Python – Getting Started

https://www.python.org/about/

- Python is powerful... and fast;
  plays well with others;
  runs everywhere;
  is friendly & easy to learn;
  is Open.

- These are some of the reasons people who use Python would rather not use anything else.

Objective of this Module

1. Python language overview, +ves and –ves, Comparison with other languages
2. Popular Python distributions and IDEs like ANACONDA, iPython/ Jupyter, Spyder
3. Python Architecture
4. Python - basic commands, basic statistics
5. Functions
6. Conditional execution - loops
7. Object Oriented Programming
8. Introduction to NUMPY, PANDAS
9. Now ready to dive further …..

Python +ves

- Easy to learn: Easy learning curve makes it popular among managers and researchers
- General purpose: Can be used for all tasks including cloud, mobile, AI/ ML web development etc
- Very powerful: Although the core of the language is small and easy to learn, standard library Modules and Other packages (134969 on 11 April 2018, 173511 on 27 March 2019, 191372 on 10 August 2019) in Python Package Index **https://pypi.python.org/pypi** make it very powerful

Python +ves contd.

- Modern: Conceived by Guido van Rossum of CWI - National Research Institute for Mathematics and Computer Science, Netherlands, 1st version 1989, 2nd in 2000, constantly upgraded by a huge pool of contributors and the Python Software Foundation
- Open source: with a very large community of contributors. Completely free.
- Interpreter based: Not compiled like C or C++ or even Java (for Byte Codes)

Python -ves

- Interpreter based: Slower than C

- Not user friendly like Excel, SPSS or GRETL

- Requires programming

Comparison with R

- R another open source data analytics software is extremely good but it is not general purpose like Python

- AI and ML packages are more powerful in Python

- AI frameworks like Google’s Tensorflow, Keras, PyTorch and many others use Python

Comparison with MS Excel®

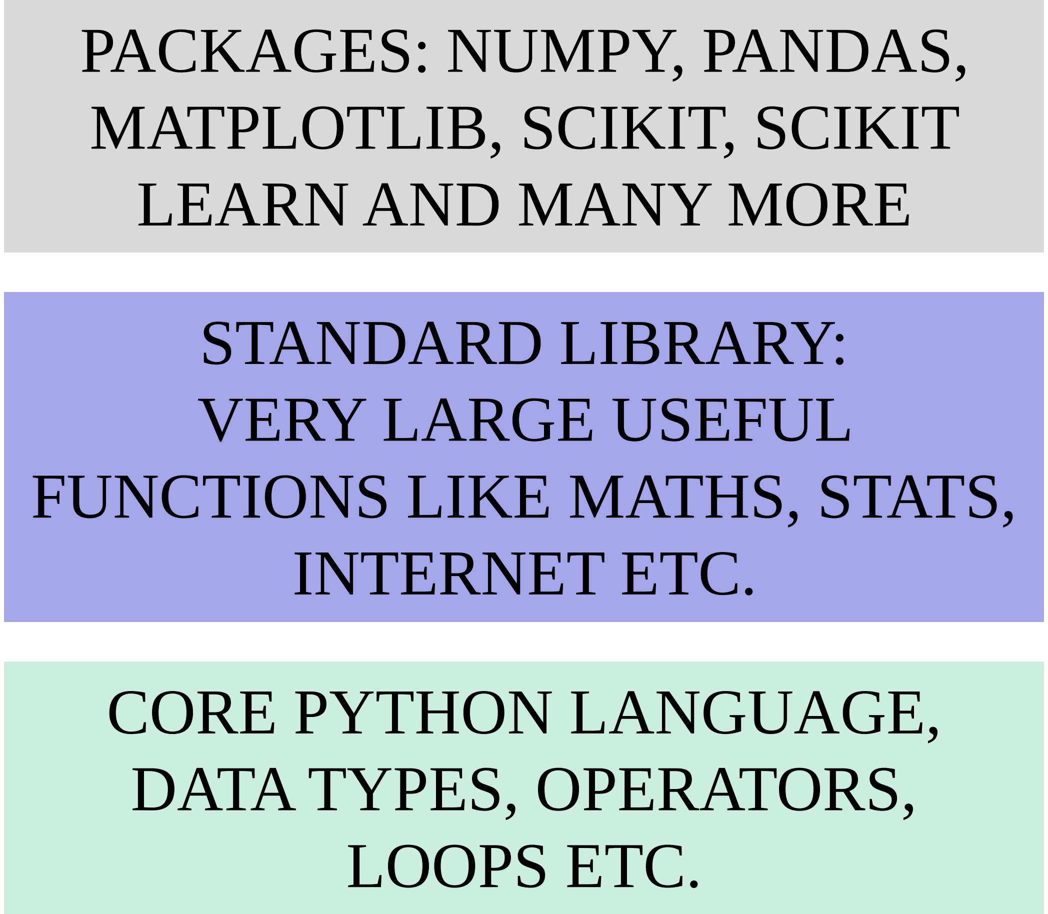
- MS Excel remains the first choice of managers
- Excel Pivot Tables, Solver and Data Analysis packs unrivalled in simplicity and power
- Falls short for advanced statistical analysis, ML, AI etc.
- Programming tool VBA limited to only MS packages
- Is not free and open source

Versions

- Currently in use:
  - Python 2.X.x
  - Python 3.X.x

- Latest 3.7.? (changes very fast!!)

Python Architecture



PACKAGES: NUMPY, PANDAS, MATPLOTLIB, SCIKIT, SCIKIT LEARN AND MANY MORE

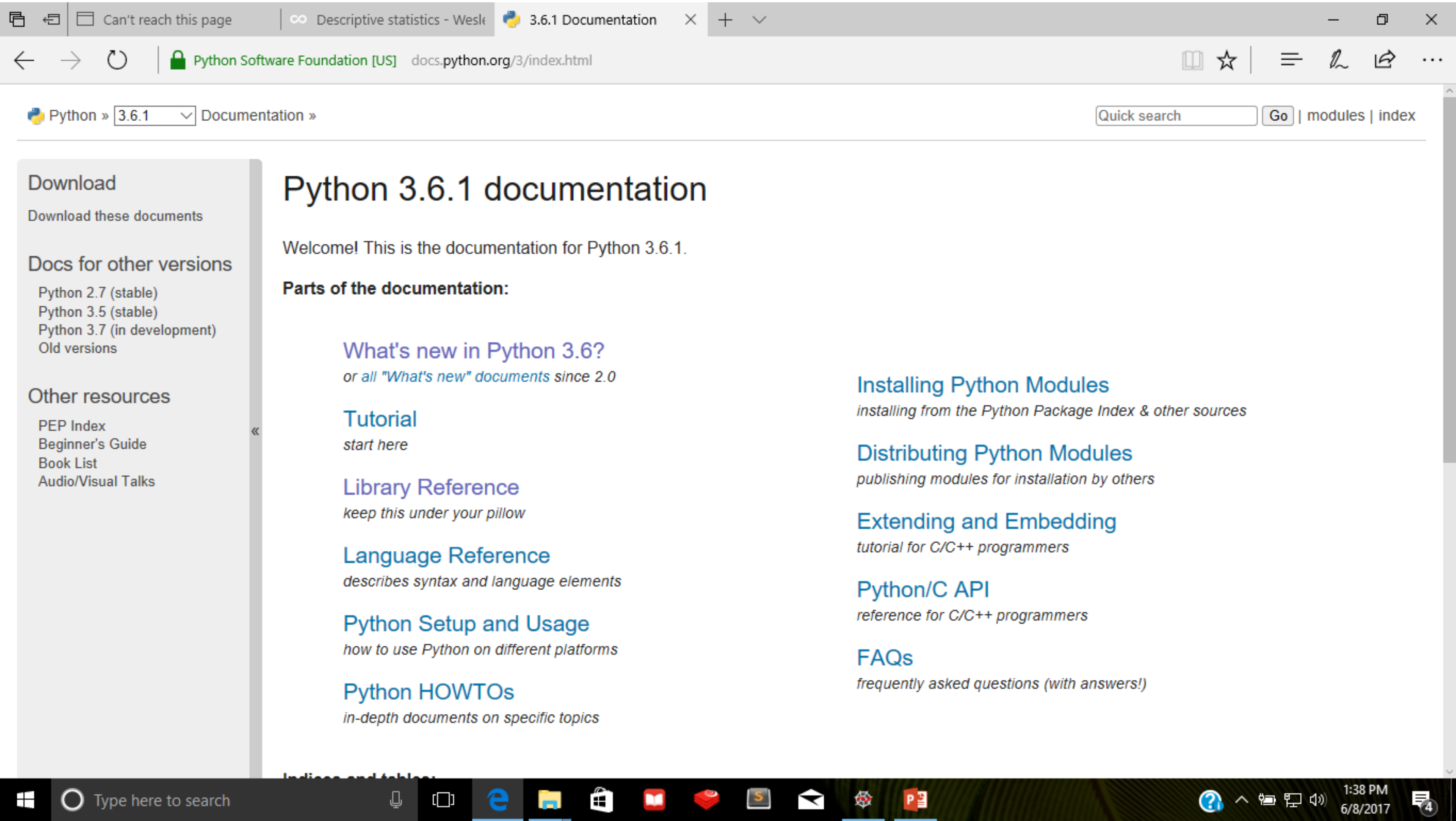
STANDARD LIBRARY:
VERY LARGE USEFUL FUNCTIONS LIKE MATHS, STATS, INTERNET ETC.

CORE PYTHON LANGUAGE, DATA TYPES, OPERATORS, LOOPS ETC.

Getting help

*https://**docs.python.org/3***

Python Documentation



Can't reach this page
Descriptive statistics - Wesle
3.6.1 Documentation
Python Software Foundation [US]
docs.python.org/3/index.html
Type here to search
1:38 PM
6/8/2017

Python » 3.6.1 Documentation »

Quick search Go | modules | index

Download

Download these documents

Docs for other versions

- Python 2.7 (stable)
- Python 3.5 (stable)
- Python 3.7 (in development)
- Old versions

Other resources

- PEP Index
- Beginner's Guide
- Book List
- Audio/Visual Talks

Python 3.6.1 documentation

Welcome! This is the documentation for Python 3.6.1.

Parts of the documentation:

What's new in Python 3.6?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

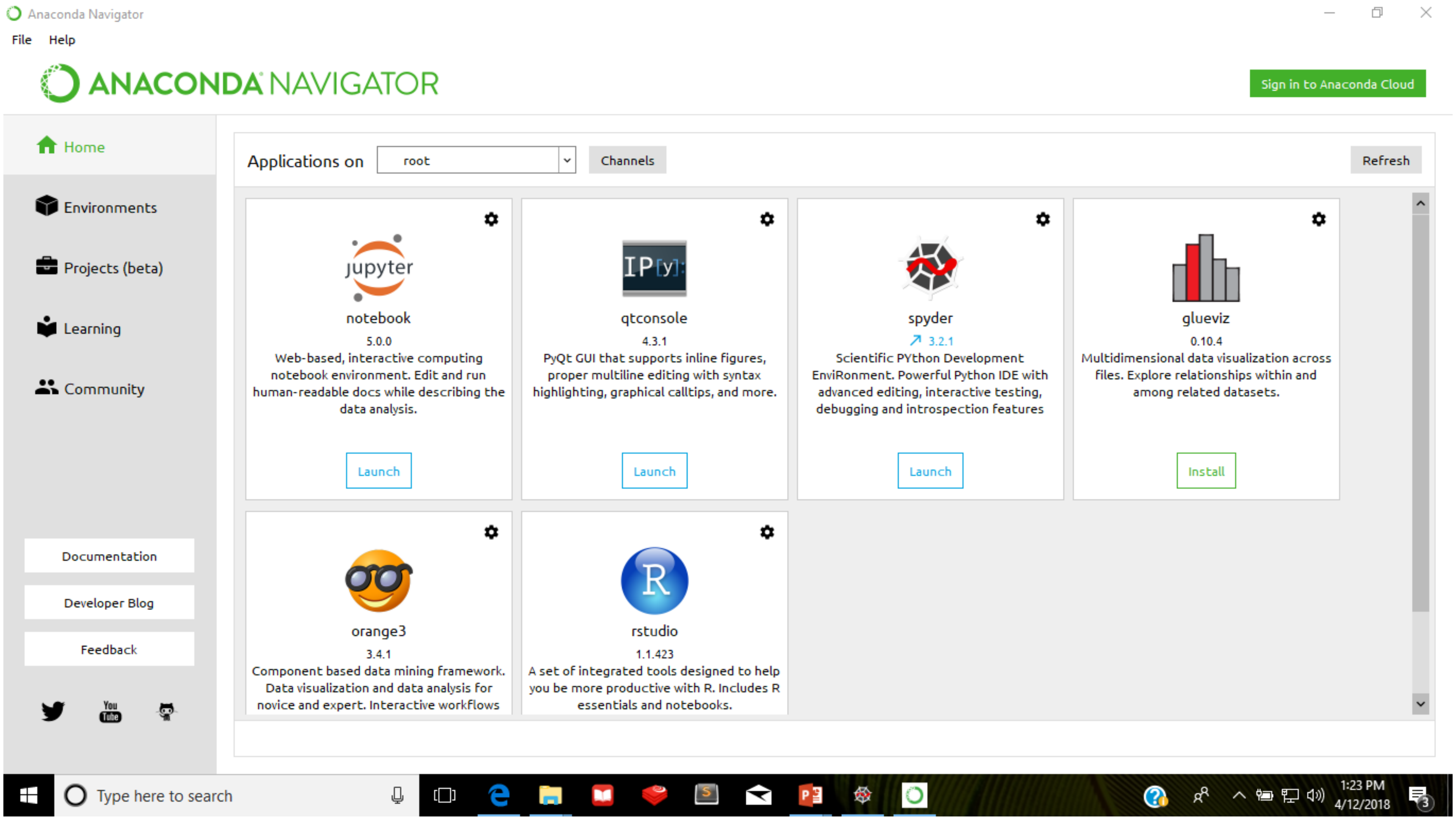
FAQs
frequently asked questions (with answers!)

Indices and tables:

Installing Python

- Python runs on many OSs including MS Windows® Unix (all variants) iOS Android and more
- Can be downloaded and installed from python.org but installing and configuring all the important packages is a challenge
- Popular distributions like iPython, Jupyter, Anaconda make it easier
- We will use Anaconda from anaconda.com and a very popular Python IDE called Spyder

Anaconda Navigator



notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder
↗ 3.2.1
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz
0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.

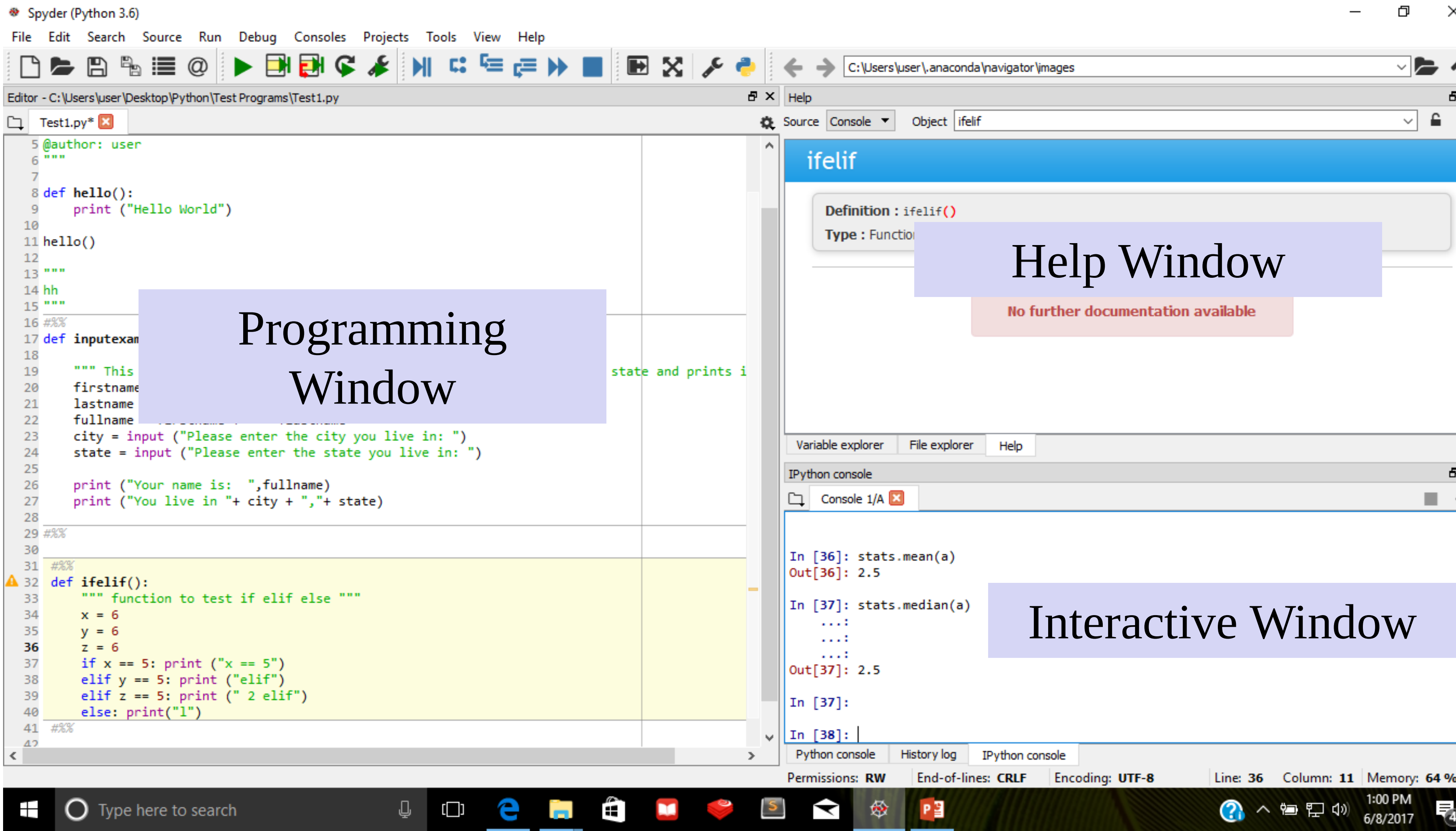
orange3
3.4.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

rstudio
1.1.423
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Let us get started …

- Click start and all programs
- Start Spyder (The Scientific Python Development EnviRonment) IDE (Integrated Development Environment)

SPYDER IDE



Programming Window

```
@author: user
"""

def hello():
    print ("Hello World")

hello()

"""
hh
"""
#%%
def inputexam

    """ This                                                  state and prints i
    firstname
    lastname
    fullname
    city = input ("Please enter the city you live in: ")
    state = input ("Please enter the state you live in: ")

    print ("Your name is:  ",fullname)
    print ("You live in "+ city + ","+ state)

#%%

 #%%
 def ifelif():
     """ function to test if elif else """
     x = 6
     y = 6
     z = 6
     if x == 5: print ("x == 5")
     elif y == 5: print ("elif")
     elif z == 5: print (" 2 elif")
     else: print("l")
 #%%
```

Help Window

ifelif

Definition : ifelif()

Type : Functio

No further documentation available

Interactive Window

```
In [36]: stats.mean(a)
Out[36]: 2.5

In [37]: stats.median(a)
    ...:
    ...:
    ...:
Out[37]: 2.5

In [37]:

In [38]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **36** Column: **11** Memory: **64 %**

Hello …

In Programming pane type #%% to open a new cell

```
name = 'JD’
print("Hello",name)
Hello JD
```

Execute by CTRL - Enter

Operators: Exercise

```
>>> 2 - 2
0
>>> 2/2
1.0
>>> 2 * 2
4
>>> 2 ** 2
4
>>> 2 * -2
-4
>>> 2 ** -2
0.25
```

Exercise

- 13/3
- Out[73]: 4.333333333333333

- 13//3
- Out[74]: 4 Quotient

- 13%3
- Out[75]: 1

Comments

- Comments are not executed and are used for our own reference. Start with #

```
>>> 25 / 100 # comment is not executed
0.25
>>>
>>> 25/ 100 without hash it is an error
SyntaxError: invalid syntax
>>>
```

Opening new cell in editor pane

- Type #%%
- A bar appears above
- Each cell is independent
- To execute use Ctrl – Enter or Shift – Enter
- You can save programs you write using editor icons

Exercise

- Using Python, and ONLY the features taught so far - find area of a circle if the radius is 237.5 cms

Try ..

```
>>> 3.14 * 237.5 **2
177115.625
>>> 3.14 * (237.5 **2)
177115.625
>>>
```

USING BRACKETS IS SAFER

Round() power ()

```
>>> 1/3 # Python gives answer upto 16 places
0.3333333333333333
>>> round (1/3, 3) # if you want 3 digit precision
0.333
>>> 2 ** 6 # 2 to the power 6
64
>>> pow(2,6) # power with two arguments
64
```

Variables, Assignment

```
a = 2

b = 3

c = a + b

c
Out[5]: 5
```

input

```
#%%
# This program demos input function

name = input ("What is your name?  ")
print ("Hello class. My name is", name)
```

String, Int and Float

```
a= "jd"

type(a)
Out[13]: str
```

```
b = 3

type (b)
Out[15]: int
```

```
c = 13.9

type(c)
Out[17]:
float
```

Type Conversion

```
 d = "30.5"

type(d)
Out[19]: str

e = float (d)

type(e)
Out[21]: float
```

Exercise

- Write a program to compute volume of a cylinder of base with diameter d and height h
- Accept d and h values using input
- Assume pi value is 3.14

```
#%%
#Compute volume of a cylinder of diameter d and height h
d = input ("Enter diameter of the cylinder: ") # d is a string
h = input ("Enter height of the cylinder:  ") # h is a string
r = float(d)/2 #r is floating number
floath = float(h) # converted h into floating number
v = 3.14*(r**2)*floath
print ("Volume of cylinder is:  ", v)
```

Exercise

- Compute FV of Rs. 100 after 5 years if interest rate is 5 % payable every annum. Try first in interactive, and then in a program

```
>>> PV = 100 # Amount invested today i.e. present value
>>> r = .05 # 5 % or 5/100 interest rate per annum
>>> t = 5  # invested for 5 years
>>> FV = PV * ((1 + r) ** t) # formula for compound interest
>>> FV
127.62815625000003
>>>
```

Python Standard Library

- Built into Python
- See https://docs.python.org/3/library/index.html#library-index
- Very extensive list but there are uncountable packages contributed by many
- Let us see few examples from Numerical and Mathematical Modules

Numerical and Mathematical Modules

- numbers, math, cmath (Complex Numbers), decimals, fractions, random, statistics

Import math

```
>>> import math as m
>>> dir(m)
['__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> m.e
2.718281828459045
```

Exercise

```
>>> help(m.log)
Help on built-in function log in module math:
log(...)
    log(x[, base])
    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
>>> m.log(100,10) # Log of 100 with base 10
2.0
>>> m.log (4,2)
2.0
>>> m.log (7.39)  # no base is mentioned so automatically becomes e
2.0001277349601105
```

More Examples

```
m.log(100,10)
Out[17]: 2.0

m.log(100, m.e)
Out[18]: 4.605170185988092

m.log(100)
Out[19]: 4.605170185988092
```

Exercise

- Try cylinder volume using math pi

Basic Stats

- import statistics as s
- dir(s) # see basic statistics functions available
- A = [1,2,3,4,5,6,7,8,9]
- s.mean(A)
- 5

Let us see NUMPY PACKAGE

NUMPY.ORG
FV/ PV Functions

```
import numpy as np
dir(np)
```

```
help(np.fv)
Help on function fv in module
numpy.lib.financial:

fv(rate, nper, pmt, pv, when='end')
    Compute the future value.
```

Example

```
np.fv(0.05/12, 10*12, -100, -100)
    15692.928894335748
```

By convention, the negative sign represents cash flow out (i.e. money not available today). Thus, saving $100 a month at 5% annual interest leads to $15,692.93 available to spend in 10 years.

Example PV

```
np.pv(.05,10,0,10000,0)
Out[13]: -6139.1325354075916
```

Function: Celsius to Fahrenheit Converter

On Spyder, open a new file and save as CtoF.py

```
#%%
#CtoF.py
print ("Celsius to Fahrenheit Converter")
c = input ("Type the Centigrade temperature you want to convert: ")
f = (int(c)* (9/5) + 32)
print ("Celsius ",c," degrees Centigrade is equal to: ",f," degrees Fahrenheit")
# dummy = input ("Type any key to exit")
```

Test and Exercise

- Run the program using ctrl-enter
- Write FtoC.py and test
- Write a program to convert inches to centimeters

Functions

```
#%%
#CtoF.py as a Function

def c2f(c):
    print ("Celsius to Fahrenheit Converter")
    f = (c* (9/5) + 32)
    print ("Celsius ",c," degrees Centigrade is equal to: ",f," degrees Fahrenheit")

c2f(20)
# dummy = input ("Type any key to exit")
```

Try c2f(40) and many more values in interactive panel

Exercise

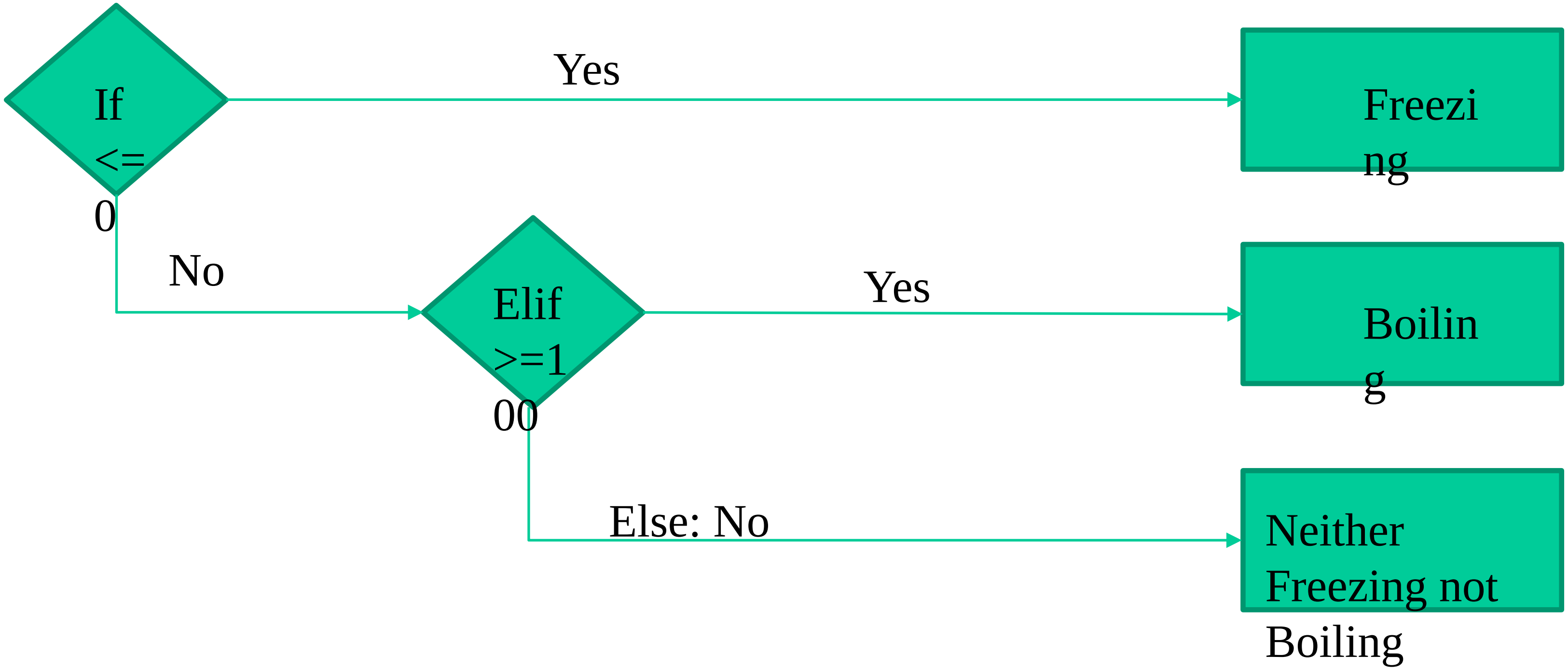
- Write function f2c(f)
- Write functions for centimeter to inches conversion; and vice versa i.e. c2i(c) and i2c(i)

Conditional Execution & Flow Charts

- If, elif, else

```
#%%
# If, elif, else

def check_temp (c):
    if c<= 0: print ("Freezing cold")
    elif c >=100: print ("Boiling hot")
    elif c == 25: print (“Comfortable”)
    else: print("Neither freezing nor boiling")
```

- Run once using ctrl-enter
- Interactive window try check-temp with values 900, -900 and 90



If <= 0

Yes

Freezing

No

Elif >=100

Yes

Boiling

Else: No

Neither Freezing not Boiling

While

```
•  #%%
•   # while loop demo
•  def while_demo(count):
•      loop = 1
•      while loop <= count:
•          print ("loop  :", loop)
•          loop = loop + 1
•
•      print ("That is all folks")
```

Try while_demo with different counts like 5, 10 etc. interactively
Create Flowchart

for

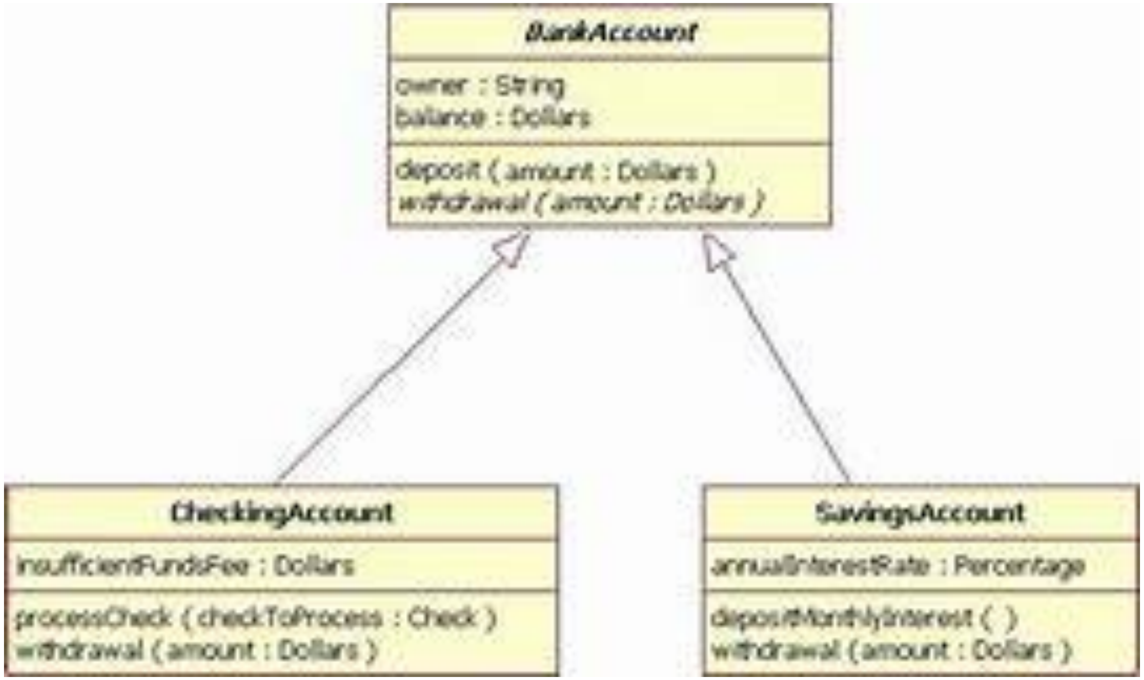
```
#%%
# for loop demo

def for_demo(count):
    for count in range (0,count):
        print("loop number:  ", count)

print("Bye")
```

- Try for_demo(5) and see the differences in loop count. See for other counts too

Object oriented systems modeling

- Objects have methods (actions) and Data
- Objects are instances of classes – like student class, dog class, college class etc.
- Encapsulates functionality, and can inherit from Parent Class i.e. Alsatian, Doberman from Dog class
- Let us see few examples next

Class Diagram (http://www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_6/uml/miniweb/images/class_box_inheritance.jpg)


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Bank Accounts

```
#%%
# Bank Account
class BankAccount:
    def __init__(self):
            self.balance = 0
    def deposit(self, amount):
            self.balance = self.balance + amount
            print ("Deposited ", amount)
    def withdraw(self, amount):
            self.balance = self.balance - amount
            print ("Withdrew ", amount)
    def getBalance(self):
            print ("New Balance", self.balance)
```

Test

```
jd = BankAccount()
print ("Initial Balance for JD",jd.balance)
jd.deposit(500)
jd.getBalance()
jd.withdraw(100)
jd.getBalance()
```

Inheritance

```
class SavingsBankAccount(BankAccount):
    def payInterest(self, interest):
        self.balance = self.balance + interest
        self.getBalance()
```

Test

```
jd = BankAccount()
jd.deposit(500)
jd.getBalance()
jd.withdraw(100)
jd.getBalance()
md = SavingsBankAccount()
md.getBalance()
md.payInterest(500)
```

Exercise

- Write a program to transfer money from one account to another
- Hint:
  - Create two accounts
  - Withdraw an amount from one account and deposit same amount to the other account

Book Store

```
class BookStore:
    noOfBooks = 0

    def __init__(self, title, author):
        self.title = title
        self.author = author
        BookStore.noOfBooks += 1

    def bookInfo(self):
        print("Book title:", self.title)
        print("Book author:", self.author,"\n")
```

Test

```
# Create a virtual book store
b1 = BookStore("Great Expectations", "Charles Dickens")
b2 = BookStore("War and Peace", "Leo Tolstoy")
b3 = BookStore("Middlemarch", "George Eliot")

# call member functions for each object
b1.bookInfo()
b2.bookInfo()
b3.bookInfo()

print("BookStore.noOfBooks:", BookStore.noOfBooks)
```

Exercise

- In BookStore, add Book Price too.

Exercise

- Create class Dog that can take breed and color as data, and bark as method
- Create inherited class with polymorphic bark as bite
- Example next slide

```
class Dog:
    def __init__(self,breed, color):
        self.breed = breed
        self.color = color

    def bark(self):
        print ("woof from", self.color,self.breed)

class BitingDog(Dog):
    def bark(self):
        print ("I only bite!")
```

Test

- D1 = Dog(“Alasatian”, “White”)
- D2 = Dog(“Doberman”, “Black”)
- D3 = BitingDog(“Wolf”, “Brown”)
- D1.bark()
- D2.bark()
- D3.bark()

More Packages

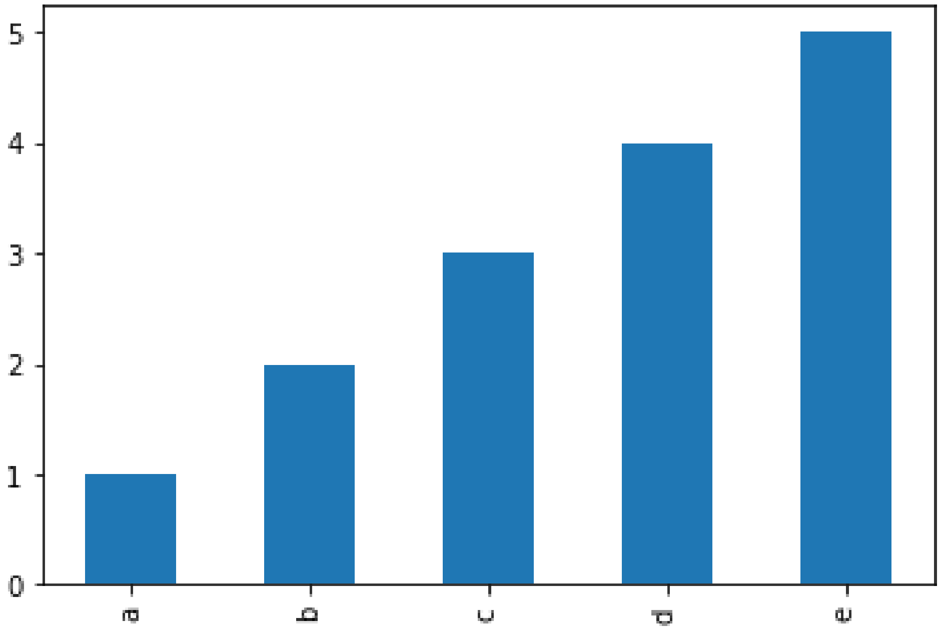
- Few important packages are
  - MATPLOTLIB for graphs and Plots
  - PANDAS for arrays, data-frames, panel data
  - SCIKIT-LEARN for AI and ML
- And many more depending upon your application area!

PANDAS Series

```
Import pandas as pd
A = pd.Series([1,2,3,4,5],index =
['a','b','c','d','e'])

A
Out[20]:
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Pandas Plot

```
A.plot.bar()
Out[21]:
<matplotlib.axes._subplots.AxesSubplot at 0x157601857b8>
```

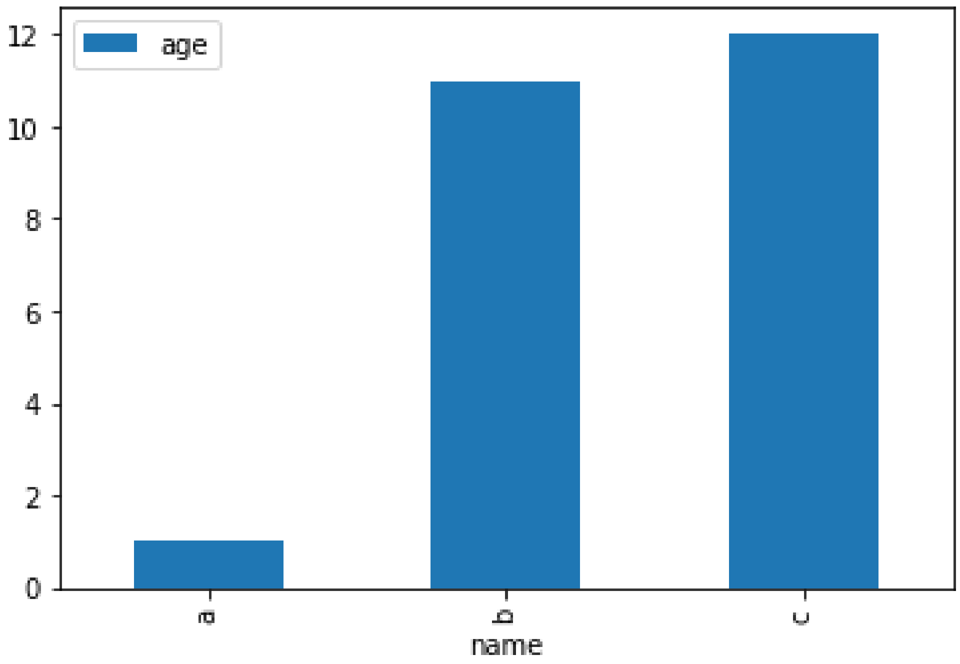


Pandas DataFrame

```
B = {'name':['a','b','c'],'age':[1,11,12]}

B
Out[11]: {'age': [1, 11, 12], 'name': ['a', 'b', 'c']}

DFB=pd.DataFrame(B)

DFB
Out[13]: 
   age name
0    1    a
1   11    b
2   12    c
```

Plot

```
DFB.plot.bar(x='name',y='age')
Out[17]: <matplotlib.axes._subplots.AxesSubplot at
0x1576000c2e8>
```



Example np and pd: series

```
import numpy as np
import pandas as pd

s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd',
'e'])
s
Out[10]:
a   -1.075724
b   -0.669115
c    0.779878
d    0.166397
e   -0.491294
dtype: float64
```

describe()

```
s.describe()
Out[29]:
count    5.000000
mean    -0.257972
std      0.733045
min     -1.075724
25%     -0.669115
50%     -0.491294
75%      0.166397
max      0.779878
dtype: float64
```

Dataframe

```
frame =
pd.DataFrame(np.random.randn(10
00, 5), columns=['a', 'b', 'c', 'd','e'])

frame
Out[20]:
           a         b         c         d         e
0    0.534041 -0.300464 -0.126352
-0.194686 -1.728177
1    1.450127 -1.041184  0.328707
0.901436  0.227058
```

head()

```
frame.head()
Out[21]:
          a         b         c         d         e
0  0.534041 -0.300464 -0.126352 -0.194686 -1.728177
1  1.450127 -1.041184  0.328707  0.901436  0.227058
2  1.169889 -0.395007 -2.031040 -0.739766  0.002057
3 -0.562570  0.935417 -0.650353 -2.773218 -0.003864
4 -2.498796 -0.917083 -1.210438 -0.027769  0.461034
```

describe()

```
frame.describe()
Out[23]:
              a           b           c           d           e
count      1000        1000        1000        1000        1000
mean  -0.004761    0.008685   -0.015524   -0.027963   -0.030574
std    0.998305    1.004691    1.051539    0.977183    0.991057
min   -2.996762   -3.023147   -3.256911   -2.773218   -3.156850
25%   -0.677937   -0.682510   -0.745531   -0.646041   -0.703703
50%   -0.010786   -0.026258   -0.025656   -0.025700   -0.021817
75%    0.676731    0.695667    0.693414    0.636796    0.657460
max    3.618155    2.899134    3.030679    2.898862    2.860351
```

corr()

```
frame.corr()
Out[26]:
          a         b         c         d         e
a  1.000000 -0.004181 -0.048300  0.041204 -0.030811
b -0.004181  1.000000  0.020562 -0.010158 -0.047979
c -0.048300  0.020562  1.000000  0.005389 -0.005460
d  0.041204 -0.010158  0.005389  1.000000  0.019832
e -0.030811 -0.047979 -0.005460  0.019832  1.000000
```

cov()

```
frame.cov()
Out[27]:
          a         b         c         d         e
a  0.996612 -0.004194 -0.050703  0.040195 -0.030484
b -0.004194  1.009403  0.021723 -0.009973 -0.047773
c -0.050703  0.021723  1.105734  0.005537 -0.005690
d  0.040195 -0.009973  0.005537  0.954886  0.019206
e -0.030484 -0.047773 -0.005690  0.019206  0.982193
```

Reading Excel Files

Python

Excel

a	b	c
1	7	20
2	8	30
3	9	40
4	6	50
5	5	60

Excel

- Create the file ideally starting row and columns 1, sheet 1
- Save at a folder of your choice and give it a name
- For this example I have used Desktop and given name test.xlsx

Python

- Start Spyder
- On RHS top corner click on the file icon to open navigation window and navigate to the folder where you stored the file
- In my case I navigate to Desktop
- Open a new cell and enter code as shown, and hit ctrl-enter to execute the cell
- a DataFrame named df is created

```
#%%
# Reading xlsx file
import pandas as pd
df = pd.read_excel('test.xlsx')
print(df)
```

```
   Unnamed: 0  a  b   c
0         NaN  1  7  20
1         NaN  2  8  30
2         NaN  3  9  40
3         NaN  4  6  50
4         NaN  5  5  60
```

Post-retirement Wealth Data

N	**# of years (X)**	**Lakhs in Bank (Y)**
1	0	45
2	5	42
3	10	33
4	15	31
5	20	29

Python Example

```
import pandas as pd
import statsmodels.api as sm
dict = {'Years': [0,5,10,15,20], 'Balance': [45,42,33,31,29]} # Data dictionary
df = pd.DataFrame(data=dict)
y=df['Balance']
x=df['Years']
x=sm.add_constant(x)
results=sm.OLS(y,x).fit()
print(results.summary())
```

Results

```
OLS Regression Results
==============================================================================
Dep. Variable:                Balance   R-squared:                       0.925
Model:                            OLS   Adj. R-squared:                  0.899
Method:                 Least Squares   F-statistic:                     36.74
Date:                Fri, 06 Jul 2018   Prob (F-statistic):            0.00901
Time:                        14:49:45   Log-Likelihood:                -9.8578
No. Observations:                   5   AIC:                             23.72
Df Residuals:                       3   BIC:                             22.93
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         44.6000      1.738     25.664      0.000      39.069      50.131
Years         -0.8600      0.142     -6.061      0.009      -1.312      -0.408
==============================================================================
```

Analysis

- R Squared is .925 close to 1, very high – so model has good predictive power
- Constant b0 is 44.6
- Co.eff b1 is – 0.86
- Regression Equation is

Balance = 44.6 – 0.86 * Age

Question: After 25 years, what would be the Balance?

Answer

- Bank balance = 44.6 + (- .86 * Years)
- Balance at 25 = 44.6 + (- .86*25) = 23.1

… to summarize

1. Python language overview, +ves and –ves, Comparison with other languages
2. Popular Python distributions and IDEs like ANACONDA, iPython/ Jupyter, Spyder
3. Python Architecture
4. Python - basic commands, basic statistics
5. Functions
6. Conditional execution - loops
7. Object Oriented Programming
8. Introduction to NUMPY, PANDAS
9. Now ready to dive further …..

END MODULE