):
        self.var3 = self.var1 + var
        return 'Sum of var1 and var2 is: %r' %self.var3

obj = overloading(10, 20)
print obj
print len(obj)
print obj + 20
```

Function Overloading

```
class overloading():

    def __init__(self, var1, var2):
        self.var1 = var1
        self.var2 = var2
        print "Value of var1 and var2 is %d and %d" %(var1, var2)

    def __str__():
        return 'Returning a string from it'

    def __len__():
        return self.var1

    def __add__(self, var):
        self.var3 = self.var1 + var
        return 'Sum of var1 and var2 is: %r' %self.var3

obj = overloading(10, 20)
print obj
print len(obj)
print obj + 20
```

Debugging, Framework & Regular expression

- Debug Python programs using pdb debugger
- Pycharm Debugger
- Assert statement for debugging
- Testing with Python using UnitTest Framework
- What are regular expressions?
- The match and search Function
- Compile and matching
- Matching vs searching
- Search and Replace feature using RE
- Extended Regular Expressions
- Wildcard characters and work with them

Python Debugger

- Pdb is the interactive source code debugger for Python
- It is by default available along with the Python interpreter.
- It is very common for IDLE or command line user who do not use any specific debugger tool and Python IDE.
- Pdb is the name of the module and pdb.set_trace is the name of the function which enable debugging mode.

```
import sys
import os
import pdb

pdb.set_trace()

def sumOfTwoNumbers(a, b):
    c = a + b
    return c

a = int(sys.argv[1])
b = int(sys.argv[2])

sum = sumOfTwoNumbers(a, b)
print sum
```

Generic commands

Debugger Commands	Description
h	Help
n	Execute the next statement
Enter	Repeating the last debugging command
q	Quitting debugger
p	Print the variables
c	Turning off the (pdb) prompt
l	Seeing where you are
w	Print a stack trace, with the most recent frame at the bottom
d	Move the current frame one level down in the stack trace
u	Move the current frame one level up in the stack trace

- n (Execute the next statement with ‘n’) - Lower case n key is used to execute the next statement (can be said line by line or complete block), Ultimately it will come to the end of the program and return you the normal prompt.
- Enter – Execute the last statement (Repeating the last command)
- q (Quit from the debugger) - Lower case q is used to abruptly quitting the debugger that's why you see an exception while execution of the program.
- p (Printing the value of the variable during debugging) - During the execution user can print the variable value, and NameError will be raised if the variable is not defined.
- c (continue the execution the program with c command)

- l (List the location of the program) - “l” shows you, on the screen, the general area of your program’s source code that you are executing. By default, it lists 11 (eleven) lines of code
- s (step into functions) - With the help command you will step inside the function in python
- r (return from the function) - Same as c command but for functions
- b (set the breakpoint) - b number will set the breakpoint and b directly return all the breakpoint set in the program.

- Assign value to the variable
- !Var = 10

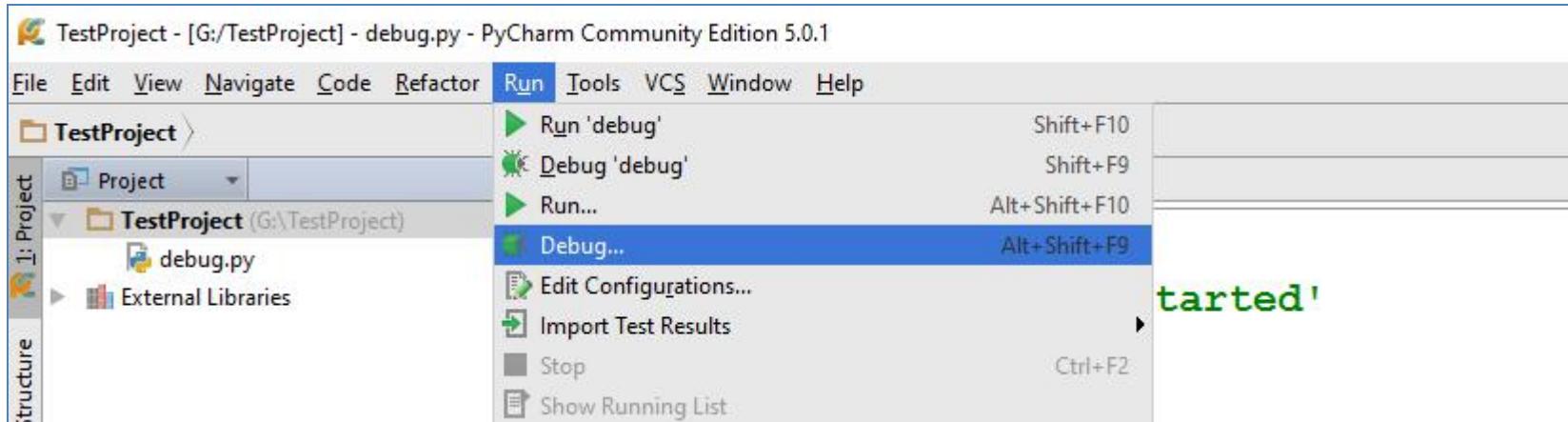
PDB Demonstration

PyCharm is an Integrated Development Environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and supports web development with Django. PyCharm is developed by the Czech company JetBrains.

It is cross-platform working on Windows, Mac OS X and Linux. PyCharm has a Professional Edition, released under a proprietary license and a Community Edition released under the Apache License. PyCharm Community Edition is less extensive than the Professional Edition.

Source - Wikipedia

Pycharm Debugger



Pycharm Debugger Demonstration



Pycharm Debugger Demonstration

- Unittest Framework is xUnit style framework for Python.
- The unittest module used to be called PyUnit.
- The framework implemented by unittest supports fixtures, test suites, and a test runner to enable automated testing for your code.
- The standard workflow is:
 1. You define your own class derived from unittest.TestCase.
 2. Then you fill it with functions that start with ‘test_’.
 3. You run the tests by placing unittest.main() in your file, usually at the bottom.

Automating Unit test case - 1

```
import unittest

def checkArguments(a,b):
    return True if a == b else False

class EqualityTestCases(unittest.TestCase):

    def test_check_if_4_equals_4(self):
        self.assertTrue(checkArguments(4,4))

if __name__ == '__main__':
    unittest.main()
```

Automating Unit test case - 2

```
import unittest

def checkArguments(a,b):
    return True if a == b else False

class EqualityTestCases(unittest.TestCase):

    def setUp(self):
        print 'This method will call before every test case called!'

    def test_check_if_4_equals_4(self):
        self.assertTrue(checkArguments(4,4))

    def test_check_string_multiplication(self):
        self.assertEqual('a' * 3, 'aaa')

if __name__ == '__main__':
    unittest.main()
```

Execute Test Cases

```
C:\Users\jatin\Dropbox\Edureka GE Batch\Day6>python testcases2.py -v  
test_check_if_4_equals_4 (__main__.EqualityTestCases) ... This method will call  
before every test case called!
```

```
ok
```

```
test_check_string_multiplication (__main__.EqualityTestCases) ... This method  
will call before every test case called!
```

```
ok
```

```
Ran 2 tests in 0.014s
```

```
OK
```

Execute Specific Test case

```
C:\Users\jatin\Dropbox\Edureka GE Batch\Day6>python  
-m unittest testcases2.EqualityTestCases.test_check_  
string_multiplication
```

```
This method will call before every test case called!
```

```
.
```

```
-
```

```
-
```

```
Ran 1 test in 0.003s
```

```
OK
```

```
?
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

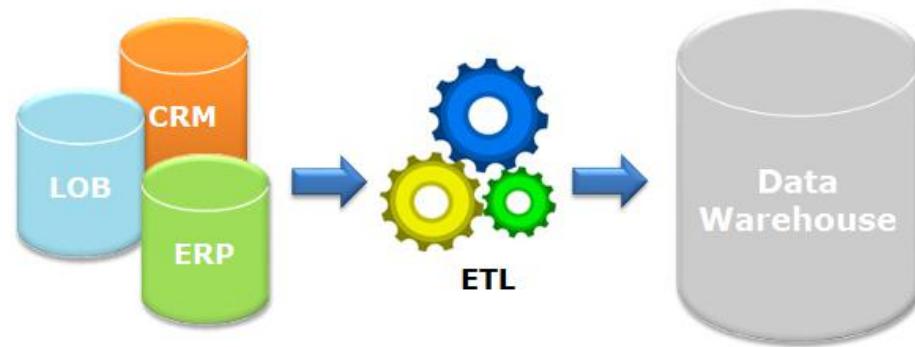
UNIT Test Framework Demo

What is Regular Expression?

- Regular expression is a set of characters together form the search pattern.
- Main use of regular expression is to matches pattern in any string forms.
- The other use of regular expression to provide ‘find and replace feature’ in the programming languages
- Many language provide regular expression capabilities, some language have it inbuilt and other are having regular expression libraries.
- Regular expression is also known by regex or regexp.
- Regular expression forms the generic pattern for the string matching with the help of pre-defined wildcard characters.
- In Python, regular expression is supported by **re module**, which comes along with the installation of Python interpreter.

Traditional Uses of Regular expression

- Online Ad Targeting
- Data filtrations
- Text Processing
- Big Data analysis
- Data science
- ETL processing
- Internet Search
- Web Scraping
- Data handling
- Data substitutions
- And much More...



Match Function

- Match function is available in re module.
- It attempts to match an RE pattern to a string with optional flags at the beginning of string.
- The re.match function returns a match object on success, None object on failure.
- Help on match function:

```
>>> import re  
>>> help(re.match)
```

Parameter	Description
pattern	The regex/pattern which need to match
string	This is the string, where re need to be matches
flags	Modifiers or Flags to support more functionality to re

Match Function - Ex

```
import re

line = "this is the string contain sentence"
matchObj = re.match("This is the string Match", line, re.I)

if matchObj:
    print "\n matchObj.group() : \n", matchObj.group()
    print "\n matchObj.group(1) : \n", matchObj.group(1)
else:
    print "No match!!"
```

Search Function

- It scans the complete string and for the first occurrence of a given pattern within a string
- The `re.search` function returns a match object on success, `None` on failure.

- Help on match function:

```
>>> import re  
>>> help(re.match)
```

Parameter	Description
pattern	The regex/pattern which need to match
string	This is the string, where we need to find matches
flags	Modifiers or Flags to support more functionality to <code>re</code>

Search Function

```
import re

line = "this is the string contain sentence"
matchObj = re.search("This is the string Match", line, re.I)

if matchObj:
    print "\n matchObj.group() : \n", matchObj.group()
    print "\n matchObj.group(1) : \n", matchObj.group(1)
else:
    print "No match!!"
```

Findall Function

- It scan the complete string and get all occurrences of a given pattern within a string
- The re.findall function returns a list of all matches.

- Help on match function:

```
>>> import re
```

```
>>> help(re.findall)
```

Parameter	Description
pattern	The regex/pattern which need to match
string	This is the string, where re need to be matches
flags	Modifiers or Flags to support more functionality to re

Findall Function ex:

```
import re

line = "this is the string contain sentence"
allMatches = re.findall("This", line, re.I)

if len(allMatches):
    print "\n allMatches : \n", allMatches
else:
    print "No match!!"
```

The First Wildcard

Wildcards (are also called as quantifiers) are the operator symbols which have specific meaning inside regular expression.

For example: . (Dot or period) matches any character, digit, alphanumeric character except newline character (\n).

```
def refind(pat, str):
    match = re.search(par, str):
        if match:
            print match.group()
        else:
            print 'Pattern not found'
```

```
>>> refind('a.b', 'this is acb string')
acb
>>> refind('a.b', 'this is atb string')
atb
>>> refind('a.b', 'this is a b string')
a b
>>> refind('a.b', 'this is a\b string')
a      b
```

Match Operator Itself

In many cases, user may wants to match the operator symbol itself in the regular expression. We can suppress the wild cards and special characters itself by backslash (\)

Ex:

```
>>> refind('10.', 'this is a number 101 and ip address 10.')
101
>>> refind('10\.', 'this is a number 101 and ip address 10.')
10.
```

Other Wildcards

These wildcard characters do not matches themselves. Until and unless they suppressed by backslash.

Following are the other wildcards:

Wildcard	Meaning
*	matches Zero or more occurrence of previous character/s
+	matches One or more occurrence of previous character/s
?	matches Zero or One occurrence of previous character/s

Other Wildcards

REGEX	Matches
AbC*	It matches A followed by b followed by either Zero or more occurrence of C i.e. Ab, AbC, AbCCCC. AbCCCCCCCCCC
AbC+	It matches A followed by b followed by minimum one or more occurrence of C i.e. Abc, AbCCCCCCC, AbCCC
AbC?	It matches A followed by b followed by one or Zero occurrence of C. i.e. Ab, AbC
Ab(cd)*	It matches A followed by b followed by either Zero or more occurrence of cd i.e. Ab, Abcd, Abcdcd
Ab(cd)+	It matches A followed by b followed by minimum one or more occurrence of cd i.e. Abcd, Abcdcd
Ab(cd)?	It matches A followed by b followed by either one or zero occurrence of cd i.e. Abcd, Ab

Combine Wildcards

REGEX	Matches
Ab+C*	It matches A followed by minimum one or more occurrence of b followed by either Zero or more occurrence of C. i.e. Ab, Abc, Abbccc. Abbcccccccccc
A.C+	It matches A followed by any character followed by minimum one or more occurrence of C i.e. AZc, Azccc. AEcccccccccc
..C?	It matches any two characters followed by b followed by one or Zero occurrence of C. i.e. Ab, AbC
<.*>	It matches anything inside tags <> i.e. <HTML>, <TAGS>
\(.+\)	It matches minimum one character inside brackets cd i.e. (Abcd), (a)
ab+c?	It matches a followed by one or more b followed by zero or one c. i.e. "abbbbc" or "abc", but not "ac"

Character Class

REGEX	Matches
[abc]	It matches any string which has either 'a' or 'b' or 'c'
[abcdefghijklmnopqrstuvwxyz]	It matches any string which has either 'a' or 'b' or 'c' or so on till 'z'
[a-zA-Z]	It matches any string which has either 'a' or 'b' or 'c' or so on till 'z'
[0-9]	It matches any string which has 0 or 1 or 2 or 3 till 9
[a-zA-Z0-9]	It matches any string which has characters from a-z and A-Z and 0-9
[a-zA-Z_]	It matches any string which has characters from a-z or _ (underscore)

Character Class

REGEX	Matches
[^abc]	It matches any string which has neither 'a' nor 'b' nor 'c'
[^abcdefghijklmnopqrstuvwxyz]	It matches any string which has neither 'a' or 'b' or 'c' or so on till 'z'
[^a-z]	It matches the string which has neither 'a' or 'b' or 'c' or so on till 'z'
[^aeiou]	It matches the string which has no vowels.
[IL][^abc]	It matches the string has 'I' or 'L' should not followed by 'a' nor 'b' nor 'c'
[^a-zA-Z_]	It matches the string doesn't have a-z or _ (underscore)

Character Class

REGEX	Matches
[aA][0-9]+	It matches any string which has 'a' or 'A' followed by any number and occurrence can any number of times.
A+.[.?]	It matches any string which has 'A' any number of times followed by any character followed by either '.' or '?
a[bc]	It matches any string which has 'a' followed by either 'b' or 'c'
A[abc]?	It matches the string which has 'A' followed by zero or one occurrence of either 'a' or 'b' or 'c'
[a-z_.]\@	It matches the string has 'a' to 'z' or '_' or '.' followed by '@'

Shortcuts

Shortcut	Say	Meaning
\s	Any space, tab or new line characters	[\t\n]
\S	Other than space, tab or newline character	[^\t\n]
\d	Any digit	[0-9]
\D	Other than digit	[^0-9]
\w	Digits, characters or _ (underscore)	[a-zA-Z0-9_]
\W	Other than digit, character or _	[^a-zA-Z0-9_]

Database Interface

- Creating a Database with SQLite 3
- CRUD Operations,
- Creating a Database Object.
- Python MySQL Database Access
- DML and DDL Operations with Databases
- Performing Transactions
- Handling Database Errors
- Disconnecting Database

Python Database modules supports a wide range of database servers:

- IBM DB2
- Firebird
- Informix
- Ingres
- MySQL
- Oracle
- PostgreSQL
- SAP DB
- Microsoft SQL Server
- Microsoft Access
- Sybase
- Teradata
- IBM Netezza

Standard for RDBMS Databases

Python DB API/drivers/modules provides conventional approach while working with databases using Python standard structures and syntax.

This includes the following:

- Importing the Python-DB module.
- Acquire a connection with the database.
- Create a DB cursor
- Issuing SQL statements and stored procedures.
- Closing the connection



- SQLite is a relational database management system contained in a C programming library.
- SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers.
- It is fast, reliable and the entire DB is stored in single disk file.
- Python module sqlite3 intended to work with SQL Lite3.

Source: <https://en.wikipedia.org/wiki/SQLite>

Acquire DB connection

- Python sqlite3 module will be used to connect SQLite3 Database
- It is default available along with the Python installable.
- connect, close are the functions in below example

```
>>> # Importing sqlite3 DB
>>> import sqlite3
>>> # Creating a DB in RAM
>>> dbr = sqlite3.connect(':memory:')
>>> # Creating/Using a DB in/from a file
>>> db = sqlite3.connect('ethans')
>>> # Closing a DB connection
>>> db.close()
```

Create table

```
import sqlite3 # Importing module
db = sqlite3.connect('ethans')
dbCursor = db.cursor() # DB cursor
dbCursor.execute(""" CREATE TABLE
    employee(id int, name text,dept text,
    salary int) """)
db.close()
```

```
>>> help(dbCursor.execute)
execute(...)
    Executes a SQL statement.
```

```
>>> dir(dbCursor)
['arraysize', 'close', 'connection', 'description', 'execute', 'executemany',
'executescript', 'fetchall', 'fetchmany', 'fetchone', 'lastrowid', 'next',
'row_factory', 'rowcount', 'setinputsizes', 'setoutputsize']
```

Insert Data - 1

```
''' Insert a record '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

dbCursor.execute(""" INSERT INTO employee
    values (1, 'ethans','training', '50000')
""")

db.commit()
db.close()
```

Insert Data - 2

```
''' Insert a record '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

dbCursor.execute(""" INSERT INTO employee
    values (?, ?, ?, ?)""",
    (1, 'ethans', 'training', 50000))

db.commit()
db.close()
```

Insert Data - 3

```
''' Insert a record '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

eid, name, dept, salary = 1, 'ethans', 'training', 50000

dbCursor.execute(""" INSERT INTO employee
    values (?, ?, ?, ?)""",
    (eid, name, dept, salary))

db.commit()
db.close()
```

Insert Data - 4

```
import sqlite3, sys
db = sqlite3.connect('Employee')
cursor = db.cursor()

id, name, dept, salary = 6, 'Chetan', 'IT', 10000
sql = '''INSERT INTO emp values (%d, '%s', '%s', %d)''' \
      %(id, name, dept, salary)

cursor.execute(sql)
db.commit()

cursor.execute('''Select * from emp''')
for record in cursor.fetchall():
    print record

db.close()
```

Insert Data - 5

```
''' Insert a record '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

eid, name, dept, salary = 1, 'ethans', 'training', 50000

dbCursor.execute(""" INSERT INTO employee
    values (:id, :name, :dept, :salary)""",
    {'name':name, 'id':eid, 'dept':dept, 'salary':salary})

db.commit()
db.close()
```

Insert Data - 6

```
''' Insert a record '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

eid, name, dept, salary = 1, 'ethans', 'training', 50000

dbCursor.executemany(""" INSERT INTO employee
    values (?,?,?,?)""",
    [(eid, name, dept, salary), (2, 'Steve', 'training', 60000)])

db.commit()
db.close()
```

Insert and Select Data

```
import sqlite3, sys
db = sqlite3.connect('Employee')
cursor = db.cursor()

id, name, dept, salary = 7, 'Santosh', 'IT', 10000
cursor.execute('''INSERT INTO emp values (:id, :name, :dept, :sal)'''
               {'sal':salary, 'name':name, 'dept':dept, 'id':id})
db.commit()

cursor.execute('''Select * from emp where dept = ?''', ('IT',))
for record in cursor.fetchall():
    print list(record)

db.close()
```

Insert using executemany and select

```
import sqlite3, sys
db = sqlite3.connect('Employee')
cursor = db.cursor()

id1, name1, dept1, salary1 = 8, 'Nitin', 'IT', 10000
id2, name2, dept2, salary2 = 9, 'Bob', 'IT', 20000

dataset = [(id1, name1, dept1, salary1), (id2, name2, dept2, salary2)]
cursor.executemany('''INSERT INTO emp values (?, ?, ?, ?)''', dataset)
db.commit()

cursor.execute('''Select * from emp where dept = ?''', ('IT',))
for record in cursor.fetchall():
    print list(record)

db.close()
```

Drop table

```
''' Drop a table '''

import sqlite3 # Importing module

db = sqlite3.connect('ethanstech')
dbCursor = db.cursor() # DB cursor

dbCursor.execute(""" DROP table employee""")

db.close()
```

Introduction to Machine Learning

- What is Machine Learning?
- Areas of Implementation of Machine Learning,
- Why companies prefer ML
- Major Classes of Learning Algorithms
- Supervised vs Unsupervised Learning, Learning
- Why Numpy?
- Learning Numpy and Scipy,
- Basic plotting using Matplotlib
- Algorithms using Skikit learn

What is Machine Learning?

Machine learning was defined in 1959 by Arthur Samuel as the "field of study that gives computers the ability to learn without being explicitly programmed." This means imbuing knowledge to machines without hard-coding it. – via Wikipedia

Machine learning is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look. – via SAS

Method of teaching computers or programs/scripts to make and improve predictions or behaviours based on some training data. Machine learning is a *huge* field, with hundreds of different algorithms for solving different stats problems – via Ethans

Fields of study



Gaming and AI

Samuel created a computer checkers in 1959, and seminal research on machine learning beginning in 1949. He thought that teaching computers to play games was very fruitful for developing tactics appropriate to general problems.



Robotics and self driven cars



Forecasting & Bioinformatics

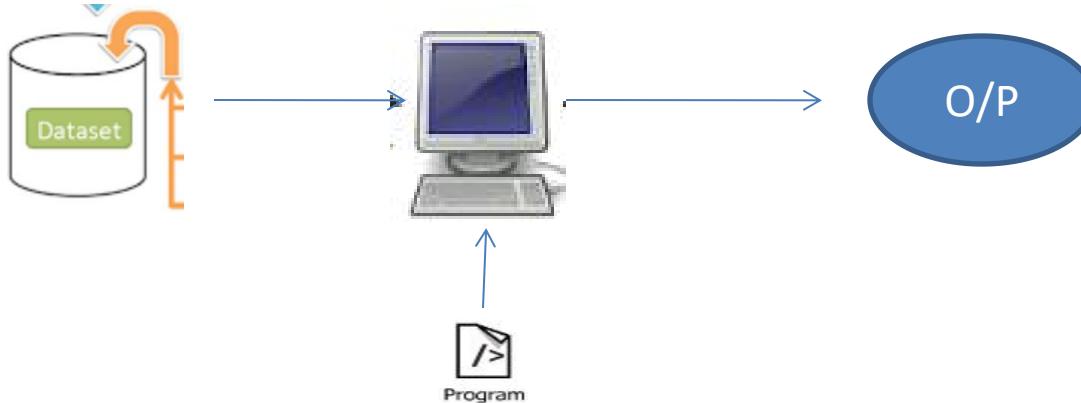


Why companies using Machine learning?

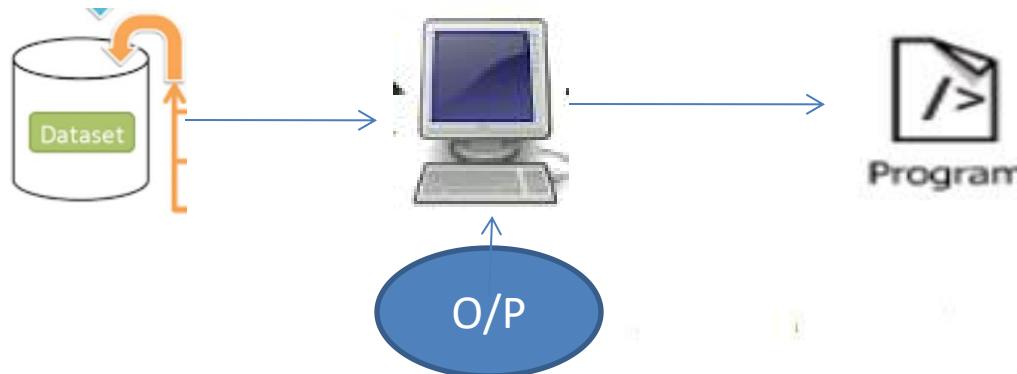
- Online recommendation engines like those from Amazon, snap deal and flip kart. Machine learning applications help them to sell products.
- Got to know the product sentiments from the market from Twitter and Facebook, Machine learning applications help them to analyse the market response.
- Remember face recognition and thumb impression software's for laptop and mobile, Machine learning applications help them to build cool products.
- Fraud detection on time, Machine learning applications help them to build secure products.
- What will be the weather forecast for tomorrow? , Machine learning applications help them to build forecasting products.
- Looking for the automated engine for routing the traffic in the cluster nodes? , Machine learning applications help them to build robotics products.

Comparison of two programming's

Traditional Programming



Machine Learning Application



How is machine learning used today?

Ever wonder how an online retailer provides nearly instantaneous offers for other products that may interest you? Or how lenders can provide near-real-time answers to your loan requests? Many of our day-to-day activities are powered by machine learning algorithms, including:

- Fraud detection.
- Web search results.
- Real-time ads on web pages and mobile devices.
- Text-based sentiment analysis.
- Credit scoring and next-best offers.
- Prediction of equipment failures.
- New pricing models.
- Network intrusion detection.
- Pattern and image recognition.
- Email spam filtering.

Via: SAS website

- Supervised learning
- Unsupervised learning
- Semi-supervised
- Reinforcement learning

Two of the most widely adopted machine learning methods are **supervised learning** and **unsupervised learning**. Most machine learning – about 70 percent – is supervised learning. Unsupervised learning accounts for 10 to 20 percent. Semi-supervised and reinforcement learning are two other technologies that are sometimes used. Via SAS

Supervised learning

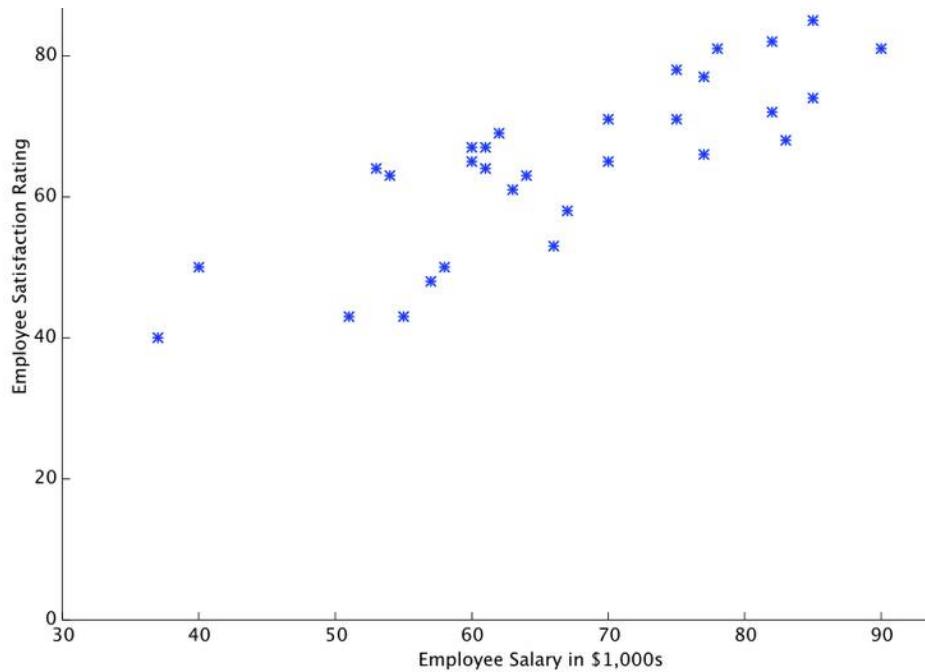
Supervised learning algorithms are trained using labeled training dataset. The training data consist of a set of training examples.

In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples



Examples of Supervised Learning



Linear Regression - Problem

Regression example

Predict House Pricing:

Based on: House size, floor Size, area, features, Builder etc

Predict Body BMI:

Based on: Sex, height, weight etc

Predict crop yields:

Based on : Soil conditions, rail fall in an areas, Fertilizers used, water level etc

Predict Life expectancy:

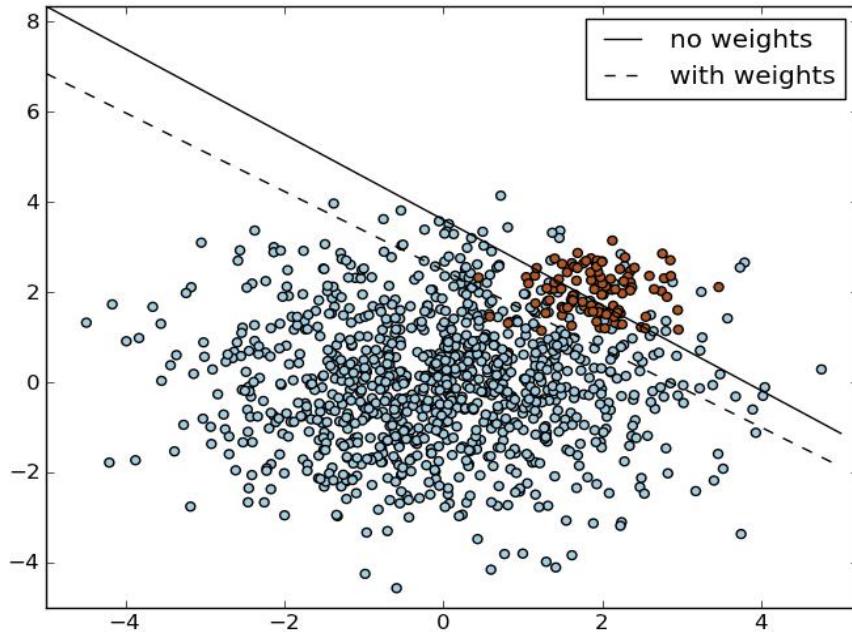
Based on: Eating Pattern, exercises pattern, smoker, drinker , disease status, forefather history etc.

Examples of Supervised Learning



Classification Problem

Examples of Supervised Learning



SVM Algorithm

text categorization - (e.g., spam filtering)

fraud detection – Finance, anti virus etc

optical character recognition – Handwriting recognition

machine vision - (e.g., palm detections, thumb impression face detections)

natural-language processing - (e.g., spoken language understanding)

market segmentation - (e.g. predict if customer will respond to promotion)

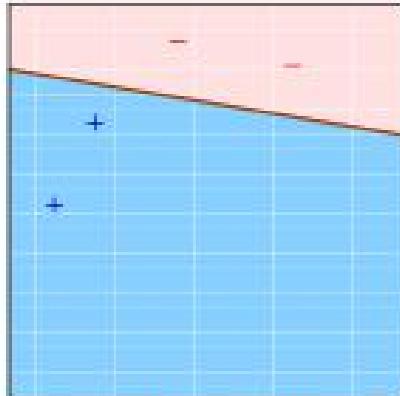
Bioinformatics - (e.g., classify proteins according to their function)

- Get enough training set examples
- It should good in performance on training set
- Classifier should not be not too “complex” to predict
- Classifiers should be “as simple as possible, but no simpler”
- “simplicity” closely related to prior expectations
- In general cases get two label of the classifiers
- Test it many times to get the expected result before finalizing.

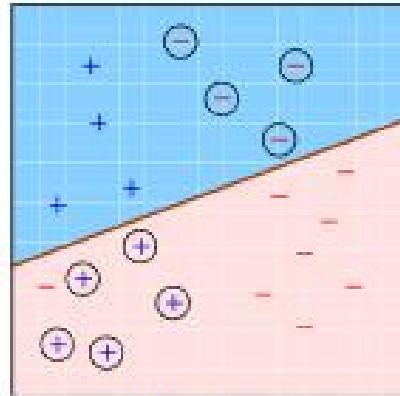
Measure “complexity” by:

- number bits needed to write down
- number of parameters

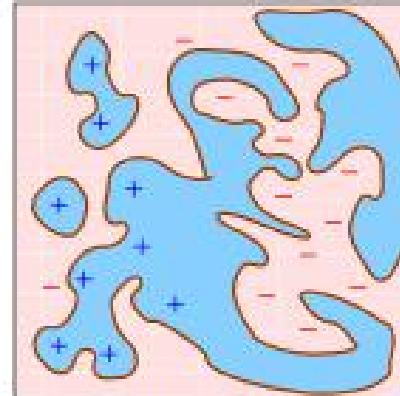
Bad Classifiers



insufficient data



training error
too high



classifier
too complex

Unsupervised Learning

Input data is not labelled and does not have a known result. Data is organised to form cluster of data.

A model is prepared by creating structures present in the input data.

This may be to extract general rules. It may through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity.

Example algorithms include: the k-Medians, Expectation Maximisation (EM), Hierarchical Clustering and k-Means.

Unsupervised example:



IPL 9, Qualifier 1: AB de Villiers' 'Superman' knock helps RCB storm ...

Zee News - 8 hours ago

With the spectacular win, RCB are into their third **IPL** final and one step away from their maiden title. Lions on other hand will have another ...

IPL: AB de Villiers pulls off a heist, RCB reach third final

Times of India - 9 hours ago

IPL 9 RCB vs GL: AB De Villiers stands tall amidst ruins, Bangalore ...

Deccan Chronicle - 7 hours ago

IPL 2016, Highlights: RCB vs GL - De Villiers, Abdulla Take Royal ...

NDTVSports.com - 10 hours ago

IPL: De Villiers guides RCB to **IPL 2016 final, beat Gujarat by 4 wickets**

Hindustan Times - 13 hours ago

Daniel Vettori and Bangalore into **IPL** final thanks to stars de Villiers ...

Opinion - Stuff.co.nz - 5 hours ago



Times of India



Deccan Chro...



NDTVSports....



Hindustan Tim...



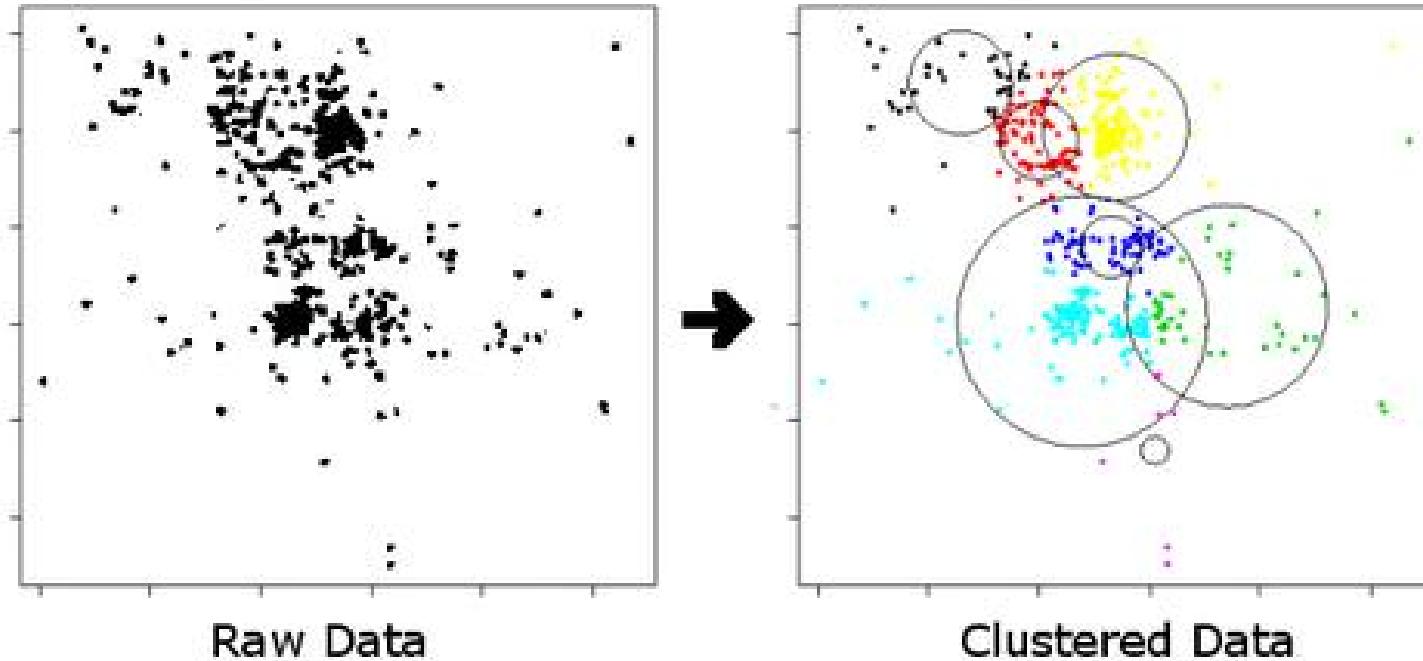
News18



ABP Live

Explore in depth (675 more articles)

Unsupervised example:



NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays

To install NumPy run:

python setup.py install

To perform an in-place build that can be run from the source folder run:

python setup.py build_ext --inplace

The NumPy build system uses distutils and numpy.distutils. setuptools is only used when building via pip or with python setupegg.py.

NumPy performance Test

```
__author__ = 'Ethans'

import timeit

# Sum of 10000 number from Python
py_sec = timeit.timeit
('sum(x*x for x in xrange(10000))', number=10000)

# Sum of 10000 number using numpy
np_sec = timeit.timeit
('na.dot(na)', setup="import numpy as np; na=np.arange(10000)", number=10000)

print("Normal Python: %f sec" %py_sec)
print("Through NumPy: %f sec" %np_sec)
```

>>>

```
Normal Python: 22.872751 sec
Through NumPy: 0.176976 sec
```

Getting Started with Numpy

```
>>> # Importing Numpy module  
>>> import numpy  
>>> import numpy as np
```

- Arrays are the central feature of NumPy.
- Arrays in Python are similar to lists in Python, the only difference being the array elements should be of the same type

```
import numpy as np  
  
a = np.array([1,2,3], float)      # Accept two arguments list and type  
  
print a                          # Return array([ 1.,  2.,  3.])  
  
print type(a)                    # Return <type 'numpy.ndarray'>  
  
print a[2]                        # 3.0  
  
print a[1:]                       # array([ 2.,  3.])  
  
print a[:2]                       # array([ 1.,  2.])
```

Two Dimensional Array

```
import numpy as np

a = np.array([[1,2,3], [4,5,6]], int) # Accept two arguments list and type

>>> a[0][0]  # Return 1

>>> a[0,0]  # Return 1

>>> a[1,2]  # Return 6

>>> a[1:]   # Return array([[4, 5, 6]])

>>> a[1,:]  # Return array([4, 5, 6])

>>> a[:2]   # Return array([[1, 2, 3],[4, 5, 6]])

>>> a[:,2]  # Return array([3, 6])

>>> a[:,1]  # Return array([2, 5])

>>> a[:,0]  # Return array([1, 4])
```

Two Dimensional Array

```
import numpy as np

a = np.array([[1,2,3], [4,5,6]], int)      # Accept two arguments list and type

>>> a.shape                                # Return the dimension of a i.e. (2,3)

>>> a.dtype                                 # Return the type of array i.e. dtype('int32')

>>> b = a.reshape(3,2)                      # Return the new shape of array

>>> b.shape                                # Return (3,2)

>>> len(a)                                  # Return length of a

>>> 2 in a                                  # Check if 2 is available in a

>>> b.tolist()                             # Return [[1, 2], [3, 4], [5, 6]]

>>> list(b)                                # Return [array([1, 2]), array([3, 4]), array([5, 6])]

>>> c = b

>>> b[0][0] = 10

>>> c                                     # Return array([[10, 2], [3, 4], [5, 6]])
```

Two Dimensional Array

```
>>> b                                     # Return array([[10, 2],[ 3, 4],[ 5, 6]])  
  
# Convert the multidimensional array to single dimension  
  
>>> b.flatten()                           # Return array([10, 2, 3, 4, 5, 6])  
  
  
# Concatenating two arrays  
  
>>> a = np.array([1,2,3])  
  
>>> b = np.array([4,5,6])  
  
>>> np.concatenate((a,b))                 # Returns array([1, 2, 3, 4, 5, 6])  
  
  
  
  
# arange function and ones function  
  
>>> a = np.arange(5, dtype=float)          # Return array([ 0., 1., 2., 3., 4.])  
  
>>> b = np.ones((3,2), dtype=float)        # Return array([[ 1., 1.], [ 1., 1.], [ 1., 1.]])
```

Update Array

```
# Assign the new values
```

```
>>> c = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
>>> c[0,0] = 10  
>>> c  
array([[10, 2, 3],  
       [ 4, 5, 6],  
       [ 7, 8, 9]])  
>>> c[:,2] = [10, 10, 10]  
>>> c  
array([[10, 2, 10],  
       [ 4, 5, 10],  
       [ 7, 8, 10]])  
>>> c[0, :] = [10, 10, 10]  
>>> c  
array([[10, 10, 10],  
       [ 4, 5, 10],  
       [ 7, 8, 10]])  
>>> c.shape
```

Standard Maths functions

```
# 'max', 'mean', 'min', sum, std functions
>>> np.arange(100).max()
99
>>> np.arange(100).mean()
49.5
>>> np.arange(100).min()
0
>>> np.arange(100).sum()
4950
>>> a.std()
2.8722813232690143
>>> a.all()
False
>>> a.any()
True
>>> a.nonzero()
(array([1, 2, 3, 4, 5, 6, 7, 8, 9]),)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Copy of arrays

```
# Referential operation also valid here
>>> a = np.array([1,2,3])
>>> a
array([1, 2, 3])
>>> b = a
>>> b[0] = 10
>>> a[2] = 20
>>> a
array([10, 2, 20])
>>> b
array([10, 2, 20])
```

```
# Base copy
>>> a
array([10, 2, 20])
>>> d = a.view()
>>> d
array([10, 2, 20])
>>> a[0] = 20
>>> d[1] = 30
>>> a
array([20, 30, 20])
>>> d
array([20, 30, 40])
>>> a is d
False
>>> d.base is a
True
```

```
# Shallow copy
>>> a
array([10, 2, 20])
>>> c = a.copy()
>>> c
array([10, 2, 20])
>>> a
array([10, 2, 20])
>>> c[0] = 20
>>> c
array([20, 2, 20])
>>> a
array([10, 2, 20])
```

List vs Array

```
# List operations
>>> list1 = [1,2,3]
>>> list2 = [4,5,6]

>>> list1 + list2
[1, 2, 3, 4, 5, 6]

>>> list2 * 2
[4, 5, 6, 4, 5, 6]
```

```
>>> import numpy as np
>>> a1 = np.array([1,2,3])
>>> a2 = np.array([4,5,6])
>>> a1 + a2
array([5, 7, 9])
>>> a1 * a2
array([ 4, 10, 18])
>>> a2 / a1
array([4, 2, 2])
>>> a1 ** a2
array([ 1, 32, 729])
```

Boolean Opr

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>>
>>> div = a % 2 == 0
>>> div
array([ True, False,  True, False,  True, False,  True, False], 
>>> a[div]
array([0, 2, 4, 6, 8])
```

Maths Functions

```
# Mathematical functions
>>> c
array([[1, 2, 4],
       [5, 7, 8]])
>>> c.transpose()
array([[1, 5],
       [2, 7],
       [4, 8]])
>>> np.invert(c)
array([[-2, -3, -5],
       [-6, -8, -9]])
>>> np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> c = np.array([[1,2], [2,3]])
>>> np.dot(c, c) # Returns dot product array([[ 5,  8],
>>> a = np.array([1,2,3,4,5,6,7,8,9])
>>> np.median(a) # Return 5.0
>>> np.corrcoef(a) # Return 1.0
```

```
# Documentation of any function
>>> np.info(np.sum)
sum(a, axis=None, dtype=None, out=None, keepdims=False)
```

Sum of array elements over a given axis.

Parameters

a : array_like
Elements to sum.

Sort function

```
# Sort function
>>> a
array([ 2,  4,  5,  1,  2,  3, 20, 10, 15])
>>> a.sort()
>>> a
array([ 1,  2,  2,  3,  4,  5, 10, 15, 20])
>>> a = np.array([2,4,5,1,2,3,20,10,15])
>>> np.sort(a)
array([ 1,  2,  2,  3,  4,  5, 10, 15, 20])
>>> a
array([ 2,  4,  5,  1,  2,  3, 20, 10, 15])
>>> a.argsort()
array([3, 0, 4, 5, 1, 2, 7, 8, 6])
>>> a[a.argsort()]
array([ 1,  2,  2,  3,  4,  5, 10, 15, 20])
>>> a = np.array([10,2,1])
>>> a.argsort()
array([2, 1, 0])
>>> a[a.argsort()]
array([ 1,  2, 10])
```

Tile function

```
>>> b = np.array([[1, 2], [3, 4]])  
>>> np.tile(b, 2)  
array([[1, 2, 1, 2],  
       [3, 4, 3, 4]])  
>>> np.tile(b, (2, 1))  
array([[1, 2],  
       [3, 4],  
       [1, 2],  
       [3, 4]])  
  
>>> c = np.array([1, 2, 3, 4])  
>>> np.tile(c, (4, 1))  
array([[1, 2, 3, 4],  
       [1, 2, 3, 4],  
       [1, 2, 3, 4],  
       [1, 2, 3, 4]])
```

Know Euclidean Distance

In mathematics the **Euclidean distance** or **Euclidean metric** is the "ordinary" (i.e. straight-line) distance between two points in Euclidean space. With this distance, Euclidean space becomes a metric space. The associated norm is called the **Euclidean norm**. [From Wikipedia](#)

One dimension [edit]

In one dimension, the distance between two points on the [real line](#) is the [absolute value](#) of their numerical difference. Thus if x and y are two points on the real line, then the distance between them is given by:

$$\sqrt{(x - y)^2} = |x - y|.$$

In one dimension, there is a single homogeneous, translation-invariant [metric](#) (in other words, a distance that is induced by a [norm](#)), up to a scale factor of length, which is the Euclidean distance. In higher dimensions there are other possible norms.

Two dimensions [edit]

In the [Euclidean plane](#), if $\mathbf{p} = (p_1, p_2)$ and $\mathbf{q} = (q_1, q_2)$ then the distance is given by

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

This is equivalent to the [Pythagorean theorem](#).

Alternatively, it follows from (2) that if the [polar coordinates](#) of the point \mathbf{p} are (r_1, θ_1) and those of \mathbf{q} are (r_2, θ_2) , then the distance between the points is

$$\sqrt{r_1^2 + r_2^2 - 2r_1r_2 \cos(\theta_1 - \theta_2)}.$$

Three dimensions [edit]

In three-dimensional Euclidean space, the distance is

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$

Problem statement - 1

Find out the nearest point between several defined co ordinates?

Problem Statement can be related with Google map, to find out the shortest distance between two destination?

Solution

```
# Shortest distance between two points
co_ordinates = {1:(3,10), 2:(1,9), 3:(10,20),
                 4:(2,7), 5:(11,30), 6:(14,45)}

check_ordinates = (4,22)
out_check = {}
for key, value in co_ordinates.items():
    distance = 0
    for i in range(len(value)):
        distance = distance + \
                    (co_ordinates[key][i] - check_ordinates[i]) ** 2

    out_check[key] = distance ** 0.5

sorted_distance = sorted(out_check.items(), key=lambda x: x[1])
print '%r is near to co_ordinates %r ' \
      %(check_ordinates, co_ordinates[sorted_distance[0][0]])
```

- Matplotlib is a Python package for 2D-graphics.
- It provides a way to visualize data from Python and provide quality figures in many formats
- For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with Python. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

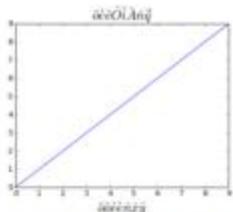
To install matplot lib run:

pip install matplotlib

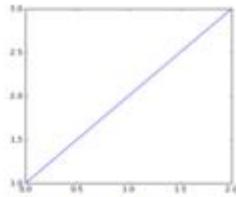
Matplot contains more than 100 figures and all of them can be accessed from:

<http://matplotlib.org/gallery.html>

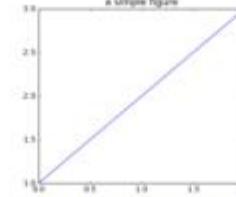
Sample Figures



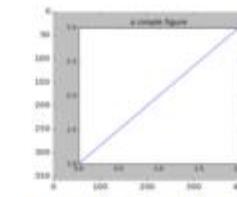
accented_text



agg_buffer



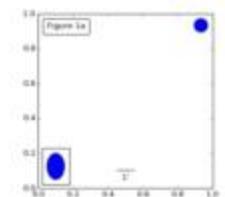
agg_buffer_to_array



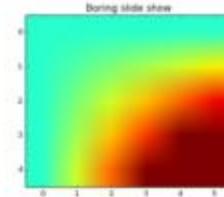
agg_buffer_to_array



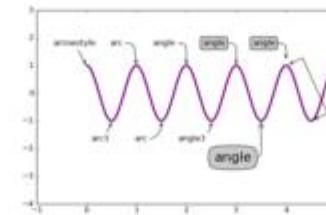
alignment_test



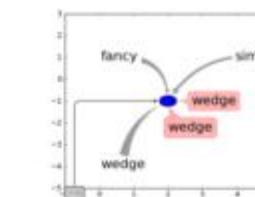
anchored_artists



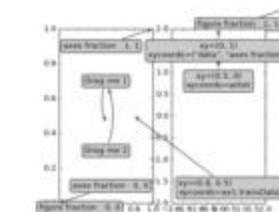
animation_demo



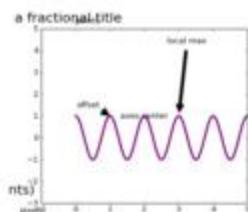
annotation_demo2



annotation_demo2



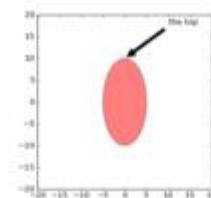
annotation_demo3



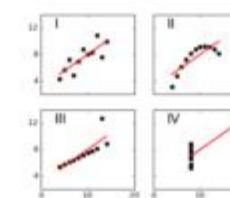
annotation_demo



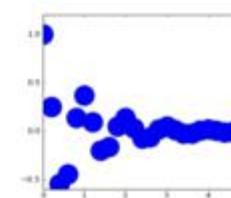
annotation_demo



annotation_demo



anscombe



arctest

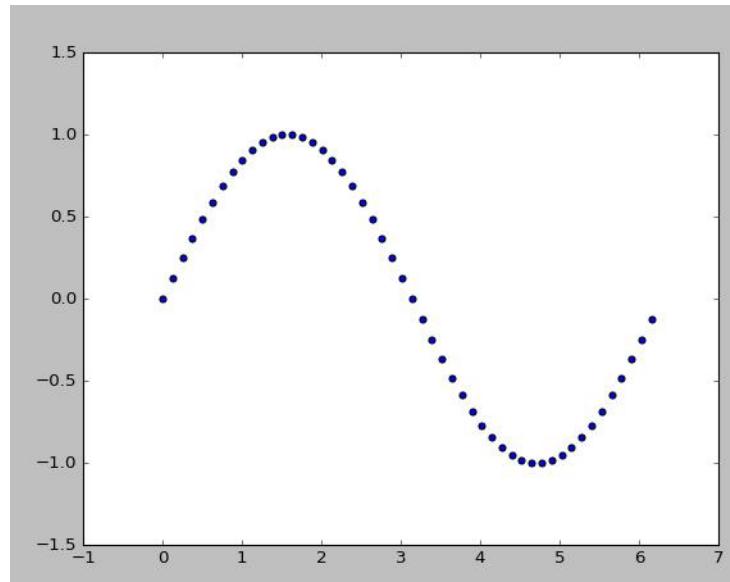
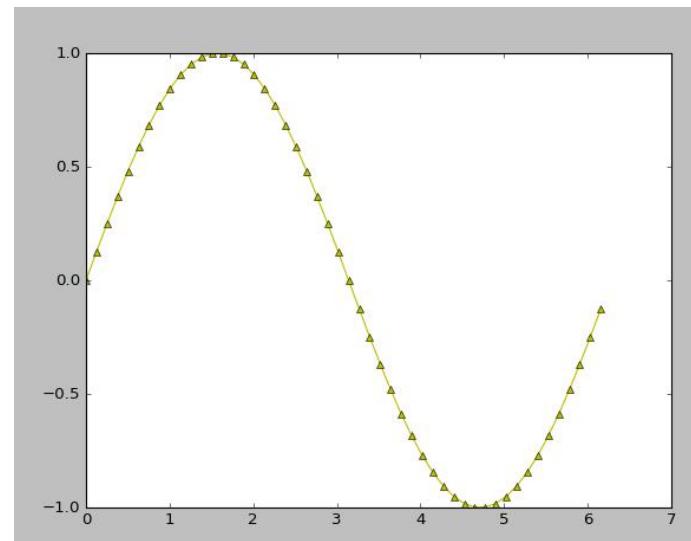
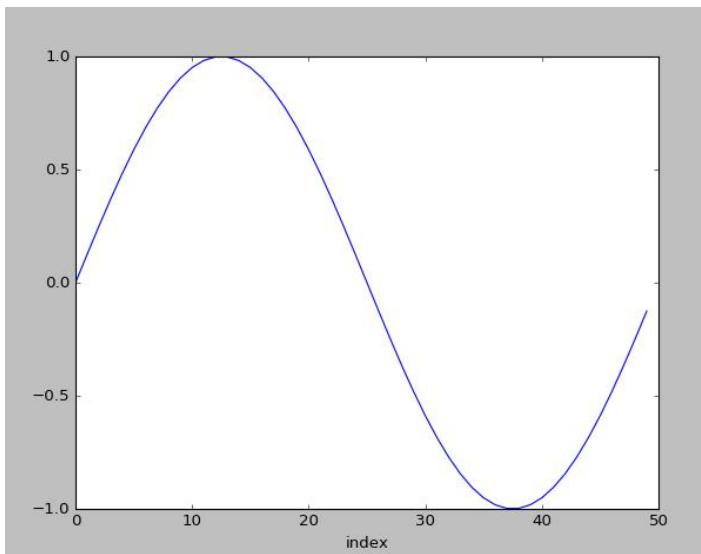
Plot Graph

```
# Line Graph
import matplotlib.pyplot as plt
from numpy import *
from pylab import *
x = arange(50)*2*pi/50
y = sin(x)
plt.plot(y)
plt.xlabel('index')
plt.show()

# Line formating
plt.plot(x, sin(x), 'y^-')
plt.show()

# Scatter Graph
plt.scatter(x, y)
plt.show()
```

Plot Graph



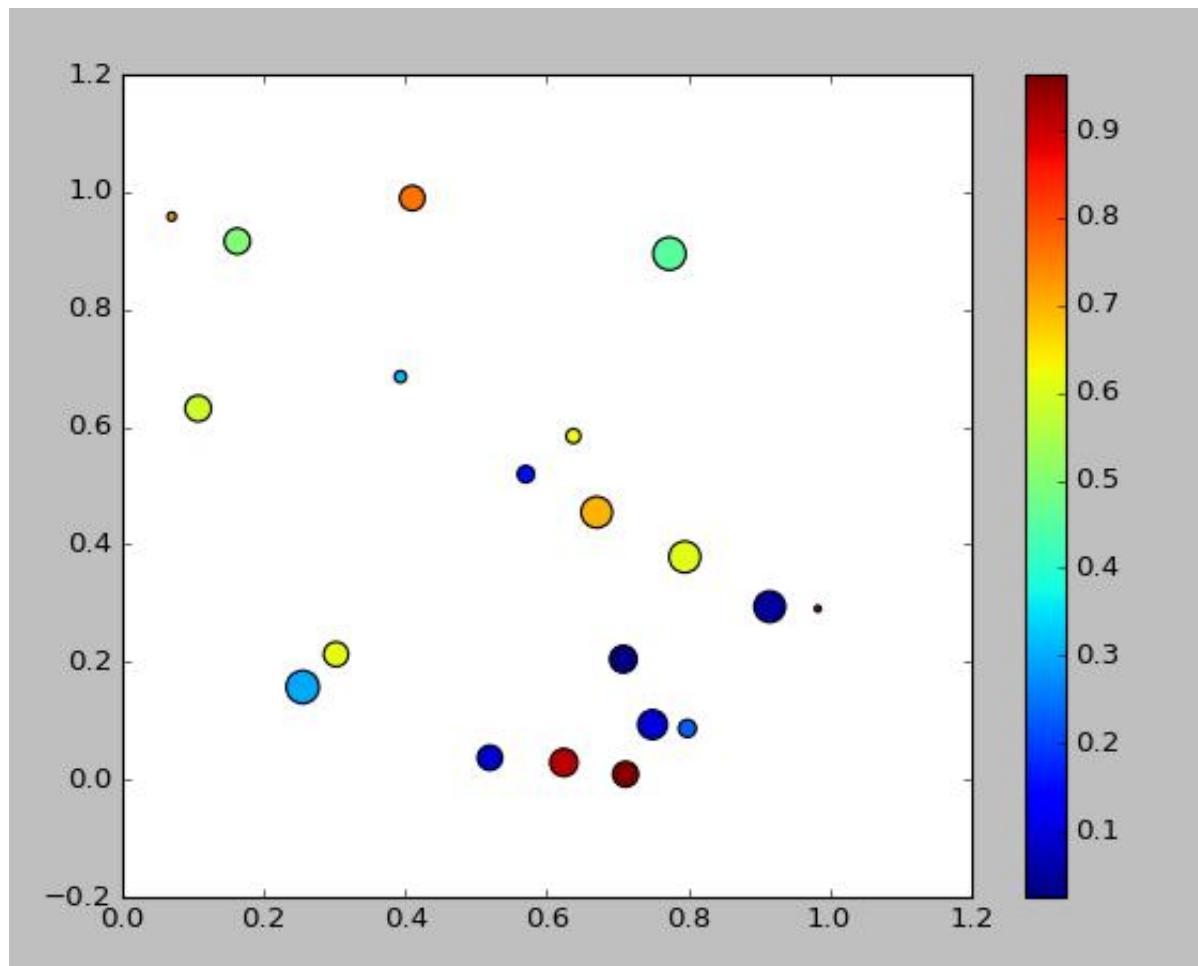
Other Graphs

```
# Color bar graph
from scipy import *
x = rand(20)
y = rand(20)
size = rand(20)*200
color= rand(20)
plt.scatter(x, y, size , color)
plt.colorbar()
plt.show()

# Bar graph
plt.bar(x, y, width = x[1]-x[0])
plt.show()

# histogram
plt.hist(randn(20))
plt.show()
```

Color Graph



Problem statement – 1 with Visuals

```
import matplotlib.pyplot as plt
from numpy import *
from pylab import *

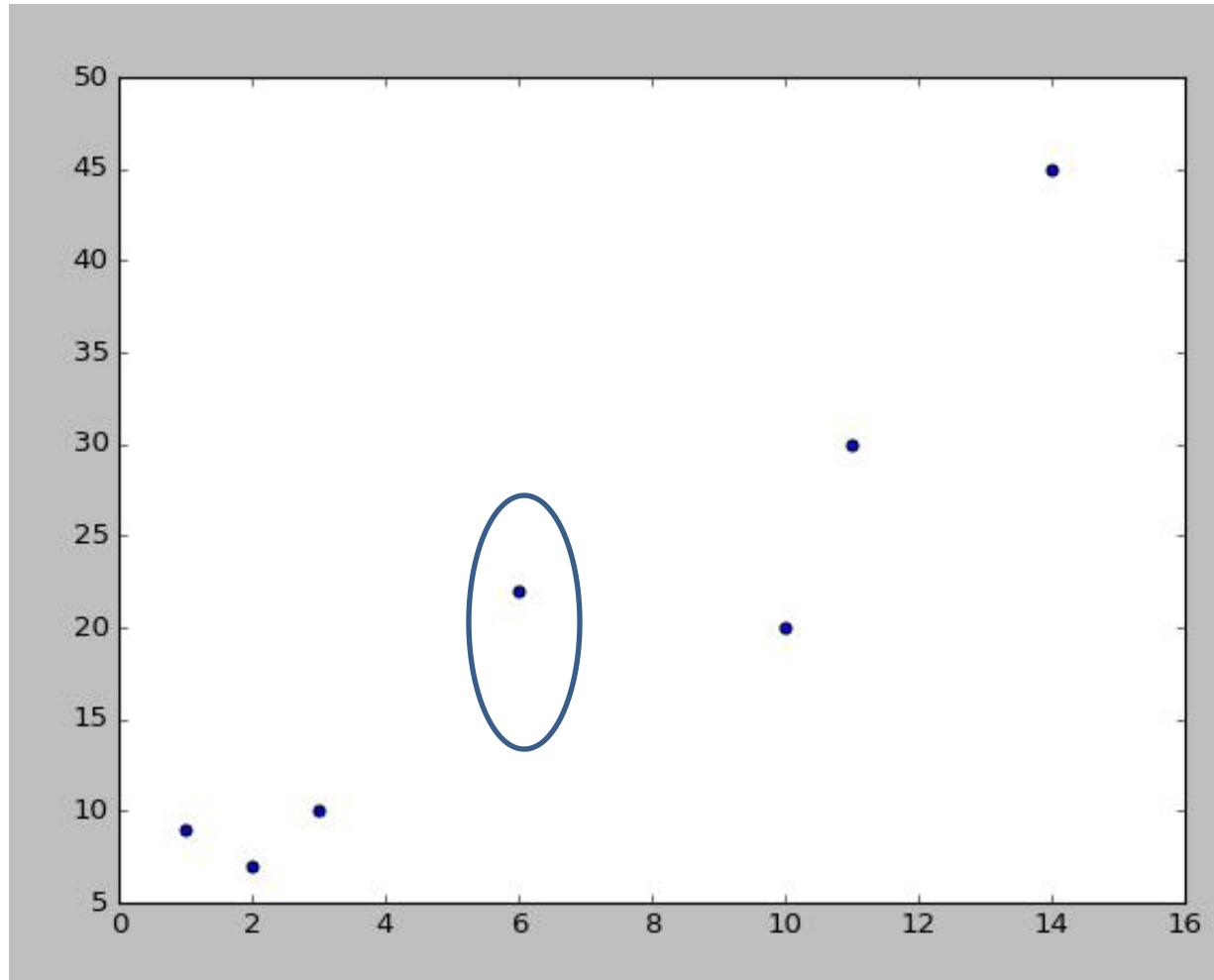
xco_coordinates = array([3,1,10,2,11,14])
yco_coordinates = array([10,9,20,7,30,45])

# Scatter Graph
plt.scatter(xco_coordinates, yco_coordinates)
plt.show()

# After adding the points
xco_coordinates = array([3,1,10,2,11,14, 6])
yco_coordinates = array([10,9,20,7,30,45, 22])

# Scatter Graph
plt.scatter(xco_coordinates, yco_coordinates)
plt.show()
```

Solution



Understanding KNN Algorithm



Movie Title	Distance to Movie "?"
California Man	20.5
He's Not really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer	117.4
Amped II	118.9

Movie Title	# of kicks	# of kisses	Type of Movie
California Man	3	104	Romance
He's Not really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer	99	5	Action
Amped II	98	2	Action

Creating Training Set

```
import numpy as np
from operator import itemgetter

# Training Dataset
def create_Dataset(dataset, label):
    group = np.array(dataset)
    labels = label
    return group, labels
```

Creating classifier

```
# Build a classifier
def classify(inX, dataset, labels, k):
    datasetSize = dataset.shape[0]
    # Taken x2-x1 and y2-y1 and so onnn
    diffMat = np.tile(inX, (datasetSize, 1)) - dataset
    # Taken the square of elements
    sqdiffMat = diffMat ** 2
    # Add x + y
    sqDistances = sqdiffMat.sum(axis=1)
    # Add square root of the result
    distance = sqDistances ** 0.5
    # Sort them based on indexes and get the all indexes
    sortedDist = distance.argsort()
    classCount = {}
    for i in range(k):
        voteLabel = labels[sortedDist[i]]
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1

    sortedCount = sorted(classCount.iteritems(), key=itemgetter(1), reverse=True)
    return sortedCount[0][0]
```



Call KNN

```
import KNN_algo as KNN

predefined_dataset = [[3, 104], [2, 100], [1, 81], [101, 10], [99, 5], [98, 2]]
label = ['Romance', 'Romance', 'Romance', 'Action', 'Action', 'Action']

group, labels = KNN.create_Dataset(predefined_dataset, label)

set1 = [18, 90]
#set2 = [2, 199]

print 'Set1 is near point: ', KNN.classify(set1, group, labels, 3)
#print 'Set2 is near point: ', KNN.classify(set2, group, labels, 3)
```

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under BSD and distribute under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

NumPy: Base n-dimensional array package

SciPy: Fundamental library for scientific computing

Matplotlib: Comprehensive 2D/3D plotting

IPython: Enhanced interactive console

Sympy: Symbolic mathematics

Pandas: Data structures and analysis

What are the features?

The library is focused on modelling data but not focused on loading, manipulating and summarizing data. For these features, we either use NumPy and Pandas.

Some popular groups of models provided by scikit-learn include:

Supervised Models: a vast array not limited to generalized linear models, discriminative analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

Clustering: for grouping unlabeled data such as KMeans.

Cross Validation: for estimating the performance of supervised models on unseen data.

Datasets: for test datasets and for generating datasets.

Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.

Ensemble methods: for combining the predictions of multiple supervised models.

Feature extraction: for defining attributes in image and text data.

Feature selection: for identifying meaningful attributes from which to create supervised models.

Manifold Learning: For summarizing and depicting complex multi-dimensional data.

Install scikit-learn

```
pip install scikit-learn
```

import it using

```
import sklearn
```



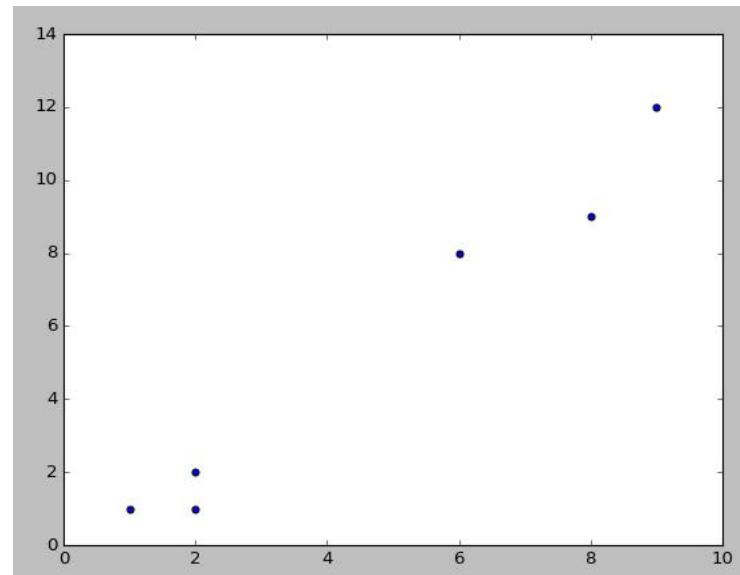
Linear SVC

The Support Vector Machine, created by Vladimir Vapnik in the 60s, but pretty much overlooked until the 90s is still one of most popular machine learning classifiers.

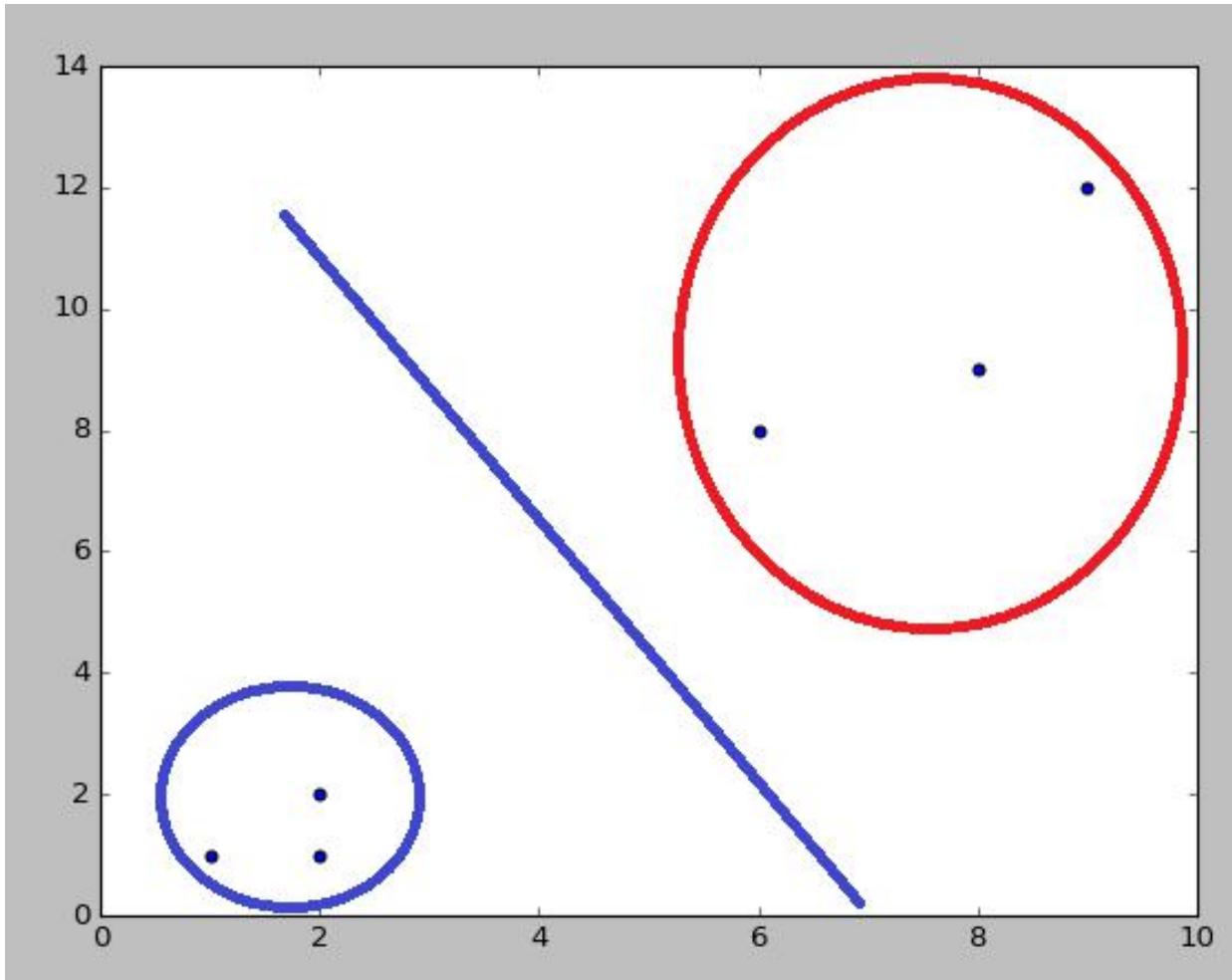
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

x = [2, 6, 2, 8, 1, 9]
y = [1, 8, 2, 9, 1, 12]

plt.scatter(x, y)
plt.show()
```

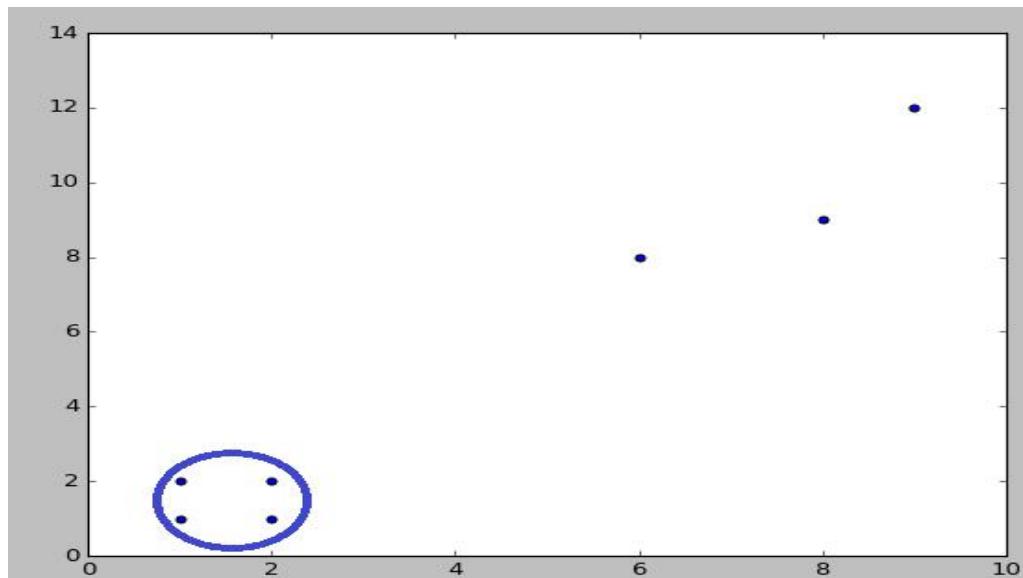


How it works?



Classifier

```
# Classifier starts here
from sklearn import svm
coordinates = np.array([[2,1],[6,8],[2,2],[8,9],[1,1],[9,12]])
labels = ['Action','Romance','Action','Romance','Action','Romance']
clf = svm.SVC(kernel='linear')
clf.fit(coordinates, labels)
print(clf.predict([1,2]))
```



Clustering

The difference between supervised and unsupervised machine learning is whether or not we have provided the machine with labeled data.

Unsupervised machine learning is where we does not provide the machine with labeled data, and the machine is expected to derive structure from the data all on its own.

Kmeans

```
from sklearn.cluster import KMeans
coordinates = np.array([[2,1],[6,8],[2,2],[8,9],[1,1],[9,12]])
kmeans = KMeans(n_clusters=2)
kmeans.fit(coordinates)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_

print 'Printing all centroids: ', centroids
print 'Printing all labels: ', labels
```

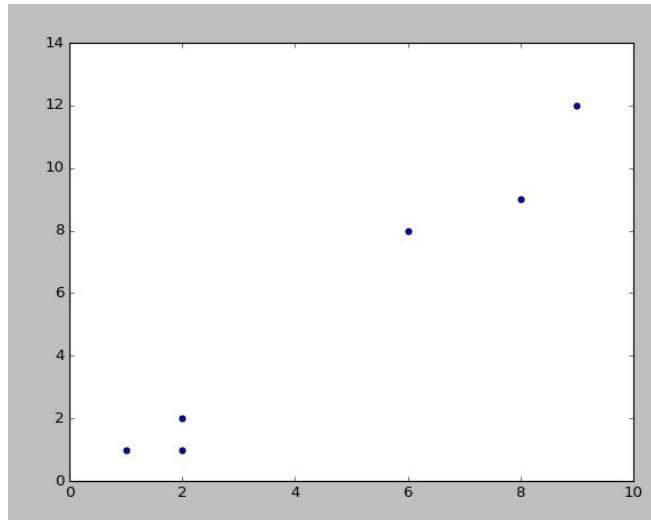
Kmeans

```
from sklearn.cluster import KMeans
coordinates = np.array([[2,1],[6,8],[2,2],[8,9],[1,1],[9,12]])
kmeans = KMeans(n_clusters=2)
kmeans.fit(coordinates)

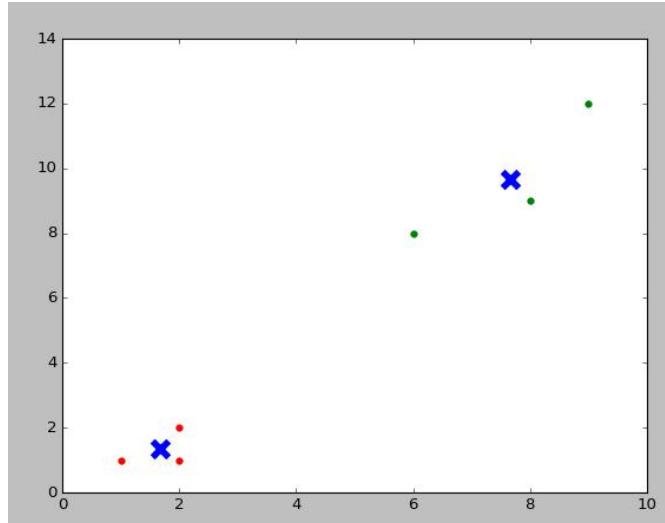
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

print 'Printing all centroids: ', centroids
print 'Printing all labels: ', labels
```

Kmeans - Graph



Before Clustering them



After Clustering

Python Pandas

- What is Pandas
- Creating Series
- Creating Data Frames,
- Grouping, Sorting
- Plotting Data
- Data analysis with data set
- Practical use cases using data analysis.

What is Pandas?

- pandas is a Python package for providing fast, flexible, and expressive data structures.
- Data structures designed in pandas to make working with ‘relational’ or ‘labeled’ data
- Pandas is the most powerful and flexible open source data analysis / manipulation tool available in python.
- pandas is built on top of NumPy module and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.
- It handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.
- Source - <http://pandas.pydata.org/pandas-docs/stable/>

- The two primary data structures of pandas are built on top of NumPy they are Series (1-dimensional) and DataFrame (2-dimensional).

These DS will help to do:

- Easy handling of missing data and adding/removing of columns
- Automatic data alignment and powerful, flexible group by functionality to perform operations on data sets, for both aggregating and transforming data
- Intelligent label-based slicing, fancy indexing, and sub setting of large data sets
- Easy merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Robust IO tools for loading data from flat files.

- Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).
- The axis labels are collectively referred to as the **index**

Create pandas Series

```
>>> import pandas as pd  
>>> S = pd.Series(data, index=index)
```

Data in above can be: List, Tuple, dict, NumPy array or any scalar value
Index – Should contains the label. **index** must be the same length as **data** .

```
>>> S = pd.Series('Ethans', index = ('Name',))  
>>> S  
Name    Ethans
```

Series – With other objects

```
>>> import pandas as pd
>>> S = pd.Series([1,2,3])
>>> S
0    1
1    2
2    3
dtype: int64
>>> S = pd.Series((1,2,3))
>>> S
0    1
1    2
2    3
dtype: int32
>>> S = pd.Series({1:2, 3:4})
>>> S
1    2
3    4
dtype: int64
```

Series – With numpy and LoL

```
>>> S = pd.Series(np.array([1,2,3]), index = (1,2,3))
>>> S
1    1
2    2
3    3
dtype: int32
>>>
>>> S = pd.Series(np.array([1,2,3]))
>>> S
0    1
1    2
2    3
dtype: int32
```

```
>>> employees = pd.Series([('ethans', 'Bob', 'Aaron'), [35, 40, 29]])
>>> employees
0    [ethans, Bob, Aaron]
1    [35, 40, 29]
dtype: object
```

Create Series: All example

```
# Creating a series in Pandas
import pandas as pd
s1 = pd.Series([1,2,3]) # from list
s1 = pd.Series((1,2,3)) # from tuple
s1 = pd.Series(list('pandas')) # from string
s1 = pd.Series((1,2,3), index=(1,2,3)) # from tuple, with indexes
s1 = pd.Series({1:2,3:4}) # from dictionary
s1 = pd.Series({1:2,3:4}, index= (1,2,3,4,5)) # from dictionary

import numpy as np
s1 = pd.Series(np.array([1,2,3]), index=(1,2,3)) # from np, with indexes
s1 = pd.Series([('Jatin','Aakash','Rahul'), [35, 28, 31]],
               index=('names', 'age')) # from list of list
```

Series – Get Index value

```
>>> d = {'Pune': 900, 'Mumbai': 1300, 'Delhi': 900, 'Bangalore': 1100,  
'Goa': 450, 'Daman': None}  
  
>>> cities = pd.Series(d)  
>>> cities  
Bangalore    1100.0  
Daman        NaN  
Delhi        900.0  
Goa          450.0  
Mumbai       1300.0  
Pune         900.0  
dtype: float64|
```

```
>>> cities['Pune']  
900.0  
>>> cities[['Pune', 'Mumbai', 'Goa']]  
Pune    900.0  
Mumbai  1300.0  
Goa     450.0  
dtype: float64
```

Series – Call basic functions

```
>>> cities.isnull()  
Bangalore    False  
Daman        True  
Delhi         False  
Goa           False  
Mumbai        False  
Pune          False  
dtype: bool  
>>> cities.notnull()  
Bangalore    True  
Daman        False  
Delhi         True  
Goa           True  
Mumbai        True  
Pune          True  
dtype: bool  
>>> cities[cities.isnull()]  
Daman    NaN
```

```
>>> cities < 1000  
Bangalore    False  
Daman        False  
Delhi         True  
Goa           True  
Mumbai        False  
Pune          True  
dtype: bool  
>>> cities[cities < 1000]  
Delhi      900.0  
Goa        450.0  
Pune      900.0  
dtype: float64  
>>> cities[cities == 900]  
Delhi      900.0  
Pune      900.0  
dtype: float64
```

Series – Call basic functions

```
>>> cities * 2
Bangalore    2200.0
Daman        NaN
Delhi         1800.0
Goa           900.0
Mumbai        2600.0
Pune          1800.0
dtype: float64
>>> np.square(cities)
Bangalore    1210000.0
Daman        NaN
Delhi         810000.0
Goa           202500.0
Mumbai        1690000.0
Pune          810000.0
dtype: float64
```

```
>>> cities['Pune'] = 800
>>> cities[cities > 1000] = 800
>>> cities
Bangalore    800.0
Daman        NaN
Delhi         900.0
Goa           450.0
Mumbai        800.0
Pune          800.0
dtype: float64
>>> 'Chennai' in cities
False
>>> 'Daman' in cities
True
```

- Data Frame is two-dimensional labeled array capable of holding any data type. DataFrame is a tabular data structure comprised of rows and columns, like a spreadsheet, database table, or R's data.frame object.

Create pandas Data Frame

```
>>> import pandas as pd
>>> df = pd.DataFrame({'names':['Ethan', 'Bob', 'Aaron'],
                      'Age':[30, 35, 40],
                      'City':['Pune', 'New York', 'Texas']})
>>>
>>> df
   Age      City  names
0   30      Pune  Ethan
1   35  New York    Bob
2   40     Texas  Aaron
```

Data Frame – with Series

```
>>> d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),  
       'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}  
  
>>>  
>>> df1 = pd.DataFrame(d)  
>>> df1  
   one  two  
a  1.0  1.0  
b  2.0  2.0  
c  3.0  3.0  
d  NaN  4.0
```

```
>>> df1 = pd.DataFrame(d, index = ['d', 'c', 'a'])  
>>> df1  
   one  two  
d  NaN  4.0  
c  3.0  3.0  
a  1.0  1.0
```

```
>>> df1 = pd.DataFrame(d, index = ['d', 'c', 'a'],  
                      columns = ['one', 'newOne'])  
>>> df1  
   one  newOne  
d  NaN      NaN  
c  3.0      NaN  
a  1.0      NaN
```

Data Frame – with LoD

```
>>> d = [{‘a’: 1, ‘b’: 2}, {‘a’: 3, ‘b’: 4, ‘c’: 5}]  
>>> df = pd.DataFrame(d)  
>>> df  
      a   b     c  
0    1   2   NaN  
1    3   4   5.0  
>>> pd.DataFrame(d, index=[‘first’, ‘second’])  
      a   b     c  
first  1   2   NaN  
second 3   4   5.0  
>>> pd.DataFrame(d, columns=[‘a’, ‘b’])  
      a   b  
0    1   2  
1    3   4
```

selection, addition and deletion

```
>>> D = {'India':[1, .4, 'Delhi'], 'China': [1.5, .2, 'Beijing']}
>>> df = pd.DataFrame(D, index = ['Population', 'PD', 'Capital'])
>>> df
          China  India
Population      1.5      1
PD              0.2     0.4
Capital        Beijing  Delhi
>>> df['India']
Population      1
PD              0.4
Capital        Delhi
Name: India, dtype: object
>>>
>>> df['India']['PD']
0.4
>>> df['Brazil'] = [.5, .4, 'Brasilia']
>>> df
          China  India    Brazil
Population      1.5      1      0.5
PD              0.2     0.4     0.4
Capital        Beijing  Delhi  Brasilia
```

```
>>> del df['China']
>>> df
          India    Brazil
Population      1      0.5
PD              0.4     0.4
Capital        Delhi  Brasilia
```

Functions

```
>>> data = {'Country':['India', 'China', 'Brazil'],
           'Year':[2013, 2014, 2015],
           'Population':[1,1.5, .5]}
```

```
>>>
```

```
>>> df = pd.DataFrame(data)
```

```
>>> df
```

	Country	Population	Year
0	India	1.0	2013
1	China	1.5	2014
2	Brazil	0.5	2015

```
>>> df.describe()
```

	Population	Year
count	3.00	3.0
mean	1.00	2014.0
std	0.50	1.0
min	0.50	2013.0
25%	0.75	2013.5
50%	1.00	2014.0
75%	1.25	2014.5
max	1.50	2015.0

```
>>> df.sum()
Country          IndiaChinaBrazil
Population            3
Year                  6042
dtype: object
>>> df.mean()
Population        1.0
Year              2014.0
dtype: float64
>>> df.max()
Country          India
Population        1.5
Year              2015
```

```
>>> df.head(1)
Country  Population  Year
0   India       1.0  2013
>>> df.tail(1)
Country  Population  Year
2   Brazil       0.5  2015
```

Pandas IO Basics - Tools

Reader functions

- `read_csv`
- `read_excel`
- `read_hdf`
- `read_sql`
- `read_json`
- `read_msgpack` (experimental)
- `read_html`
- `read_gbq` (experimental)
- `read_stata`
- `read_sas`
- `read_clipboard`
- `read_pickle`

Writer functions

- `to_csv`
- `to_excel`
- `to_hdf`
- `to_sql`
- `to_json`
- `to_msgpack` (experimental)
- `to_html`
- `to_gbq` (experimental)
- `to_stata`
- `to_clipboard`
- `to_pickle`

Read csv

```
df = pd.read_csv(r'C:\ethans\Training\Python\India_Population.csv')
```

```
>>> df.head()
      Date      Value
0  2020-12-31  1380.007
1  2019-12-31  1362.087
2  2018-12-31  1344.401
3  2017-12-31  1326.944
4  2016-12-31  1309.713
```

```
>>> df.tail()
      Date      Value
36  1984-12-31  747.000
37  1983-12-31  731.000
38  1982-12-31  715.563
39  1981-12-31  699.938
40  1980-12-31  685.688
```

```
>>> df.describe()
              Value
count    41.000000
mean    1026.812854
std     210.235406
min     685.688000
25%    847.438000
50%    1029.188000
75%    1195.063000
max    1380.007000
```

```
>>> len(df)
41
>>> df.columns
Index([u'Date', u'Value'], dtype='object')
>>> df.set_index('Date', inplace=True)
```

```
>>> df.head()
                Value
Date
2020-12-31  1380.007
2019-12-31  1362.087
2018-12-31  1344.401
2017-12-31  1326.944
2016-12-31  1309.713
```

Write html

```
>>> df.columns = ['Population']
>>> df.head()
          Population
Date
2020-12-31    1380.007
2019-12-31    1362.087
2018-12-31    1344.401
2017-12-31    1326.944
2016-12-31    1309.713
>>> df.to_html('IndiaPopulation.html')

>>> df.iloc[1]
Value    1362.087
Name: 2019-12-31 00:00:00, dtype: float64
>>> df[df.Value > 1000]

>>> df[df.Value > 1000].tail(1)
          Value
Date
1999-12-31  1010.188
```

	Population
Date	
2020-12-31	1380.007
2019-12-31	1362.087
2018-12-31	1344.401
2017-12-31	1326.944
2016-12-31	1309.713
2015-12-31	1292.707
2014-12-31	1275.921
2013-12-31	1259.353
2012-12-31	1243.000
2011-12-31	1217.438

Concatenating

```
import pandas as pd

df1 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[500, 550, 650, 650]},
                    index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'PSR':[709, 750, 802, 890]},
                    index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({'FSI':[80,85,88,85],
                    'Interest_rate':[2, 3, 2, 2],
                    'Govt_circle_rate':[450, 520, 570, 590]},
                    index = [2001, 2002, 2003, 2004])
```

```
# Concatenating df1 and df2, columns are common
concat = pd.concat([df1,df2])
print concat

# Concatenating df1 and df3, columns are common
concat = pd.concat([df1,df3])
print concat

# Concatinating df1, df2 and df3, columns are different
concat = pd.concat([df1,df2,df3])
print(concat)
```

Appending

```
import pandas as pd

df1 = pd.DataFrame({'FSI':[80,85,88,85],
                     'Interest_rate':[2, 3, 2, 2],
                     'PSR':[500, 550, 650, 650]},
                     index = [2001, 2002, 2003, 2004])

df2 = pd.DataFrame({'FSI':[80,85,88,85],
                     'Interest_rate':[2, 3, 2, 2],
                     'PSR':[709, 750, 802, 890]},
                     index = [2005, 2006, 2007, 2008])

df3 = pd.DataFrame({'FSI':[80,85,88,85],
                     'Interest_rate':[2, 3, 2, 2],
                     'Govt_circle_rate':[450, 520, 570, 590]},
                     index = [2001, 2002, 2003, 2004])
```

```
# Same columns appending
df4 = df1.append(df2)
print(df4)
```

```
# Different columns appending
df5 = df1.append(df3)
print(df5)
```

Merging

```
#-----#
# Merging
raw_data = {
    'subjectID': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
    'Marks': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
df1 = pd.DataFrame(raw_data)
print 'Subjects and Marks ', '--' * 30
print df1

raw_data = {
    'subjectID': ['1', '2', '3', '4', '5'],
    'firstname': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'lastname': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df2 = pd.DataFrame(raw_data)
print 'Names and SubjectID ', '--' * 30
print df2
```

```
#Join
print 'Inner Join ', '--' * 30
print pd.merge(df1, df2, on='subjectID')
```

```
#right join
print 'Right Join', '--' * 30
print pd.merge(df1, df2, on='subjectID', how='right')
```

```
#Left join
print 'Left Join', '--' * 30
print pd.merge(df1, df2, on='subjectID', how='left')
```

MovieLens

GroupLens Research has collected and made available rating data sets from the MovieLens web site (<http://movielens.org>). The data sets were collected over various periods of time, depending on the size of the set. Before using these data sets, please review their README files for the usage licenses and other details.

Help our research lab: Please [take a short survey](#) about the MovieLens datasets

MovieLens 100K Dataset

Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

- [README.txt](#)
- [ml-100k.zip](#) (size: 5 MB, [checksum](#))
- [Index of unzipped files](#)

Permalink: <http://grouplens.org/datasets/movielens/100k/>

Analyzing Data Set – User Data

User Data:

'UserId', 'Age', 'Sex', 'Occ', 'Zip'

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
11|39|F|other|30329
12|28|F|other|06405
```

```
user_col = ['UserId', 'Age', 'Sex', 'Occ', 'Zip']
users = pd.read_csv('u.user', sep='|', names = user_col)
```

Data Set – Rating Data

Rating Data:

'UserId', 'MovieId', 'rating', 'timeStamp'

196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817

```
data_col = ['UserId', 'MovieId', 'rating', 'time']
ratingData = pd.read_csv('u.data', sep='\t', names = data_col)
```

Data Set – Movie Data

Movie Data:

```
'Movield', 'title', 'release', 'videoRelease', 'url'
```

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(1995)
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%20
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shorty%20
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(1995)
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Shanghai+Triad+(Yao+a+yao+yao+dao+waipo+qiao)+(1995)
```

```
movie_col = ['Movield', 'title', 'release', 'videoRelease', 'url']
MovieData = pd.read_csv('u.item', sep='|', names = movie_col, usecols = range(5))
```

Problem Statements and Solutions

1 - Find the 5 top rated movies in the list.

```
movie_rating = pd.merge(MovieData, ratingData)
data = pd.merge(movie_rating, users)

print data.head(5)

most_rated = data.groupby('title').size().sort_values(ascending=False) [:5]
print most_rated

most_rated.plot()
show()
```

2 – Which age group users provide the maximum ratings?

```
#####
##2
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
data['age_group'] = pd.cut(data.Age, range(0, 81, 10), right = False, labels = labels)
print data.head(5)
print data.groupby('age_group').size().sort_values(ascending=False) [:1]
```

pandas_datareader

```
__author__ = "Ethan's"

import pandas as pd
import datetime
from pandas_datareader import data
import matplotlib.pyplot as pyplot

startDate = datetime.datetime(2015, 1, 1)
endDate = datetime.datetime(2016, 1, 1)
df = data.DataReader("AAPL", "google", startDate, endDate)

print(df.head())
df['High'].plot()
pyplot.legend()
pyplot.show()
```

Hadoop data Processing with Python

- Introduction of Big Data
- Why Big Data
- Hadoop Eco system
- Understanding problem statements
- Market data Analysis with Python
- HDFS file system
- Cloudera Cluster of single node
- Map Reduce using Python
- Introduction to MrJob Package

What is Big Data?

Dictionary says: Extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions.

Some Examples:

- Air Bus A380 engine generates apx 10 TB of data every 30 Min.
- Facebook generates apx 20 TB of data per day .
- Twitter generates apx 20 TB of data per day.
- Google processes apx 20 PB a day (2008)
- New York Stock Exchange apx 1TB of data everyday.

We are Living in data driven world

Telecom

Huge data all details records, messaging, whatapp, Hike and so on..

- Science

Huge Data from environmental data, transportation data, sensors, CCTV and so on

- Humanities and Social Sciences

Scanned books, Gmail, Facebook, twitter, social interactions data, new technology like GPS>

- Business & Commerce

Sales, Online advertising, stock market transactions, airline traffic etc

- Entertainment

Internet images, Movies, MP3 files etc

- Medicine

MRI & CT scans, patient records, DNA profiles, stem cells

Why Big Data is important?

The importance of big data doesn't revolve around how much data you have, but what you do with it. You can take data from any source and analyze it to find answers that enable:

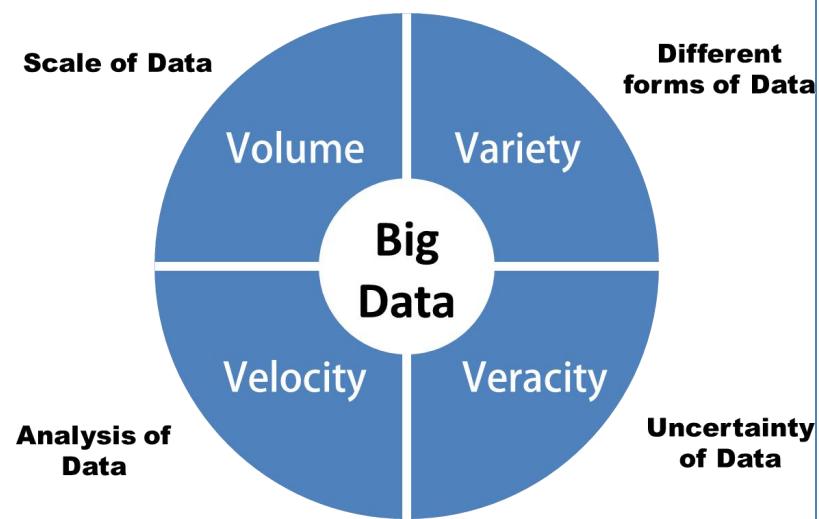
- 1) cost reductions, 2) time reductions, 3) new product development and optimized offerings, and 4) smart decision making.

When you combine big data with high-powered analytics, you can accomplish business-related tasks such as:

- Determining root causes of failures, issues and defects in near-real time – Examples of Telecom and Airline
- Generating coupons at the point of sale based on the customer's buying habits. Examples of Online advertising
- Recalculating entire risk portfolios in minutes. – Examples of Share Market
- Detecting fraudulent behaviour before it affects your organization. – Examples of Anti Virus software's and many

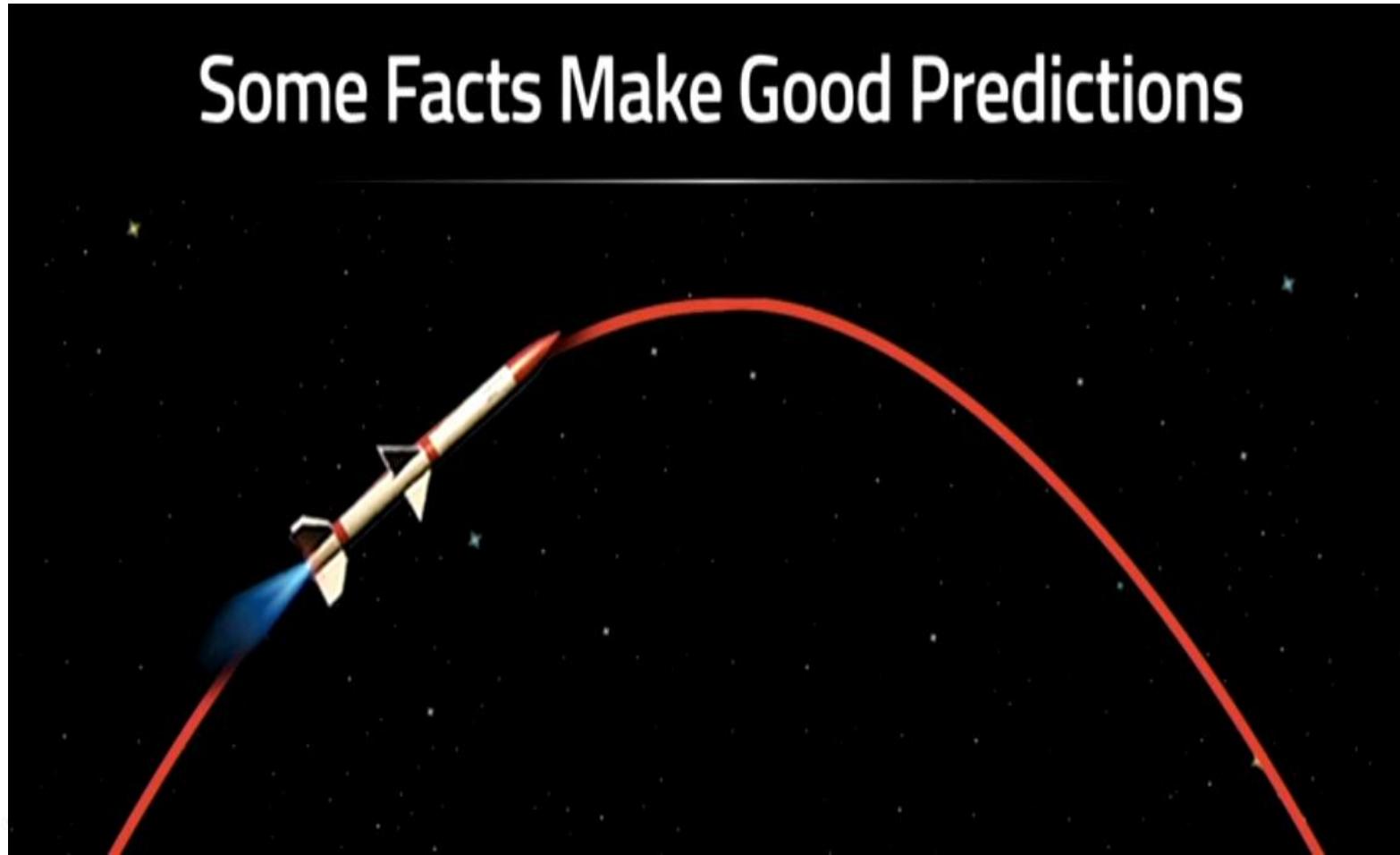
Big Data Vectors

- Volume - To many bytes
- Velocity - To high rate
- Variety - To many sources
- Veracity – Uncertainty of data

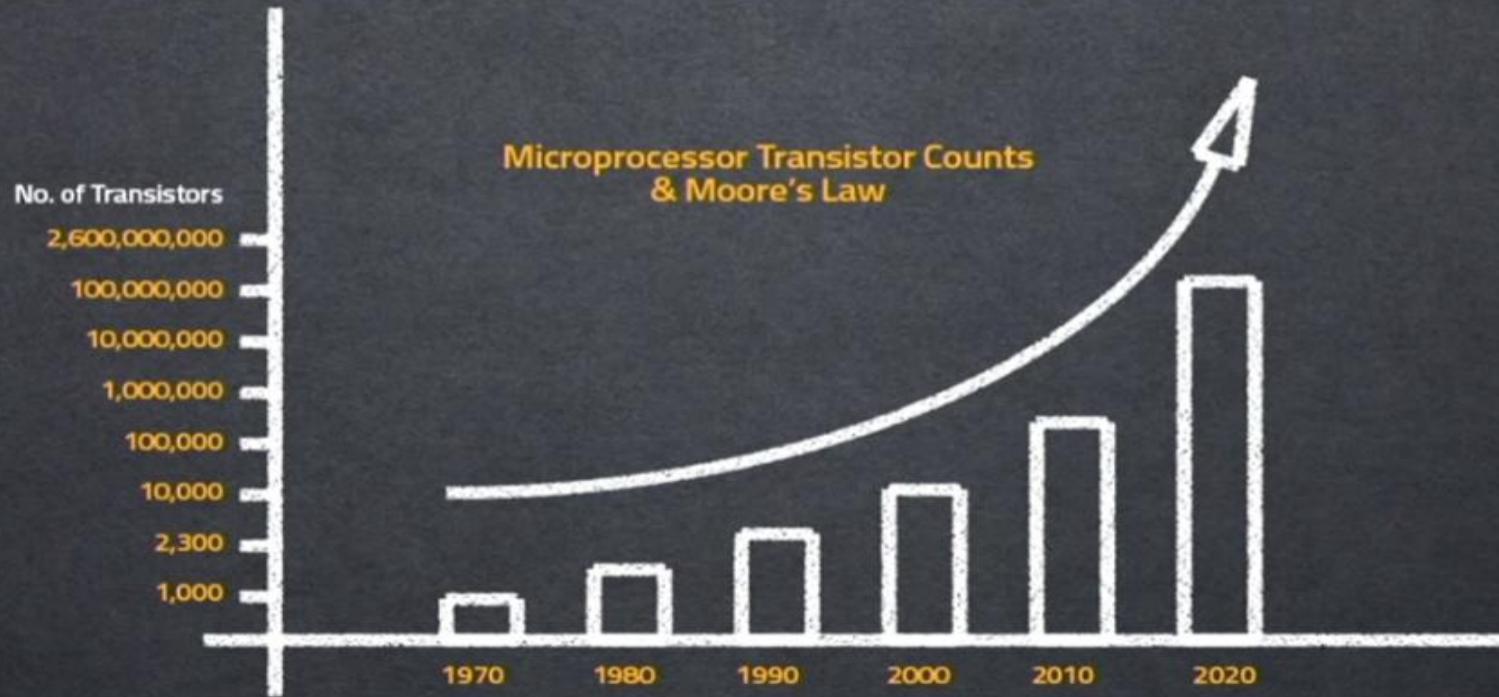


Let's have some predictions?

Some Facts Make Good Predictions

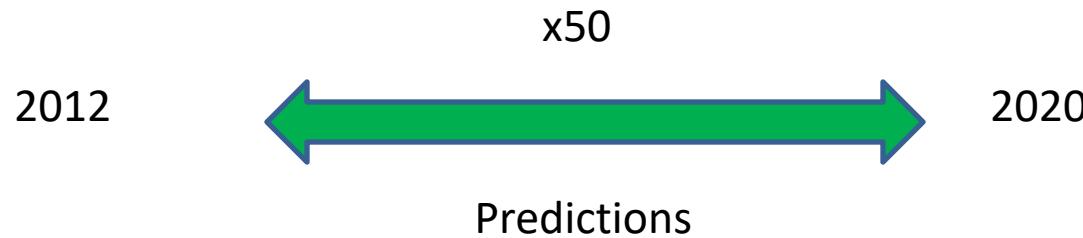


FACT: Hardware Gets Cheaper



Predictions

- As of 2009, the entire World Wide Web was estimated to contain close to 500 Exabyte's. This is a half Zeta byte
- The total amount of global data is grown to 2.7 Zeta bytes during 2012. This is 48% up from 2011



Challenges

Challenges are huge, Just take an example of internet:

- Per the survey: 250 Crores are using internet and the count will shoot every year.
- Every time you login to Facebook you create a log, if you login 5 times daily : 5 logs
- Similar for whatapp :20 Logs
- Similar for Gmail!
- Similar for Google!
- Similar for many products!
- And Data is valuable !!

Two Challenges we come across: How to store data and How to process Data?

Real world Examples

- Think about a farmer who crop rice on the fields?



- Think about government officer who signs files daily and regularly?



Where we can store data?



RDBMS

- High Concurrency
- TB Storage
- Indexed reads
- Efficient updates



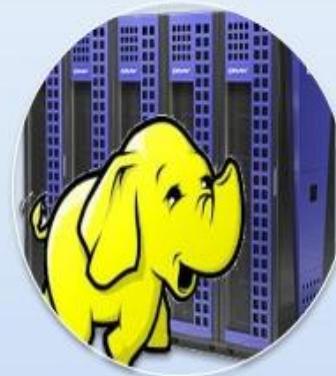
Analytic Appliances

- Scalable
- Medium Concurrency
- High Volume Processing (Postgres)



NoSQL

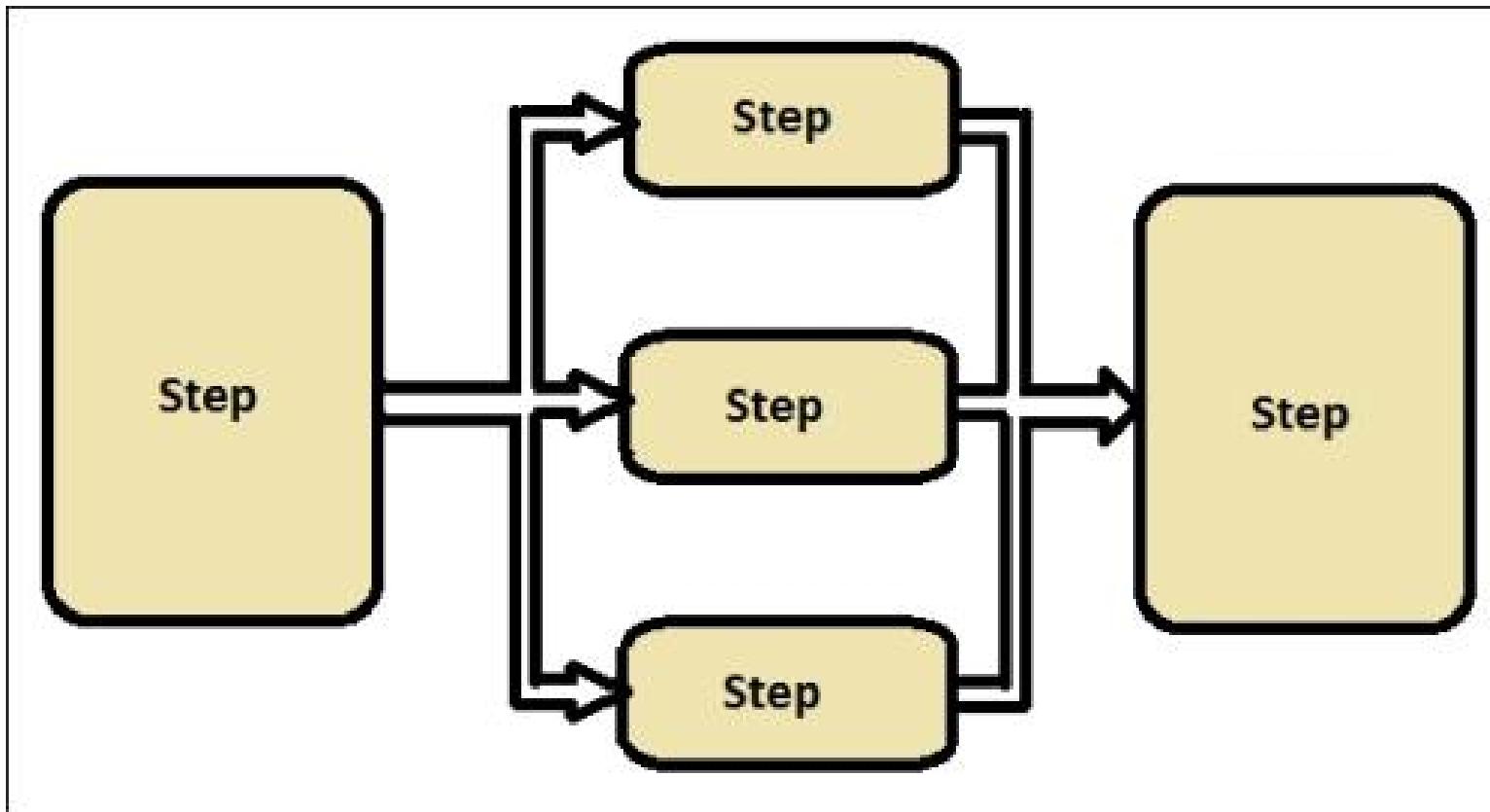
- Highly Scalable
- High Concurrency
- Storage Options
- Updates
- Real-time Capable
- Rudimentary



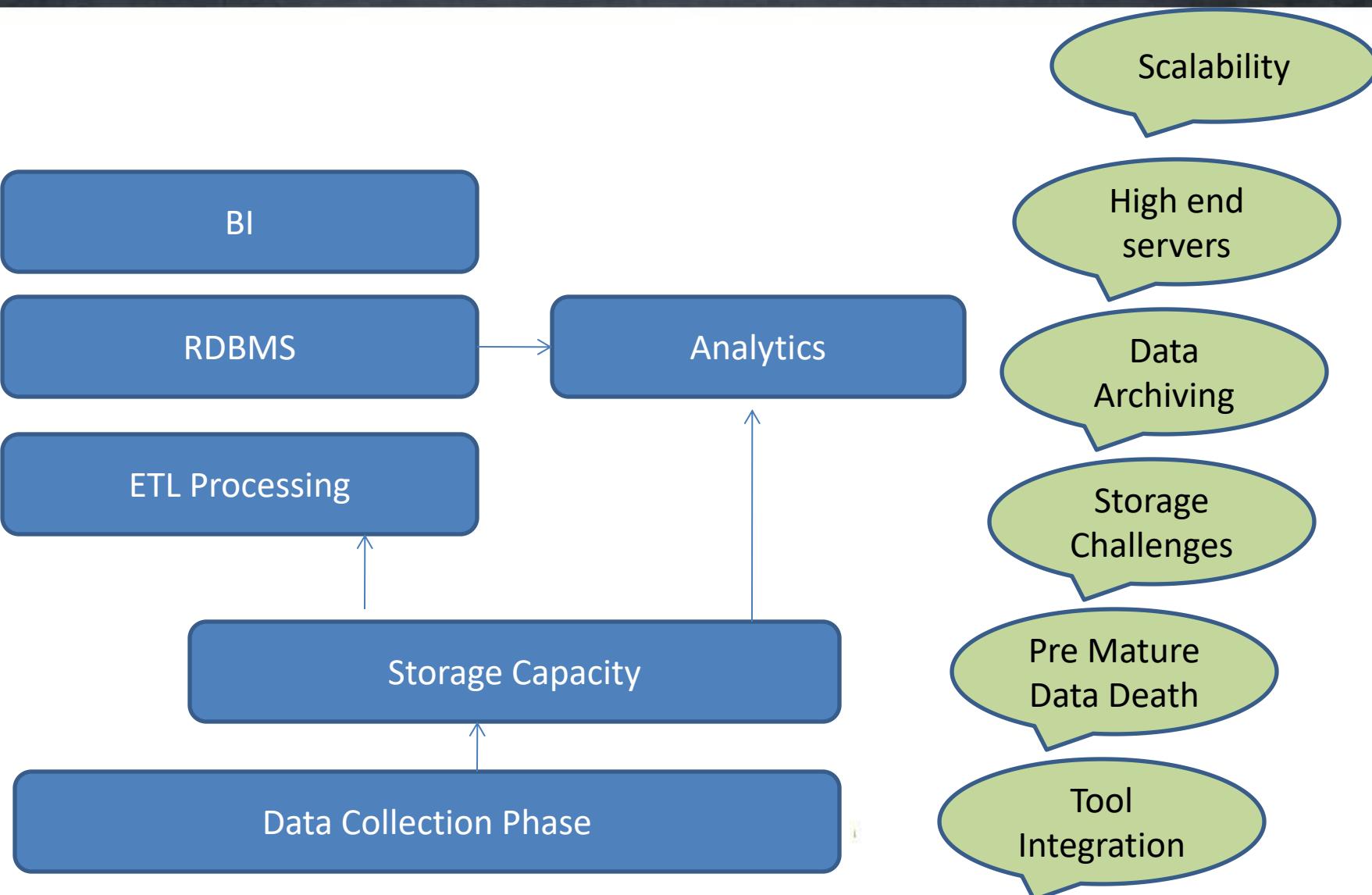
Hadoop

- Highly scalable
- Low concurrency
- Distributed Storage
- Complex Access
- Security (TBD)

How we process them?



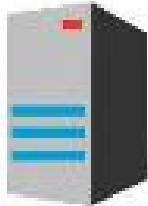
Challenges with the current infrastructure



Here is the conclusion.

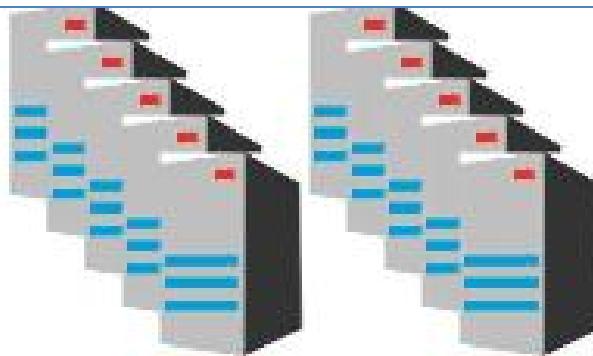
- We need storage space as much we can!
 - We need faster processing!
 - We need to process anything!
-
- Volume - To many bytes
 - Velocity - To high rate
 - Variety - To many sources

Why DFS?



1 Machine

4 I/O Channels
Each Channel – 100 MB/s

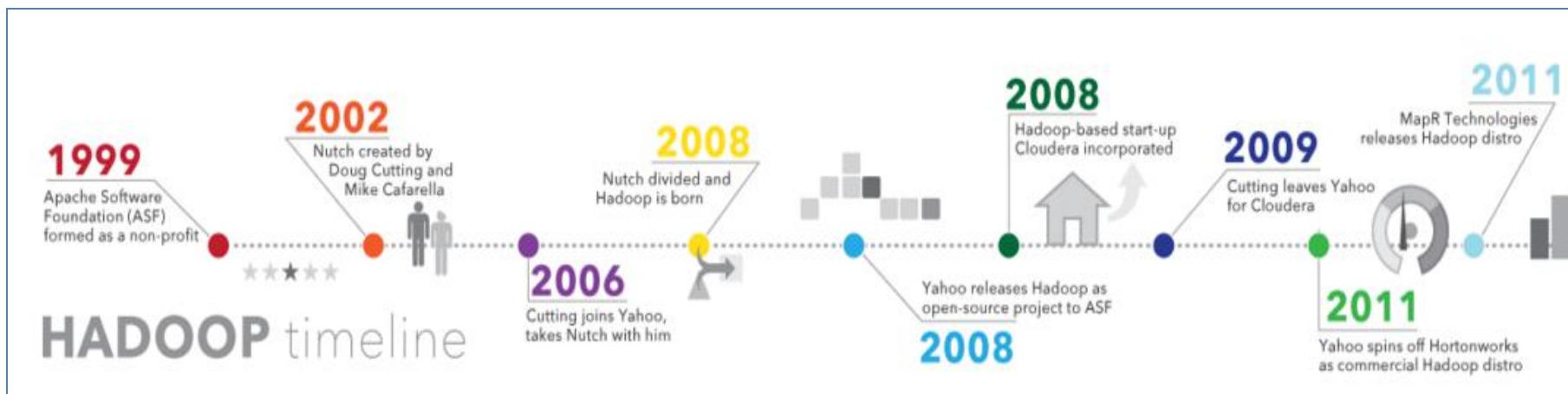


10 Machine

4 I/O Channels
Each Channel – 100 MB/s

What is Hadoop?

Per SAS - Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs



Why Hadoop? (As per SAS)

Ability to store and process huge amounts of any kind of data, quickly. With data volumes and varieties constantly increasing, especially from social media and the Internet of Things (IoT), that's a key consideration.

Computing power. Hadoop's distributed computing model processes big data fast. The more computing nodes you use, the more processing power you have.

Fault tolerance. Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.

Flexibility. Unlike traditional relational databases, you don't have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.

Low cost. The open-source framework is free and uses commodity hardware to store large quantities of data.

Scalability. You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

Cloudera

Cloudera distributes a platform of open-source projects called Cloudera's Distribution including Apache Hadoop or CDH.

Hortonworks

Major contributors to Apache Hadoop and dedicated to working with the community to make Apache Hadoop more robust and easier to install, manage, use, integrate and extend.

IBM InfoSphere BigInsights

brings the power of Apache Hadoop to the enterprise

MapR

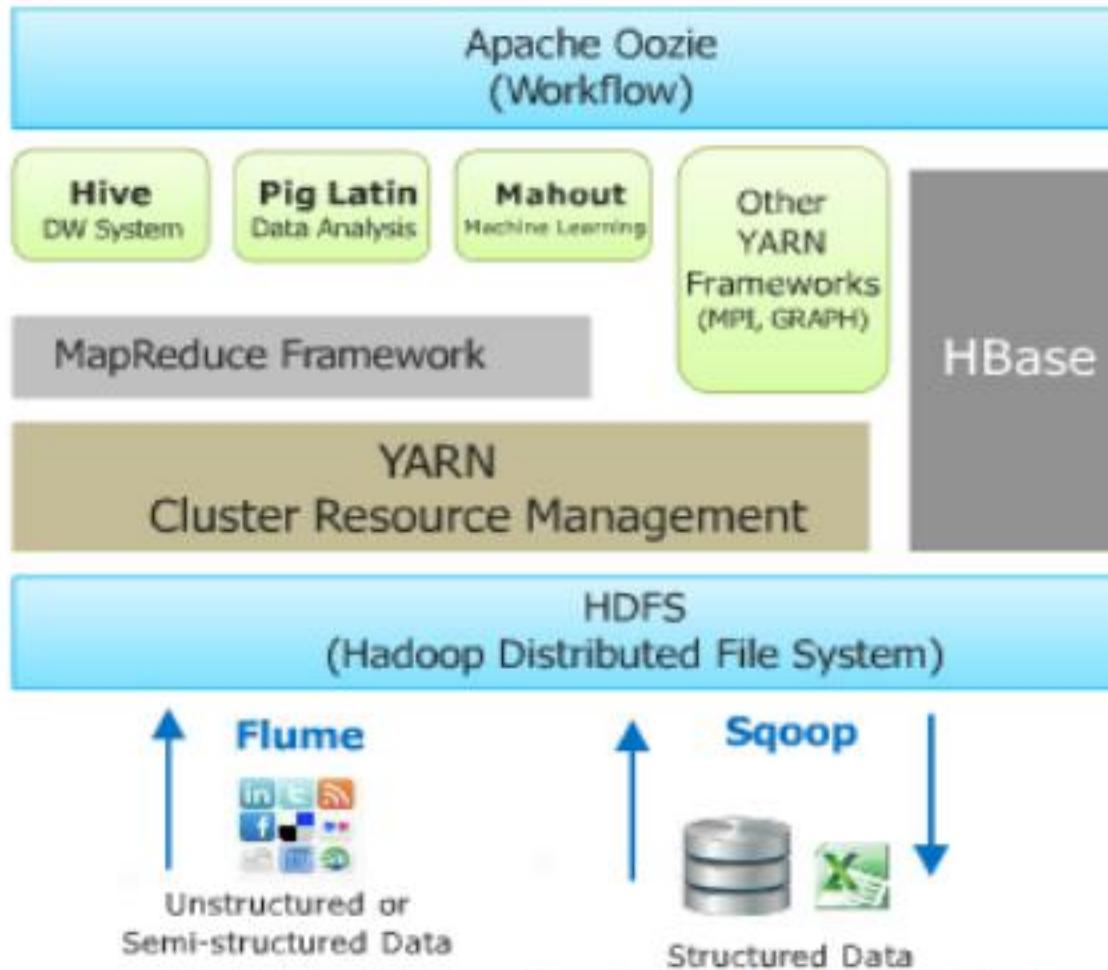
sells a high performance map-reduce framework based on Apache Hadoop that includes many of the standard eco-system components.

HDInsight

Managed Apache Hadoop, Spark, Hbase and Storm made easy

RDBMS		Hadoop
Structured	Data Types	Structure, Semi and Unstructured
Limited	Data Processing	Distributed processing
Standards	Governance	Loosely Standards
Required on write	Schema	Required on Read
Write many read many	Speed	Write once read many
Licence	cost	Open Source
OLTP ACID Transactions	Best Use	Data discovery Heavy Data processing Massive Storage/ Analytics

Hadoop Eco System







Sears (NASDAQ:SHLD) is an icon of the American business landscape. The company was founded by Richard Warren Sears and Alvah Curtis Roebuck in 1886. At its apogee, Sears was the biggest retailer in the United States. Due to its prosperity, the firm decided to build the largest building in America: the Sears Tower located in Chicago. At completion in 1973, it surpassed the World Trade Center towers and became the tallest building in the world. Now, Sears is not the firm that it once was. It is now the 13th largest retailer in the United States based on annual revenue behind Costco, Wal-Mart and Best buy for example.

Improving customer loyalty, and with it sales and profitability, is desperately important to Sears as it faces fierce competition from Wal-Mart and Target, as well as online retailers such as Amazon.com. While revenue at Sears has declined, from \$50 billion in 2008 to \$42 billion in 2011, big-box rivals Wal-Mart and Target have grown steadily, and they're far more profitable. Meantime, Amazon has gone from \$19 billion in revenue in 2008 to \$48 billion last year, passing Sears for the first time.



Enter Hadoop, an open source data processing platform gaining adoption on the strength of two promises: ultra-high scalability and low cost compared with conventional relational databases. Hadoop systems at 200 terabytes cost about one-third of 200-TB relational platforms, and the differential grows as scale increases into the petabytes, according to Sears. With Hadoop's massively parallel processing power, Sears sees little more than one minute's difference between processing 100 million records and 2 billion records.

Source: <http://www.informationweek.com/it-leadership/why-sears-is-going-all-in-on-hadoop/d/d-id/1107038>

US Presidential Elections – Problem Statement:

The 2000 and 2004 presidential elections in the United States were very close.

The largest percentage of the popular vote that any candidate received was 50.7% and the lowest was 47.9%. If a percentage of the voters were to have switched sides, the outcome of the elections would have been different. (Source: Machine learning in action)

There are small groups of voters who, when properly appealed to, will switch sides. These groups may not be huge, but with such close races, they may be big enough to change the outcome of the election.¹

How do you find these groups of people, and how do you appeal to them with a limited budget?

US 2012 Election



- Predictive modeling, clustering modeling and data mining for individual add and do the Analysis on the top of it.
- Hits on mybarackobama.com and identify the target audience.
- Drive traffic to other campaign sites
- Facebook page (33 million "likes")
- YouTube channel (240,000 subscribers and 246 million page views).
- A contest to dine with Sarah Jessica Parker
- Every single night, the team ran 66,000 computer simulations.
- Amazon web services

Job Trend from Indeed

Scale: Absolute - Relative



Introduction to HDFS

- Hadoop framework comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem.
- HDFS is a Hadoop Flagship filesystem
- It's a file system specially designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

Introduction to HDFS

- “Very large size” in this context means files of megabytes, gigabytes , terabytes or peta bytes in size.
- There are many Hadoop clusters store and processing peta bytes of data every day.
- As Moving Computation is Cheaper than Moving Data
- A computation requested by application is much more efficient if it is executed near the data it operates on. When the size of the data set is huge this minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running.

Introduction to HDFS

- HDFS is built on the principle of write-once, read-many-times pattern.
- A dataset generated or copied to the HDFS can be used to perform various analyses over time.
- Each time data analysis will involve a large proportion of dataset, sometimes only few proportions of it, so the time to read the whole dataset is more important than the latency in reading the first Record.

Commodity hardware

- Hadoop doesn't require expensive, highly reliable hardware.
- It's designed to run on clusters of commodity hardware for which the chance of node failure across the cluster is high.
- HDFS is designed to carry on working without a noticeable interruption to the user in the face of failure

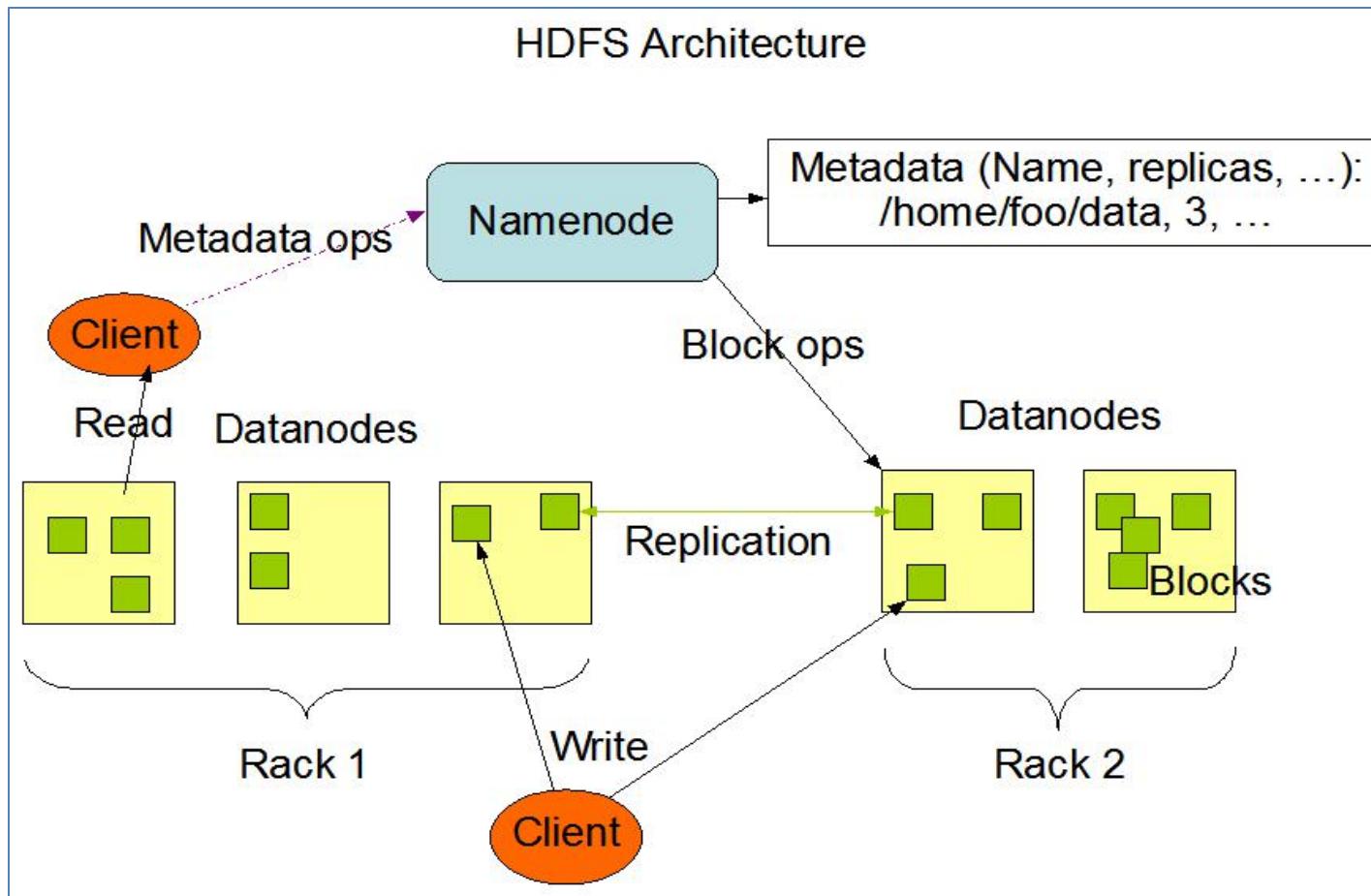
HDFS is not a good fit today:

- Low-latency data access
- Lots of small files

Why HDFS?

- Fault tolerance by detecting faults and applying quick, automatic recovery
- Data access via MapReduce streaming
- Simple and robust coherency model
- Processing logic close to the data, rather than the data close to the processing logic
- Portability across heterogeneous commodity hardware and operating systems
- Scalability to reliably store and process large amounts of data
- Economy by distributing data and processing across clusters of commodity personal computers
- Efficiency by distributing data and logic to process it in parallel on nodes where data is located
- Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures.

HDFS Architecture



Design of Name and Data Node

- HDFS has a master/slave architecture.
- HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients
- There are a number of DataNodes one per node which manage storage attached to the nodes that they run on
- A file is split into one or more blocks and these blocks are stored in a set of DataNodes

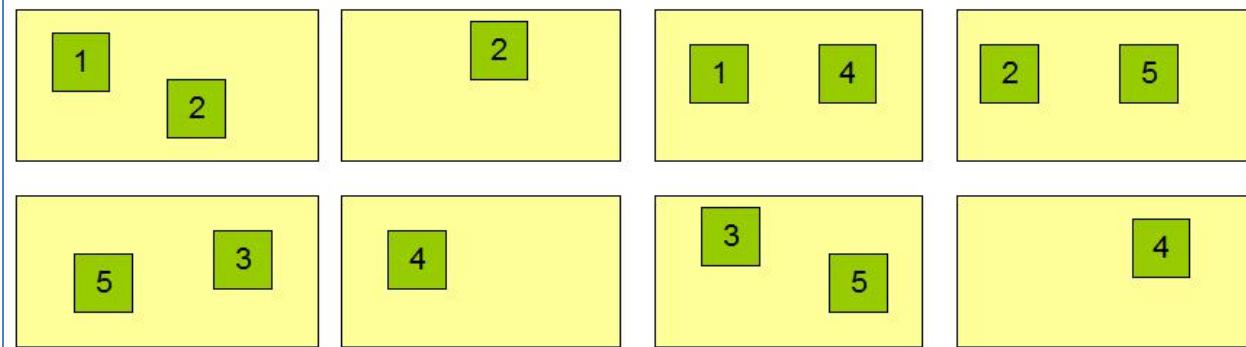
Replication Factor

- HDFS is designed to reliably store very large files across machines in a large cluster.
- It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
- The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
- The NameNode makes all decisions regarding replication of blocks.
- It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster

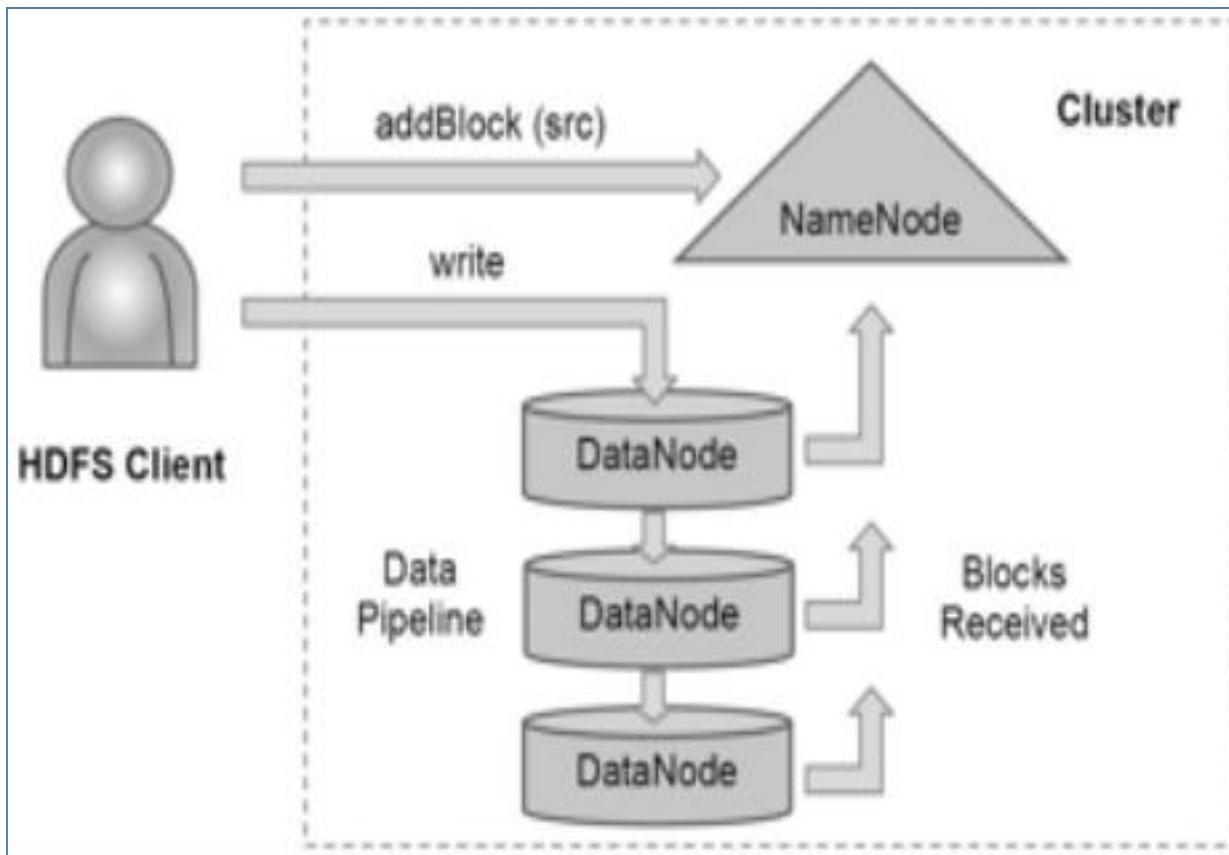
Block Replication

Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

Datanodes



Data Flow



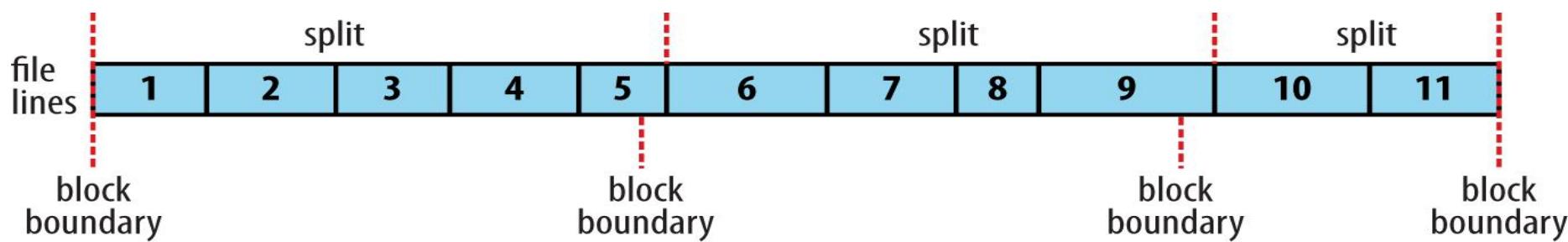
- Disk has a block size, which is the minimum amount of data that it can read or write.
- HDFS block is much larger unit—128 / 256 MB by default
- Files in HDFS are broken into block-sized chunks, which are stored as independent units
- File in HDFS that is smaller than a single block does not occupy a full block's storage
- HDFS blocks are large to minimize the cost of seeks
- If the block is large enough, the time it takes to transfer the data from the disk can be longer than the time to seek to the start of the block
- Ex. if the seek time is around 10 ms and the transfer rate is
- 100 MB/s, to make the seek time 1% of the transfer time, we need to make the block size around 100 MB. The default is actually 128 MB

- File can be larger than any single disk in the network
- Making the unit of abstraction a block rather than a file simplifies the storage subsystem eliminating metadata concerns
- Blocks fit well with replication for providing fault tolerance and availability.
- A block that is no longer available due to corruption or machine failure can be replicated from its alternative locations to other live machines to bring the replication factor back to the normal level
- HDFS's fsck command understands blocks
- % hdfs fsck / -files –blocks
- list the blocks that make up each file in the filesystem

- HDFS breaks down very large files into large blocks and stores three copies of these blocks on different nodes in the cluster.
 - Hadoop uses a logical representation of the data stored in file blocks, known as input splits
-
- The number of input splits that are calculated for a application determines the number of mapper tasks. Each of these mapper tasks is assigned to a slave node where the input split is stored.

Relation between HDFS and split

- Blocks are physical chunks of data store in disks where as InputSplit is not. It is a Java class with pointers to start and end locations in blocks
- A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information
- Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split



HDFS COMMANDS DEMO

- There are many ways to create a map reducer program in python, some of the frequent used module and ways are mentioned below:
- Each of the process have it own advantage and drawback, in this course we mainly focus on Streaming and Mrjob.

- Hadoop Streaming
- Mrjob
- dumbo
- hadoopy
- pydoop

- Hadoop Streaming is the way of supplying any executable to Hadoop as a mapper or a reducer, including standard Unix commands or Python scripts.
- The executable must read from stdin and write to stdout. One of the disadvantages of using Streaming directly is that while the inputs to the reducer are grouped by key, they are still iterated over line-by-line, and the boundaries between keys must be detected by the user.

Create Python Mapper

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    line = line.strip()
    keys = line.split()
    for key in keys:
        value = 1
        print( "%s\t%d" % (key, value) )
```

Create Python Reducer

```
#!/usr/bin/env python

import sys

last_key = None
running_total = 0
for input_line in sys.stdin:
    input_line = input_line.strip()
    this_key, value = input_line.split("\t", 1)
    value = int(value)

    if last_key == this_key:
        running_total += value
    else:
        if last_key:
            print( "%s\t%d" % (last_key, running_total) )
        running_total = value
        last_key = this_key

    if last_key == this_key:
        print( "%s\t%d" % (last_key, running_total) )
```

Step 1: Create Mapper and Reducer code

Step 2: Test it on local file system

```
cat words.txt | ./mapper.py | sort | ./reducer.py > output.txt
```

Step 3: Copy the files to HDFS

```
hadoop dfs -mkdir wordcount
```

```
hadoop dfs -copyFromLocal ./words.txt wordcount/words.txt
```

```
hadoop dfs -ls wordcount/
```

Step 4: Test the hadoop streaming jar

Steps involved in streaming

```
hadoop jar
  /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2- cdh3u0.jar \
  -Dmapred.reduce.tasks=1 \
  -input wordcount/words.txt \
  -output wordcount/output \
  -mapper cat \
  -reducer "wc -l"
```

Step 5:

```
hadoop jar /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2-
cdh3u0.jar \
-Dmapred.reduce.tasks=1 \
-mapper mapper.py \
-reducer reducer.py \
-input wordcount/words.txt \
-output wordcount/output \
-file mapper.py \
-file reducer.py
```

mrjob lets you write MapReduce jobs in Python 2.6+/3.3+ and run them on several platforms.

You can:

- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)
- Run in the cloud using Google Cloud Dataproc (Dataproc)

To Install MRjob

```
sudo pip install mrjob
```

```
sudo apt-get install python-mrjob
```

mrjob is the easiest route to writing Python programs that run on Hadoop. If you use mrjob, you'll be able to test your code locally without installing Hadoop or run it on a cluster of your choice.

Here are a number of features of mrjob that make writing MapReduce jobs easier:

- Keep all MapReduce code for one job in a single class
- Easily upload and install code and data dependencies at runtime
- Switch input and output formats with a single line of code
- Automatically download and parse error logs for Python tracebacks
- Put command line filters before or after your Python code

Rating Program

```
from mrjob.job import MRJob

class MRRatingCounter(MRJob):
    def mapper(self, key, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer(self, rating, occurrences):
        yield rating, sum(occurrences)

if __name__ == '__main__':
    MRRatingCounter.run()
```

Parallelism

- GIL
- Multithreading in Python
- Create Threads with parameters
- Create Daemon and Non Daemon processes
- Multiprocessing in Python

Multiple Thread Demo

```
import time

start = time.time()
def start_Single_thread(i):
    while i:
        i -= 1
    return

Total_iter = 50000000 # 50 million iteration
start_Single_thread(Total_iter)

end = time.time()
print "Single thread: "
print "Program took %.2f seconds to execute" %(end-start)
```

Multiple Thread Demo

```
import time, threading
start = time.time()
def start_Single_thread(i):
    while i:
        i -= 1
    return

Total_iter = 50000000 # 50 million iteration
t1 = threading.Thread(target=start_Single_thread,
                      args=(Total_iter/2,))
t2 = threading.Thread(target=start_Single_thread,
                      args=(Total_iter/2,))
#start the thread
t1.start(); t2.start()
# It will wait for main thread to wait
t1.join();t2.join()
end = time.time()
print "Multi thread: "
print "Program took %.2f seconds to execute" %(end-start)
```

Multiple Thread Demo

```
C:\Users\jatin\Desktop\Ethans\Python\I  
py  
Single thread:  
Program took 2.66 seconds to execute
```

```
C:\Users\jatin\Desktop\Ethans\Python\I  
py  
Multi thread:  
Program took 10.87 seconds to execute
```

Global Interpreter Lock

- Impossible to allow multiple native threads to execute code in parallel.
- It imposes various restrictions on threads
- Namely, you can't utilize multiple CPUs
- Exception: C extension like numpy can release GIL

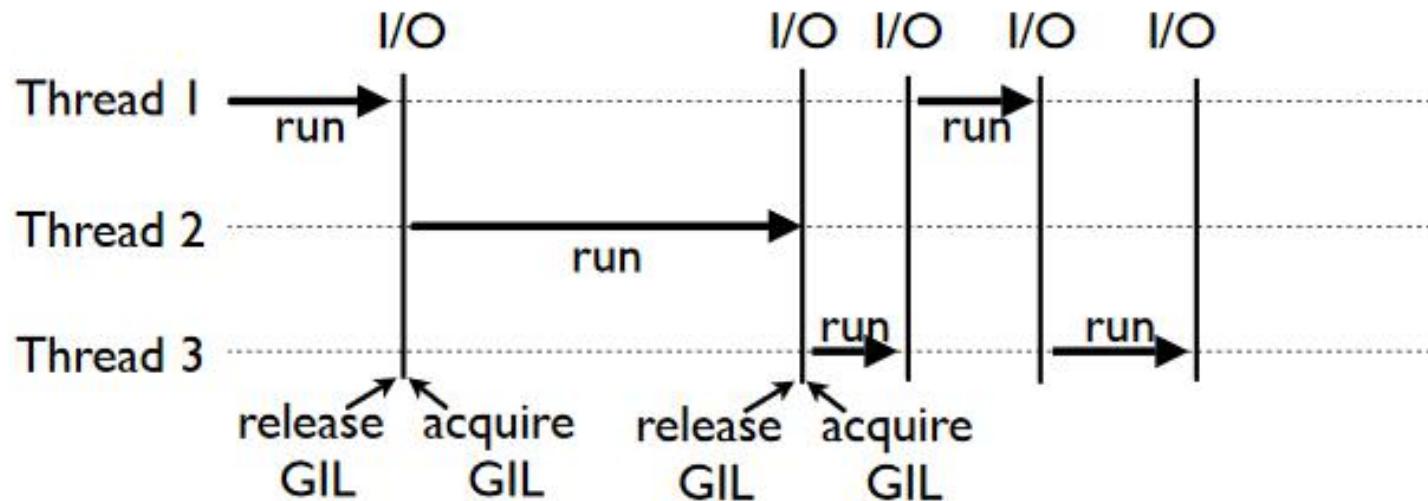
Applications written in programming languages with a GIL can be designed to use separate processes to achieve full parallelism, as each process has its own interpreter and in turn has its own GIL.

Benefits of the GIL

- Increased speed of single-threaded programs.
- integration of C libraries that usually are not thread-safe.

How Threads runs

- With the GIL, you get cooperative multitasking



- When a thread is running, it holds the GIL
- GIL released on I/O (read, write, send, recv, etc.)

Why/When to/not use threads?

- Not to use threads on CPU bound process
 - Not to use when application make use of multiple cores simultaneously for performance reasons
 - Not to use thread in distributed environment.
-
- Threads run in the same memory space (threads use the same memory, precautions have to be taken or two threads will write to the same memory at the same time. This is what the global interpreter lock is for)
 - Great option for I/O-bound applications
 - Great option of UI and non CPU bound processes.

- Module named threading is used to create threads.

They shared memory between processes

Each thread is running in parallel sequence.

There support GIL

- Thread function in threading module used to create thread.
- Target is the parameter in Thread function.

```
import threading

def start_thread():
    """thread function"""
    print 'Function for thread'
    return

threads = []
for i in range(5):
    t = threading.Thread(target=start_thread)
    threads.append(t)
    t.start()
```

Thread function

- Args is the parameter in thread function to pass the argument

```
import threading

def start_thread(arg):
    """thread function"""
    print 'Function for thread ', arg
    return

threads = []
for i in range(5):
    t = threading.Thread(target=start_thread, args=(i,))
    threads.append(t)
    t.start()
```

Get the Thread name

- Threading.currentThread().getName()

```
import threading
import time

def func1():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(2)
    print threading.currentThread().getName(), 'Exiting'

def func2():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(3)
    print threading.currentThread().getName(), 'Exiting'

t1 = threading.Thread(target=func1, name='MyThread 1')
t2 = threading.Thread(target=func2, name='MyThread 2')
t3 = threading.Thread(target=func2) # use default name

t1.start()
t2.start()
t3.start()
```

Use Logging Module

```
import logging
import threading
import time

logging.basicConfig(level=logging.DEBUG,
                    format='[%(levelname)s] %(threadName)-10s %(message)s',
                    )

def func1():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

def func2():
    logging.debug('Starting')
    time.sleep(3)
    logging.debug('Exiting')
```

Create Daemon Thread

```
def daemon_process():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon_process', target=daemon_process)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)
```

Wait for Daemon to complete

```
def daemon_process():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon_process', target=daemon_process)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()
d.join()
```

Multiprocessing

- Module named multiprocessing is used to create processes.

No shared memory between processes

Each process run 1 thread each

There is a support for managed shared data.

Multiprocessing Example – Using Pool

```
from multiprocessing import Pool
from time import sleep

def start(n):
    sleep(1)
    function_result = n * n
    return function_result

if __name__ == '__main__':
    pool = Pool(processes=5)          # Using Map function
    results = pool.map(start, range(20), chunkszie=10)
    print results
```

Multiprocessing Example – Using Process

```
from multiprocessing import Process
from os import getpid

def target_function():
    print getpid()

if __name__ == '__main__':
    p = Process(target=target_function, args=())
    p.start()
    p.join()
    p1 = Process(target=target_function, args=())
    p1.start()
    p1.join()
```

Process Queues

Effective use of multiple processes usually requires some communication between them, so that work can be divided and results can be aggregated.

Process Pipe

The two connection objects returned by `Pipe()` represent the two ends of the pipe. Each connection object has `send()` and `recv()` methods (among others).

Note that data in a pipe may become corrupted if two processes (or threads) try to read from or write to the *same* end of the pipe at the same time

Multiprocessing Communication

```
from multiprocessing import Process, Pipe
from os import getpid
from time import sleep

def target_function(proc, fmsg):
    count = 0
    while count < 10:
        msg = proc.recv()
        print "Process %r got message: %r" %(getpid(), msg)
        sleep(1)
        proc.send(fmsg)
        count += 1

if __name__ == '__main__':
    parent, child = Pipe()
    p = Process(target=target_function, args=(child, 'Message From child Process'))
    p.start()
    parent.send('Message from Parent')
    target_function(parent, 'Message from Parent')
    p.join()
```

Multiprocessing Communication

```
from multiprocessing import Process, Queue

def target_function(q):
    q.put(['First', 'Second'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=target_function, args=(q,))
    p.start()
    print q.get()
    p.join()
```

Synchronized processes

Lock on the process is used to ensure that only one process prints to standard output at a time:

```
from multiprocessing import Process, Lock

def func(l, i):
    l.acquire()
    print 'hello world', i
    l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        p = Process(target=func, args=(lock, num))
        p.start()
```

Synchronized processes

```
from multiprocessing import Process, Lock

def func(l, i):
    l.acquire()
    print 'hello world', i
    l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        p = Process(target=func, args=(lock, num))
        p.start()
        p.join()
```

It is possible to create shared objects using shared memory which can be inherited by child processes.

Data can be stored in a shared memory map using Value or Array

Shared Memory

```
from multiprocessing import Process, Value, Array
import ctypes

def func(n, list1):
    n.value = 'M'
    for i in range(len(list1)):
        list1[i] = -list1[i]

if __name__ == '__main__':
    num = Value('c', 't') # 'c' indicates a character
    arr = Array('i', range(10)) # 'i' indicates integer.

    p = Process(target=func, args=(num, arr))
    p.start()
    p.join()

    print num.value
    print arr[:]
```

Shared Memory

A manager object returned by Manager() controls a server process which holds Python objects and allows other processes to manipulate them using proxies.

A manager returned by Manager() will support types list, dict, Namespace, Lock Rlock etc

Shared Memory

```
from multiprocessing import Process, Manager

def func(details, managers):
    details['Age'] = 35
    details['Name'] = 'Jatin'
    details['Dept'] = 'IT'
    managers.reverse()

if __name__ == '__main__':
    manager = Manager()

    details = manager.dict()
    managers = manager.list(['Aakash', 'rahul', 'Akshay'])

    p = Process(target=func, args=(details, managers))
    p.start()
    p.join()

    print details
    print managers
```

Please NOTE:

This Tutorial is meant for year 2017-2018.

if YEAR > 2017:

Please contact Ethan's for new study material

THANK YOU