Module V

Data Processing

The os and sys modules

NumPy

Basics, Creating arrays, Arithmetic, Slicing, Matrix Operations, Random numbers

Plotting and visualization

Matplotlib

Basic plot

Ticks

Labels

Legends

Working with CSV files

Pandas

Reading, Manipulating, and Processing Data

Introduction to Micro services using Flask.

The os and sys modules

The os module is a part of the standard library in Python 3.

Must import in the for accessing the class/methods .

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

The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

All functions in this module raise OSError (or subclasses thereof) in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

import os

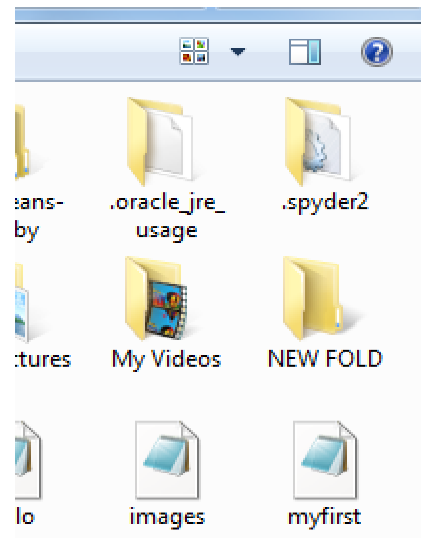
Get your current working directory

```
curDir = os.getcwd()
print(curDir)
```

```
>>> import os
>>> os.getcwd()
'C:\\Users\\sreeraj'
>>>
```

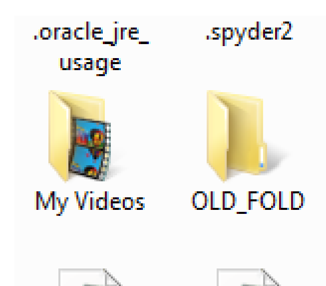
To make a new directory:

```
>>> os.mkdir("NEW FOLD")
```



To change the name of, or rename, a directory:

```
>>> os.rename("NEW FOLD","OLD_FOLD")
```



Changing the Current Working Directory

Using the chdir() function.

```
>>> import os
>>> os.chdir("C:\MyPythonProject")
>>> os.getcwd()
'C:\MyPythonProject'
```

Change CWD to Parent

```
>>> os.chdir("C:\\MyPythonProject")
>>> os.getcwd()
'C:\\MyPythonProject'
>>> os.chdir("..")
>>> os.getcwd()
'C:\\'
```

Removing a Directory

The rmdir() function in the OS module removes the specified directory either with an absolute or relative path.

For a directory to be removed, it should be empty.

```
>>> import os
>>> os.getcwd()
'C:\\MyPythonProject'
>>> os.rmdir("C:\\MyPythonProject")
PermissionError: [WinError 32] The process cannot access the file because it is
>>> os.chdir("..")
>>> os.rmdir("MyPythonProject")
```

os.remove() method in Python is used to remove or delete a file

List Files and Sub-directories

```
>>> import os
>>> os.listdir("c:\python37")
['DLLs', 'Doc', 'fantasy-1.py', 'fantasy.db', 'fantasy.py', 'frame.py',
'gridexample.py', 'include', 'Lib', 'libs', 'LICENSE.txt', 'listbox.py', 'NEWS.
'place.py', 'players.db', 'python.exe', 'python3.dll', 'python36.dll', 'pythonw
'sclst.py', 'Scripts', 'tcl', 'test.py', 'Tools', 'tooltip.py', 'vcruntime140.d
'virat.jpg', 'virat.py']
```

The listdir() function returns the list of all files and directories in the specified directory.

os.name: This function gives the name of the operating system dependent module imported.

The following names have currently been registered: ‘posix’, ‘nt’, ‘os2’, ‘ce’, ‘java’ and ‘riscos’

os.error: All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

os.error is an alias for built-in OSError exception.

sys module

The sys module contains variables and functions that pertain to the operation of the interpreter and its environment.

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

Like all the other modules, the sys module has to be imported with the import statement, i.e.

import sys

The current version number of Python can be accessed using,

>>> sys.version

```
>>> sys.version
'3.7.0 (default, Aug 14 2018, 19:12:50) [MSC v.1900 32 bit (Intel)]'
>>>
```

Command-line arguments

Lots of scripts need access to the arguments passed to the script, when the script was started sys.argv returns a list, which contains the command-line arguments passed to the script.

```
import sys

# list of arguments:
print(sys.argv)

# or it can be iterated via a for loop:

for i in range(len(sys.argv)):
    if i == 0:
        print("Function name: ", sys.argv[0])
    else:
        print(f"{i:1d}. argument: {sys.argv[i]}")
```

```
(base) C:\Users\sreeraj>python D:\Programs\python\arglist.py a c v
Function name:  D:\Programs\python\arglist.py
1. argument: a
2. argument: c
3. argument: v
```

- len(sys.argv) provides the number of command-line arguments.
- sys.argv[0] is the name of the current Python script

sys.exit([arg]) can be used to exit the program.

The optional argument arg can be an integer giving the exit or another type of object.

If it is an integer, **zero is considered "successful termination".**

```
# Python program to demonstrate
# sys.exit()
import sys
age = int(input("Enter age"))
if age < 18:
  # exits the program

  sys.exit()
else:
  print("Age is not less than 18")
```

Input and Output using sys

The sys modules provide variables for better control over input or output.

- stdin
- stdout
- stderr

stdin: It can be used to get input from the command line directly. It is used for standard input. It internally calls the input() method. It, also, automatically adds '\n' after each sentence.

```
import sys


for line in sys.stdin:
  if 'q' == line.rstrip():
    break
  print(f'Input : {line}')

print("Exit")
```

```
==================
hello
Input : hello

q
Exit
>>> |
```

stdout: A built-in file object that is analogous to the interpreter's standard output stream in Python.

stdout is used to display output directly to the screen console.

Output can be of any form, it can be output from a print statement, an expression statement, and even a prompt direct for input.

By default, streams are in text mode. In fact, wherever a print function is called within the code, it is first written to sys.stdout and then finally on to the screen.

```
>>> sys.stdout.write("SRS")
SRS3
```

Stderr: Whenever an exception occurs in Python it is written to sys.stderr.

```
import sys


def print_to_stderr(a):

  # Here a is the array holding the objects
  # passed as the argument of the function
  print(a, file = sys.stderr)

print_to_stderr("Hello World")
```

sys.path

Is a built-in variable within the sys module that returns the list of directories that the interpreter will search for the required module.

When a module is imported within a Python file, the interpreter first searches for the specified module among its built-in modules.

If not found it looks through the list of directories defined by **sys.path**.

```
>>> import sys
>>> sys.path
['', 'C:\\Users\\sreeraj\\Anaconda3\\python37.zip', 'C:\\Users\\sreeraj\\Anacond
a3\\DLLs', 'C:\\Users\\sreeraj\\Anaconda3\\lib', 'C:\\Users\\sreeraj\\Anaconda3'
, 'C:\\Users\\sreeraj\\Anaconda3\\lib\\site-packages', 'C:\\Users\\sreeraj\\Anac
onda3\\lib\\site-packages\\win32', 'C:\\Users\\sreeraj\\Anaconda3\\lib\\site-pac
kages\\win32\\lib', 'C:\\Users\\sreeraj\\Anaconda3\\lib\\site-packages\\Pythonwi
n']
>>> _
```

NumPy Basics

NumPy, short for Numerical Python.

Fundamental package required for high performance scientific computing and data analysis.

ndarray, a fast and space-efficient multidimensional array providing vectorized arithmetic operations.

Standard mathematical functions for fast operations on entire arrays of data without having to write loops

Tools for reading / writing array data to disk and working with memory-mapped Files

Linear algebra, random number generation, and Fourier transform capabilities Tools for integrating code written in C, C++, and Fortran

The NumPy ndarray: A Multidimensional Array Object

Key feature of NumPy - N-dimensional array object, or ndarray.

Fast, flexible container for large data sets in Python.

Arrays permits mathematical operations on whole blocks of data using similar syntax to the equivalent operations between scalar elements.

```
In [8]: data
Out[8]:
array([[ 0.9526, -0.246 , -0.8856],
       [ 0.5639,  0.2379,  0.9104]])

In [9]: data * 10
Out[9]:
array([[ 9.5256, -2.4601, -8.8565],
       [ 5.6385,  2.3794,  9.104 ]])
```

```
In [10]: data + data
Out[10]:
array([[ 1.9051, -0.492 , -1.7713],
       [ 1.1277,  0.4759,  1.8208]])
```

An ndarray is a generic multidimensional container for homogeneous data.

All of the elements must be the same type.

Every array has a shape, a tuple indicating the size of each dimension, and a dtype, an object describing the *data type of the array.*

```
In [11]: data.shape
Out[11]: (2, 3)
```

```
In [12]: data.dtype
Out[12]: dtype('float64')
```

Creating ndarrays

pip install numpy

import numpy as np

Use the array function.

This accepts any sequence- like object (including other arrays) and produces a new NumPy array containing the passed data.

```
>>> data1 = [6, 7.5, 8, 0, 1]
>>> arr1 = np.array(data1)
>>> arr1
array([6. , 7.5, 8. , 0. , 1. ])
>>>
```

```
>>> import numpy as np
>>> data=[[1,2,3,4],[5,6,7,8]]
>>> arr2d=np.array(data)
>>> arr2d
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> arr2d[0][0]
1
>>> arr2d.size
8
>>> arr2d.shape
(2, 4)
>>> arr2d.dtype
dtype('int32')
>>> 
```

empty() creates an array without initializing its values to any particular value and return garbage values.

```
>>> np.zeros(10)
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
>>> np.zeros((3, 6))
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
>>> np.empty((2, 3, 2))
array([[[1.05449727e-304, 1.78021527e-306],
        [8.45549797e-307, 1.37962049e-306],
        [1.11260619e-306, 1.78010255e-306]],

       [[9.79054228e-307, 4.45057637e-308],
        [8.45596650e-307, 9.34602321e-307],
        [4.94065646e-322, 2.11610174e-307]]])
>>> |
```

arrange() is an array-valued version of the built-in Python range function:

```
>>> np.arange(15)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> |
```

Function	Description
array	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype. Copies the input data by default.
asarray	Convert input to ndarray, but do not copy if the input is already an ndarray
arange	Like the built-in range but returns an ndarray instead of a list.
ones, ones_like	Produce an array of all 1's with the given shape and dtype. ones_like takes another array and produces a ones array of the same shape and dtype.
zeros, zeros_like	Like ones and ones_like but producing arrays of 0's instead
empty, empty_like	Create new arrays by allocating new memory, but do not populate with any values like ones and zeros
eye, identity	Create a square N x N identity matrix (1's on the diagonal and 0's elsewhere)

```
>>> np.ones((3, 6))
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
>>> np.ones_like((2,3))
array([1, 1])
>>> np.eye(2)
array([[1., 0.],
       [0., 1.]])
>>> np.identity(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> 
```

Data Types for ndarrays

The data type or dtype is a special object containing the information the ndarray needs to interpret a chunk of memory as a particular type of data:

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)

arr1.dtype
dtype('float64')

arr2.dtype
dtype('int32')
```

Type	Description
`int8, uint8`	Signed and unsigned 8-bit (1 byte) integer types
`int16, uint16`	Signed and unsigned 16-bit integer types
`int32, uint32`	Signed and unsigned 32-bit integer types
`int64, uint64`	Signed and unsigned 32-bit integer types
`float16`	Half-precision floating point
`float32`	Standard single-precision floating point. Compatible with C float
`float64, float128`	Standard double-precision floating point. Compatible with C double and Python `float` object
`float128`	Extended-precision floating point
`complex64, complex128, complex256`	Complex numbers represented by two 32, 64, or 128 floats, respectively
`bool`	Boolean type storing `True` and `False` values
`object`	Python object type
`string_`	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use `'S10'`.
`unicode_`	Fixed-length unicode type (number of bytes platform specific). Same specification semantics as `string_` (e.g. `'U10'`).

convert or cast an array from one dtype to another using ndarray’s **astype** method:

```
arr = np.array([1, 2, 3, 4, 5])

arr.dtype
dtype('int64')

float_arr = arr.astype(np.float64)

float_arr.dtype
dtype('float64')
```

```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
```

Operations between Arrays and Scalars

Arrays express batch operations on data without writing any for loops.

This is usually called vectorization.

Any arithmetic operations between equal-size arrays applies the operation element wise.

```
>>> arr = np.array([[1., 2., 3.], [4., 5., 6.]])
>>> arr
array([[1., 2., 3.],
       [4., 5., 6.]])
>>> arr*arr
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
>>> arr+arr
array([[ 2.,  4.,  6.],
       [ 8., 10., 12.]])
>>> arr-arr
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> 1/arr
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
>>>
```

Basic Indexing and Slicing

NumPy array indexing is a way to select a subset of data or individual elements.

One-dimensional arrays are similar to Python lists:

```
arr = np.arange(10)

arr
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

arr[5]
5

arr[5:8]
array([5, 6, 7])

arr[5:8] = 12

arr
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

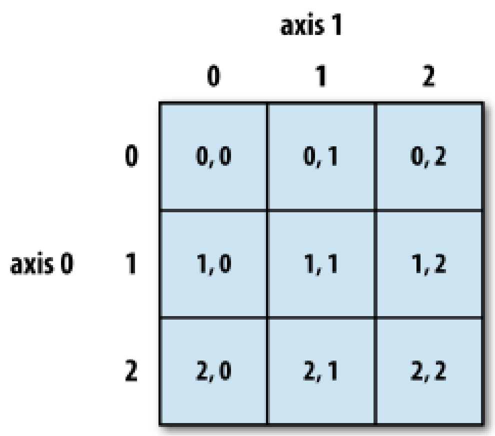
```
>>> arr=np.arange(15)
>>> arr
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> arr[5:7]
array([5, 6])
>>> arr[5:6]=[12]
>>> arr
array([ 0,  1,  2,  3,  4, 12,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> 
```

```
>>> arr_slice=arr[5:8]
>>> arr_slice
array([12,  6,  7])
>>> 
```

Multidimensional arrays

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

arr2d[2]
array([7, 8, 9])
```

```
arr2d[0][2]
3
```



```
>>> import numpy as np
>>> arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
>>> arr3d[0]
array([[1, 2, 3],
       [4, 5, 6]])
>>> old_values = arr3d[0].copy()
>>> old_values
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr3d[0] = 42
>>> arr3d
array([[[42, 42, 42],
        [42, 42, 42]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> arr3d[0] = old_values
>>> arr3d
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

Both scalar values and arrays can be assigned to arr3d[0]:

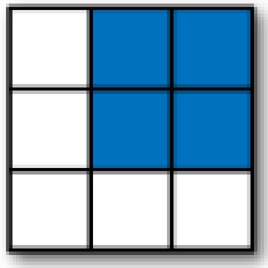
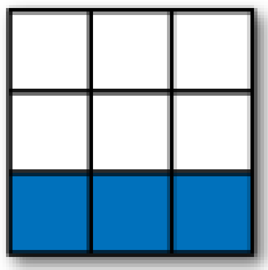
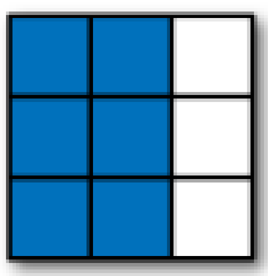
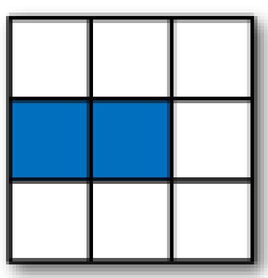
Indexing with slices

Like one-dimensional objects such as Python lists, ndarrays can be sliced.

Higher dimensional objects give more options, can slice one or more axes.

```
>>> arr2d=np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> arr2d
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> arr2d[:2]
array([[1, 2, 3],
       [4, 5, 6]])
>>>
```

sliced along axis 0, the first axis.

	Expression	Shape
	arr[:2, 1:]	(2, 2)
	arr[2]	(3,)
	arr[2, :]	(3,)
	arr[2:, :]	(1, 3)
	arr[:, :2]	(3, 2)
	arr[1, :2]	(2,)
	arr[1:2, :2]	(1, 2)

A slice selects a range of elements along an axis, can pass multiple slices like multiple indexes:

```
>>> arr2d=np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> arr2d
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
>>> arr2d[:2, 1:]
array([[2, 3],
       [5, 6]])
>>> arr2d[1, :2]
array([4, 5])
>>> arr2d[2, :1]
array([7])
>>> arr2d[:, :1]
array([[1],
       [4],
       [7]])
>>>
```

colon by itself means to take the entire axis

```
>>> arr[1,:1]
array([4])
>>> arr[1:,:1]
array([[4],
       [7]])
```

```
>>> arr[0:,:1]
array([[1],
       [4],
       [7]])
>>> arr[1:,:1]
array([[4],
       [7]])
>>> arr[2:,:1]
array([[7]])
>>>
```

Assigning to a slice expression assigns to the whole selection

```
>>> arr[:2, 1:] = 0
>>> arr
array([[1, 0, 0],
       [4, 0, 0],
       [7, 8, 9]])
>>>
```

Boolean Indexing

```
>>> names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
>>> names
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
>>>
```

To select all the rows with corresponding name 'Bob‘, Like arithmetic operations, comparisons (such as ==) with arrays are vectorized.

Comparing names with the string 'Bob' yields a boolean array:

```
>>> names=='Bob'
array([ True, False, False,  True, False, False, False])
>>>
```

Boolean array can be passed when indexing the array.

```
>>> data=np.random.randn(7,4)
>>> data
array([[-0.86774435, -1.3647272 , -0.51271666, -0.07731781],
       [-1.82871179,  0.77963895, -0.30853886, -2.07472519],
       [ 0.83641759,  0.28762554,  1.47243285,  0.84812344],
       [ 1.23327685, -0.02372054, -1.34387156,  0.68854698],
       [-1.17592275,  0.36851639,  0.72551948,  0.47677279],
       [ 1.40663368, -1.3725112 ,  0.26591758, -0.91105195],
       [ 1.01298754, -1.102427  , -0.49478179,  1.16180763]])
```

```
>>> data[names=='Bob']
array([[-0.86774435, -1.3647272 , -0.51271666, -0.07731781],
       [ 1.23327685, -0.02372054, -1.34387156,  0.68854698]])
```

```
>>> data[names == 'Bob', 2:]
array([[-0.51271666, -0.07731781],
       [-1.34387156,  0.68854698]])
```

To select everything but 'Bob', either use != or negate the condition using -

```
>>> names != 'Bob'
array([False,  True,  True, False,  True,  True,  True])
```

```
>>> data[names != 'Bob']
array([[-1.82871179,  0.77963895, -0.30853886, -2.07472519],
       [ 0.83641759,  0.28762554,  1.47243285,  0.84812344],
       [-1.17592275,  0.36851639,  0.72551948,  0.47677279],
       [ 1.40663368, -1.3725112 ,  0.26591758, -0.91105195],
       [ 1.01298754, -1.102427  , -0.49478179,  1.16180763]])
```

Selecting two of the three names to combine multiple boolean conditions, use boolean arithmetic operators like & (and) and | (or):

```
>>> mask = (names == 'Bob') | (names == 'Will')
>>> mask
array([ True, False,  True,  True,  True, False, False])
>>> data[mask]
array([[-0.86774435, -1.3647272 , -0.51271666, -0.07731781],
       [ 0.83641759,  0.28762554,  1.47243285,  0.84812344],
       [ 1.23327685, -0.02372054, -1.34387156,  0.68854698],
       [-1.17592275,  0.36851639,  0.72551948,  0.47677279]])
```

The Python keywords and and or do not work with boolean arrays.

```
>>> data[data < 0] = 0
>>> data
array([[0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.77963895, 0.        , 0.        ],
       [0.83641759, 0.28762554, 1.47243285, 0.84812344],
       [1.23327685, 0.        , 0.        , 0.68854698],
       [0.        , 0.36851639, 0.72551948, 0.47677279],
       [1.40663368, 0.        , 0.26591758, 0.        ],
       [1.01298754, 0.        , 0.        , 1.16180763]])
>>> data[names != 'Joe'] = 7
>>> data
array([[7.        , 7.        , 7.        , 7.        ],
       [0.        , 0.77963895, 0.        , 0.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [1.40663368, 0.        , 0.26591758, 0.        ],
       [1.01298754, 0.        , 0.        , 1.16180763]])
```

Fancy Indexing

Describe indexing using integer arrays

To select out a subset of the rows in a particular order, pass a list or ndarray of integers specifying the desired order:

```
>>> for i in range(8): arr[i] = i

>>> arr
array([[0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [4., 4., 4., 4.],
       [5., 5., 5., 5.],
       [6., 6., 6., 6.],
       [7., 7., 7., 7.]])
>>> arr[[4, 3, 0, 6]]
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

Using negative indices select rows from the end

```
>>> arr[[-3, -5, -7]]
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```

Transposing Arrays and Swapping Axes

Transposing is a special form of reshaping which similarly returns a view on the underlying data without copying anything.

Arrays have the transpose method and also the special T attribute:

```
>>> arr.reshape((5,3))
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
>>> arr
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
>>> arr = np.arange(15).reshape((3, 5))
>>> arr
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> arr.T
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

Compute the inner matrix product $X^T X$ using np.dot:

```
>>> arr=np.array([[1,2],[2,1]])
>>> arr
array([[1, 2],
       [2, 1]])
>>> np.dot(arr.T, arr)
array([[5, 4],
       [4, 5]])
```

Universal Functions: Fast Element-wise Array Functions

A universal function, or ufunc, is a function that performs element wise operations on data in ndarrays.

They are fast vectorized wrappers for simple functions that take one or more scalar values and produce one or more scalar results.

unary ufuncs

```
>>> arr = np.arange(10)
>>> arr
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.sqrt(arr)
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
>>> x=np.random.randn(8)
>>> y=np.random.randn(8)
>>> x
array([-0.22249215,  0.34766336, -0.86044609,  0.3514947 , -0.19405489,
       -0.78016562,  1.15165501,  0.14213304])
>>> y
array([-0.9050007 ,  0.01487067,  1.61904932,  0.85195635,  0.18685696,
       -0.45370959,  0.33643267,  3.07762033])
>>> np.maximum(x, y)
array([-0.22249215,  0.34766336,  1.61904932,  0.85195635,  0.18685696,
       -0.45370959,  1.15165501,  3.07762033])
```

add or maximum, take 2 arrays (thus, *binary ufuncs) and return a single array as the result*

ufunc can return multiple arrays

modf is a vectorized version, returns the fractional and integral parts of a floating point array.

```
>>> arr = np.random.randn(7) * 5
>>> arr
array([-3.38357207, 10.0757978 ,  8.51818849, -8.36465422, -1.40462229,
       -1.41573831, -1.45414175])
>>> np.modf(arr)
(array([-0.38357207,  0.0757978 ,  0.51818849, -0.36465422, -0.40462229,
       -0.41573831, -0.45414175]), array([-3., 10.,  8., -8., -1., -1., -1.]))
```

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating point, or complex values. Use `fabs` as a faster alternative for non-complex-valued data
`sqrt`	Compute the square root of each element. Equivalent to `arr ** 0.5`
`square`	Compute the square of each element. Equivalent to `arr ** 2`
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
`ceil`	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
`floor`	Compute the floor of each element, i.e. the largest integer less than or equal to each element
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as separate array
`isnan`	Return boolean array indicating whether each value is NaN (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-NaN) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise. Equivalent to `-arr`.

. *Unary ufuncs*

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum. fmax ignores NaN
minimum, fmin	Element-wise minimum. fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument

Binary universal functions

Linear Algebra

Matrix multiplication

```
>>> x = np.array([[1., 2., 3.], [4., 5., 6.]])
>>> x = np.array([[1., 2., 3.], [4., 5., 6.]])
>>> y = np.array([[6., 23.], [-1, 7], [8, 9]])
>>> x.dot(y)
array([[ 28.,  64.],
       [ 67., 181.]])
```

x.dot(y)

Function dot

Or

np.dot(x, y)

numpy.linalg has a standard set of matrix funtions like inverse, determinant etc.

```
>>> x=np.array([[1,-1],[0,2]])
>>> x
array([[ 1, -1],
       [ 0,  2]])
>>> inv(x)
array([[1. , 0.5],
       [0. , 0.5]])
>>> x.dot(inv(x))
array([[1., 0.],
       [0., 1.]])
```

```
>>> q,r=qr(x)
>>> q
array([[ 1.,  0.],
       [-0.,  1.]])
>>> r
array([[ 1., -1.],
       [ 0.,  2.]])
```

qr factorization of a matrix

Function	Description
diag	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
dot	Matrix multiplication
trace	Compute the sum of the diagonal elements
det	Compute the matrix determinant
eig	Compute the eigenvalues and eigenvectors of a square matrix
inv	Compute the inverse of a square matrix
pinv	Compute the Moore-Penrose pseudo-inverse inverse of a square matrix
qr	Compute the QR decomposition
svd	Compute the singular value decomposition (SVD)
solve	Solve the linear system Ax = b for x, where A is a square matrix
lstsq	Compute the least-squares solution to y = Xb

Commonly-used numpy.linalg functions

Random Number Generation

The numpy.random module supplements the built-in Python random with functions for efficiently generating whole arrays of sample values.

```
>>> import numpy as np
>>> samples = np.random.normal(size=(4, 4))
>>> samples
array([[ 0.29589026,  0.86814652,  2.44539663, -1.6558674 ],
       [ 0.26038384, -0.42915868,  1.11900427, -0.27310049],
       [-1.43695887,  0.70857343, -0.98984068,  0.0874919 ],
       [ 1.61193605, -0.1116812 ,  0.24035819, -0.60275642]])
>>> |
```

4 by 4 array of samples from the standard normal distribution using normal:

Function	Description
seed	Seed the random number generator
permutation	Return a random permutation of a sequence, or return a permuted range
shuffle	Randomly permute a sequence in place
rand	Draw samples from a uniform distribution
randint	Draw random integers from a given low-to-high range
randn	Draw samples from a normal distribution with mean 0 and standard deviation
binomial	Draw samples a binomial distribution
normal	Draw samples from a normal (Gaussian) distribution
beta	Draw samples from a beta distribution
chisquare	Draw samples from a chi-square distribution
gamma	Draw samples from a gamma distribution
uniform	Draw samples from a uniform [0, 1) distribution

Partial list of numpy.random functions

Python’s built-in random module, by contrast, only samples one value at a time.

numpy.random is well over an order of magnitude faster for generating very large samples:

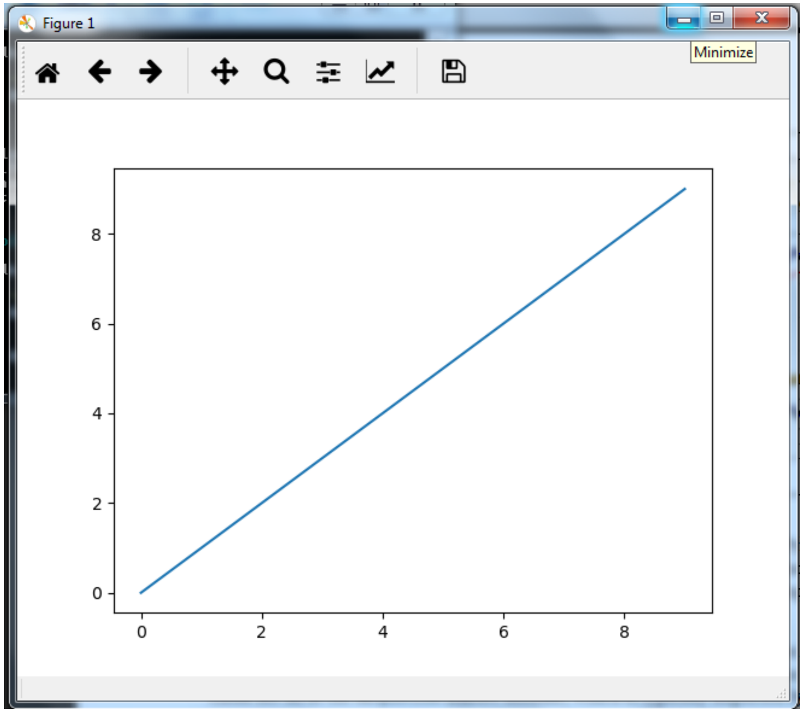
Plotting and visualization

Making plots and static or interactive visualizations is one of the most important tasks in data analysis.

matplotlib is a (primarily 2D) desktop plotting package designed for creating publication- quality plots.

There are several ways to interact with matplotlib.

The most common is through *pylab mode in IPython by running ipython --pylab.*



Test the working by making a simple plot:

plot(np.arange(10))

import matplotlib.pyplot as plt

Figures and Subplots

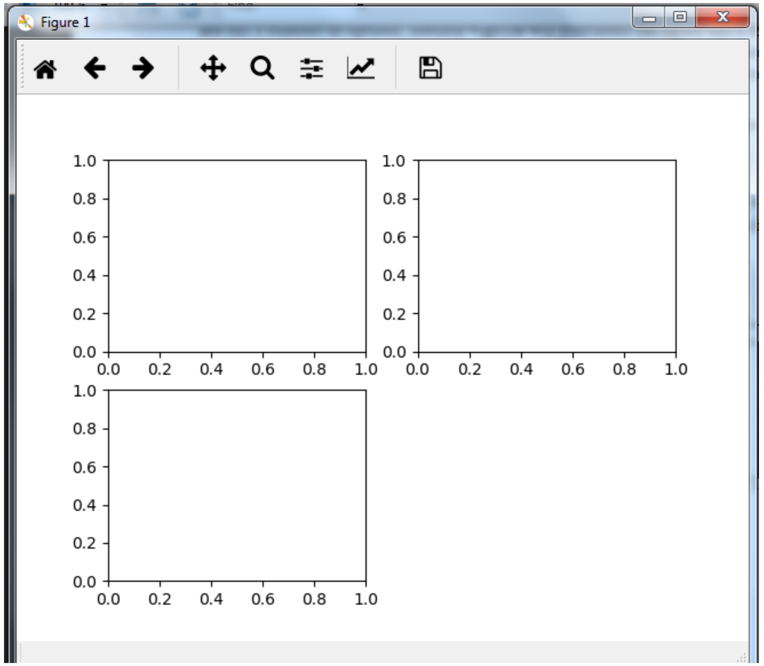
Plots in matplotlib reside within a Figure object.

fig = plt.figure()

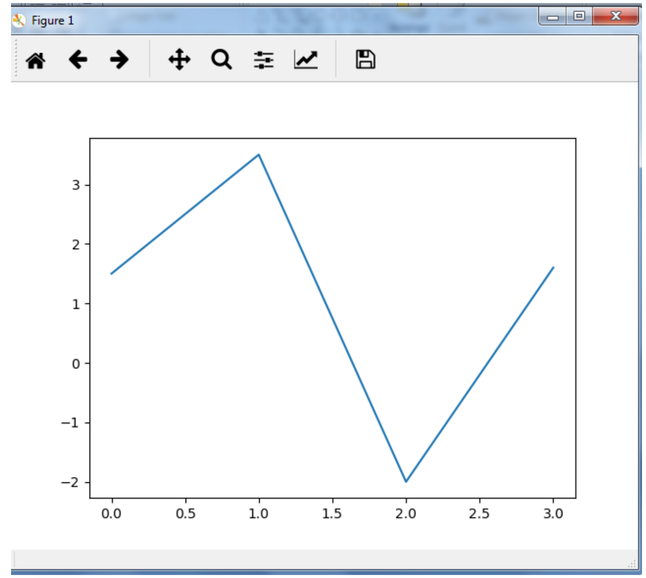
A new empty window should pop up

Create one or more subplots using add_subplot:

ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)



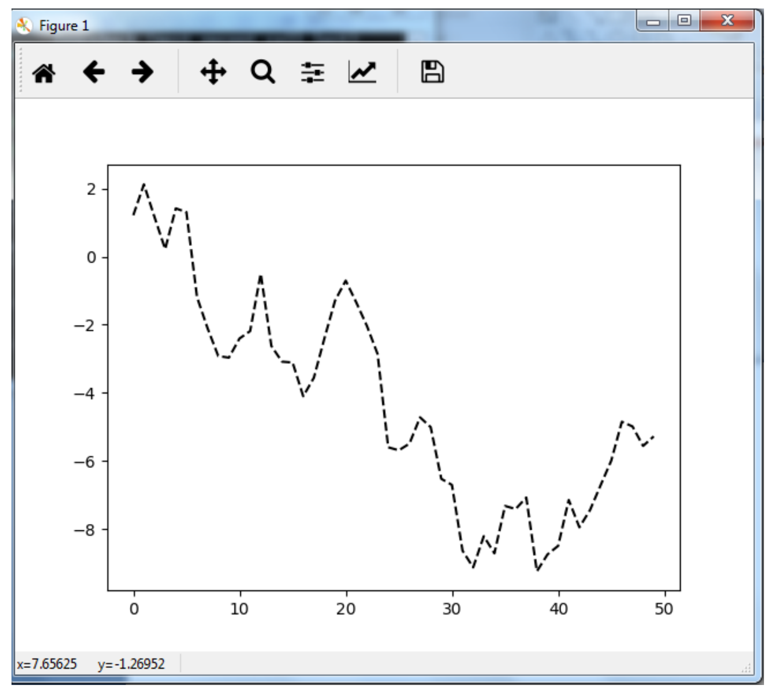
An empty matplotlib Figure with 3 subplots plt.plot([1.5, 3.5, -2, 1.6])



from numpy.random import randn

plt.plot(randn(50).cumsum(), 'k--')

'k--' *style option to plot a black dashed line.*



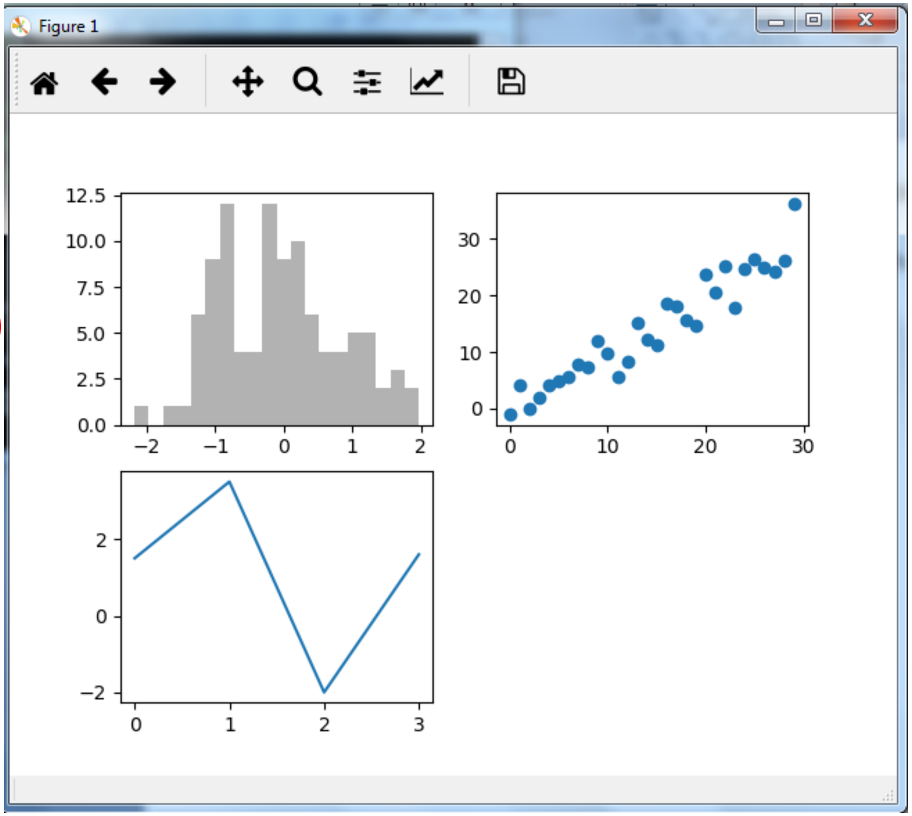
The objects returned by fig.add_subplot are AxesSubplot objects

Object can directly plot on empty subplots by calling each one's instance methods

_ = ax1.hist(randn(100), bins=20, color='k', alpha=0.3)

ax2.scatter(np.arange(30), np.arange(30) + 3 * randn(30))

plt.plot([1.5, 3.5, -2, 1.6])



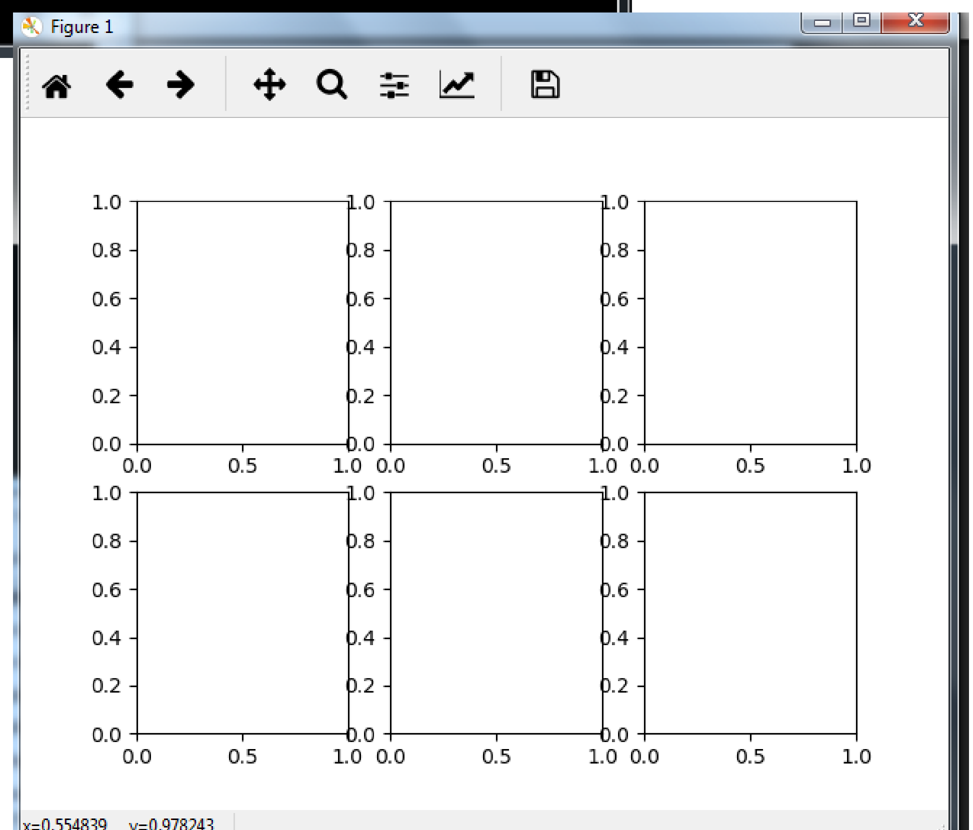
Argument	Description
`nrows`	Number of rows of subplots
`ncols`	Number of columns of subplots
`sharex`	All subplots should use the same X-axis ticks (adjusting the `xlim` will affect all subplots)
`sharey`	All subplots should use the same Y-axis ticks (adjusting the `ylim` will affect all subplots)
`subplot_kw`	Dict of keywords for creating the
`**fig_kw`	Additional keywords to `subplots` are used when creating the figure, such as `plt.subplots(2, 2, figsize=(8, 6))`

pyplot.subplots options

plt.subplots creates a new figure and returns a NumPy array containing the created subplot objects

```
In [39]: fig, axes = plt.subplots(2, 3)

In [40]: axes
Out[40]:
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0594E030>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x05964110>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0597F050>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x05992F70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x059ACED0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x059C9E10>]],
      dtype=object)

In [41]:
```

fig, axes = plt.subplots(2, 3)



Adjusting the spacing around subplots

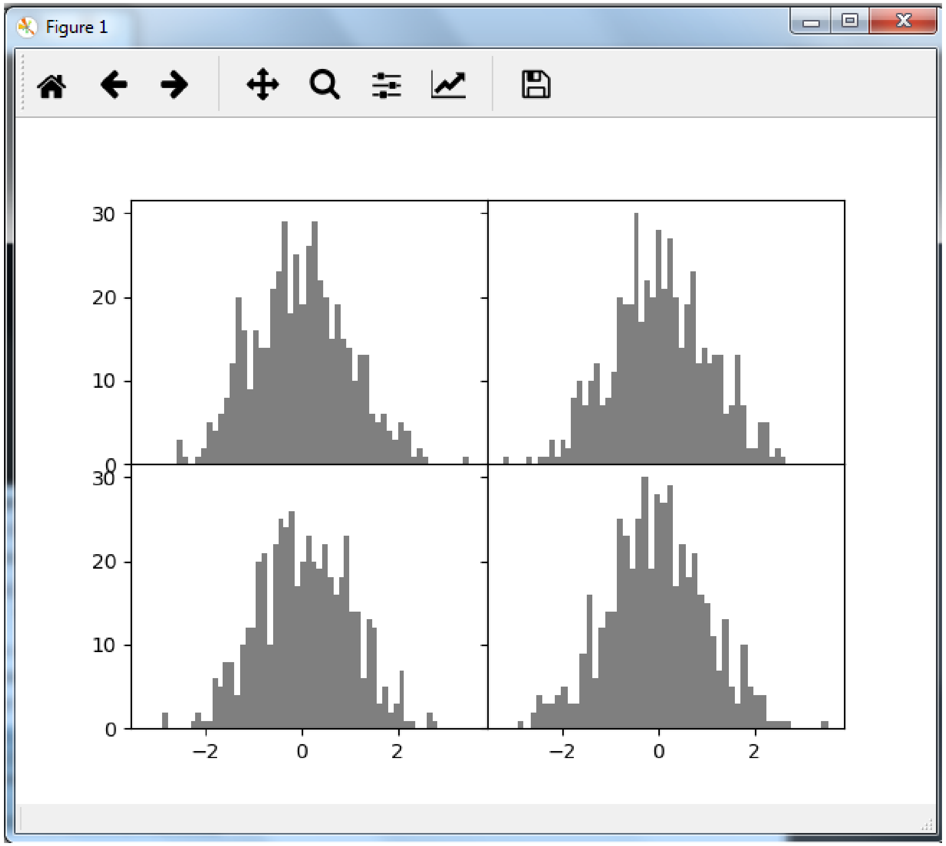
By default matplotlib leaves a certain amount of padding around the outside of the subplots and spacing between subplots.

This spacing is all specified relative to the height and width of the plot, the plot will dynamically adjust itself.

The spacing can be most easily changed using the subplots_adjust Figure method.

subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=None)

wspace and hspace controls the percent of the figure width and figure height, respectively, to use as spacing between subplots.



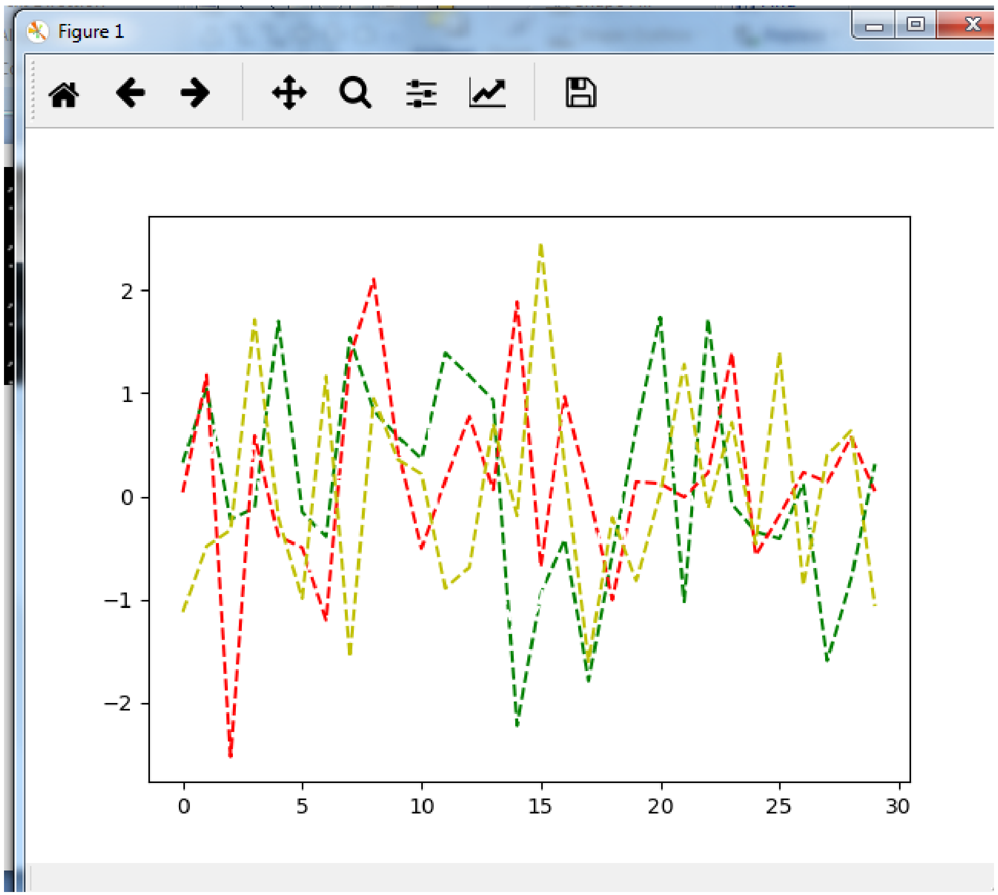

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
        for j in range(2):
                axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```

Colors, Markers, and Line Styles

Matplotlib's main plot function accepts arrays of X and Y coordinates and optionally a string abbreviation indicating colour and line style.

```
ax.plot(x, y, 'g--')

ax.plot(x, y, linestyle='--', color='g')
```

```
In [11]: plt.plot(randn(30), 'g--')
Out[11]: [<matplotlib.lines.Line2D at 0x521a390>]

In [12]: plt.plot(randn(30), 'r--')
Out[12]: [<matplotlib.lines.Line2D at 0x4c87850>]

In [13]: plt.plot(randn(30), 'w--')
Out[13]: [<matplotlib.lines.Line2D at 0x5237390>]

In [14]: plt.plot(randn(30), 'y--')
Out[14]: [<matplotlib.lines.Line2D at 0x52475f0>]
```



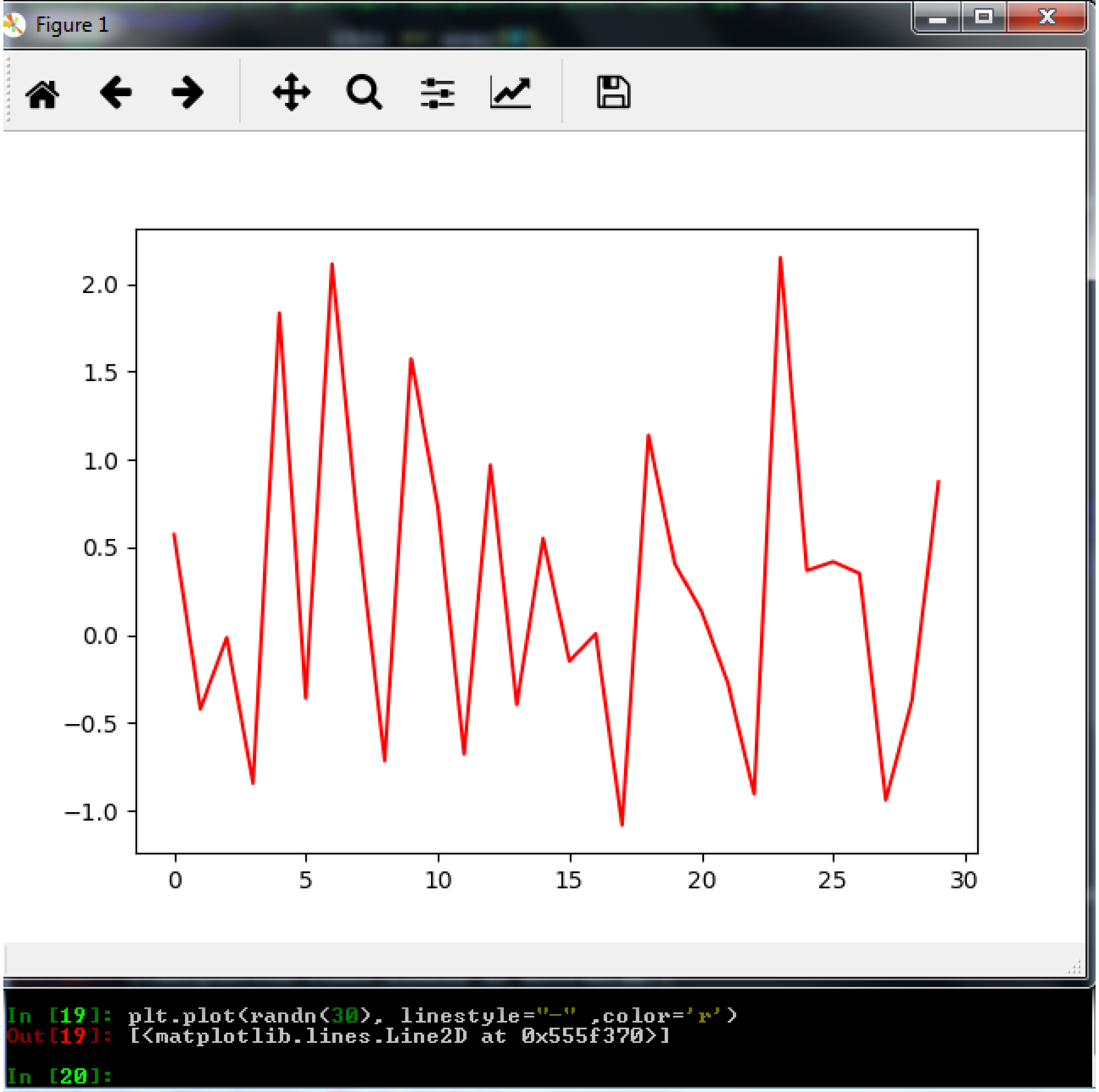


Figure 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
In [19]: plt.plot(randn(30), linestyle="-" ,color='r')
Out[19]: [<matplotlib.lines.Line2D at 0x555f370>]
In [20]:

Line plots can additionally have *markers to highlight the actual data points.*

matplotlib creates a continuous line plot, interpolating between points, it can occasionally be unclear where the points lie.

The marker can be part of the style string, which must have colour followed by marker type and line style

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

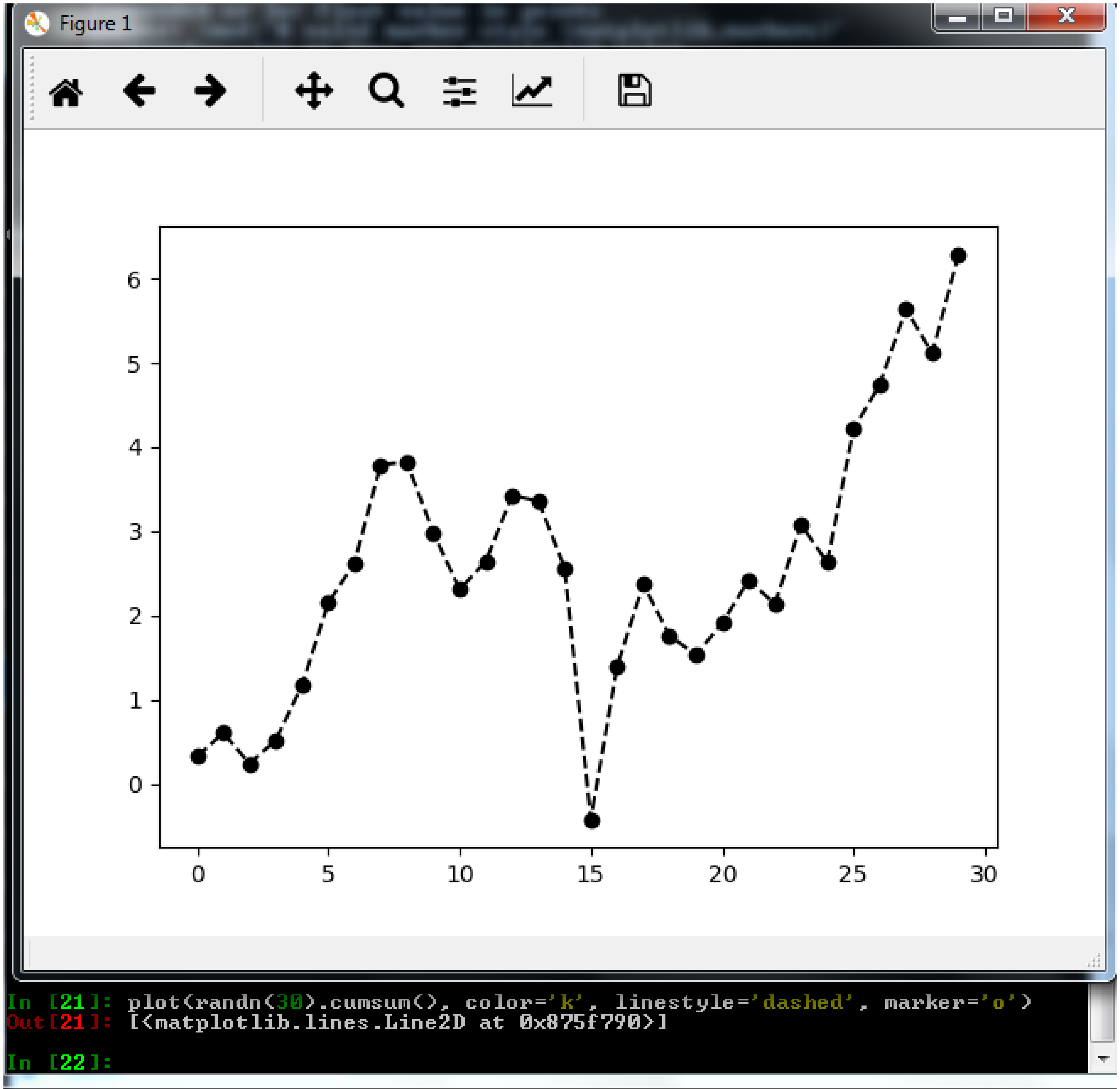
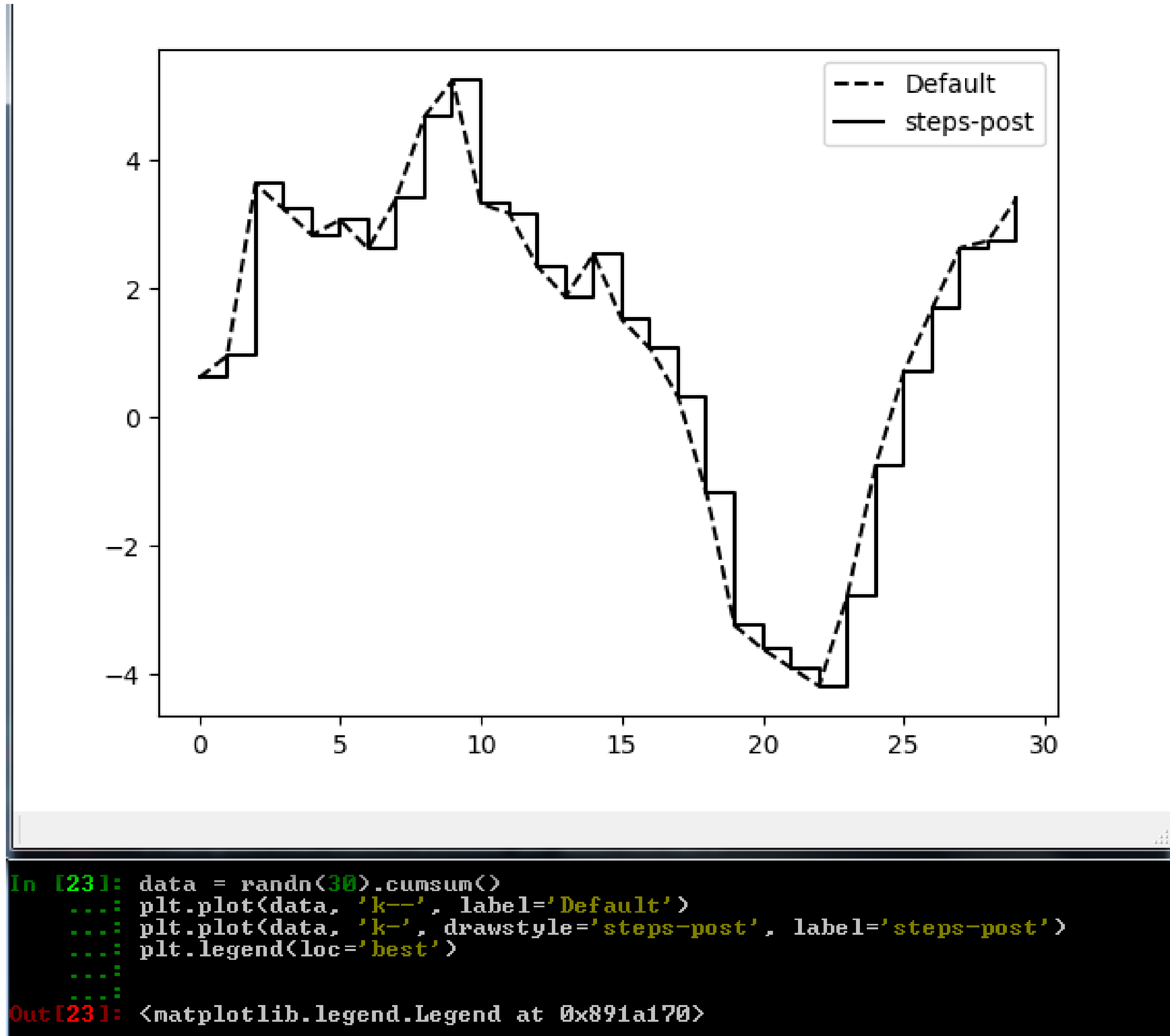


Figure 1
6
5
4
3
2
1
0
0
5
10
15
20
25
30
In [21]: plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
Out[21]: [<matplotlib.lines.Line2D at 0x875f790>]
In [22]:

drawstyle option

```
data = randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
plt.legend(loc='best')
```



Ticks, Labels, and Legends

The pyplot interface, designed for interactive use, consists of methods

- xlim - plot range
- xticks - tick locations
- xticklabels - tick labels

They can be used in two ways

- **Called with no arguments returns the current parameter value.**
  For example plt.xlim() returns the current X axis plotting range

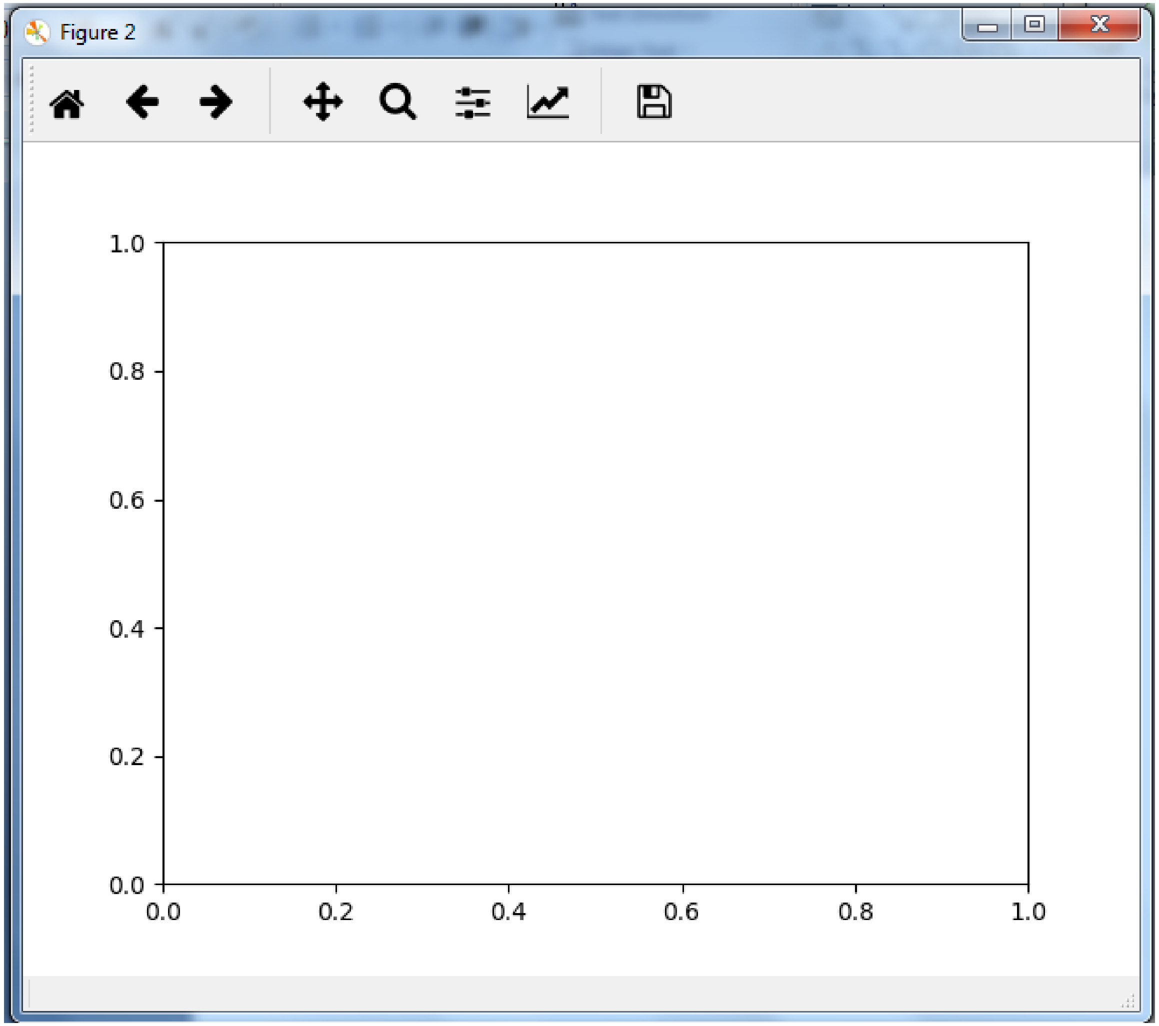
- **Called with parameters sets the parameter value.**
  So plt.xlim([0, 10]), sets the X axis range to 0 to 10

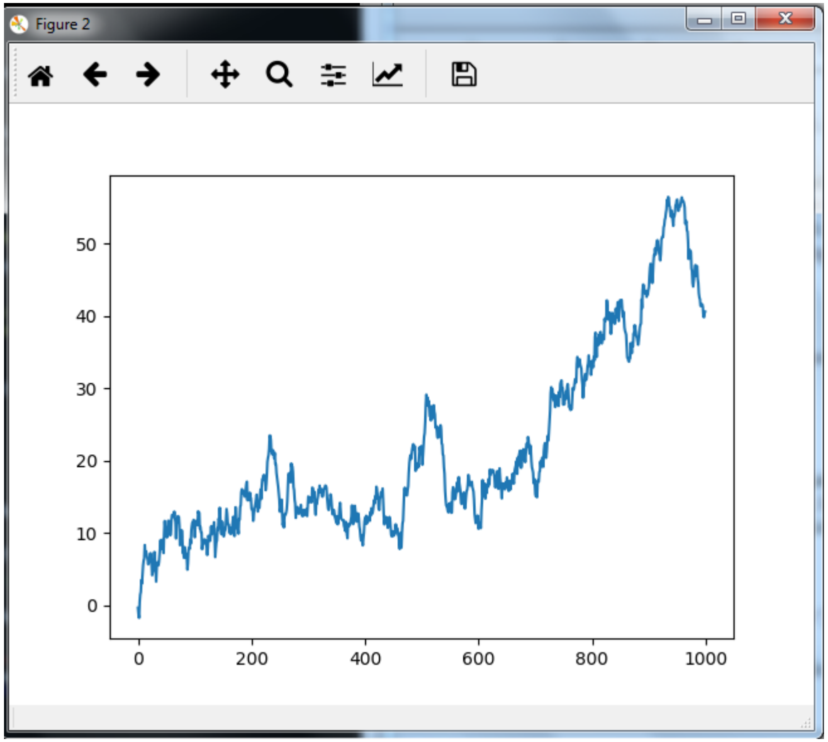
All such methods act on the active or most recently-created AxesSubplot

```
In [24]: plt.xlim([0,10])
Out[24]: (0, 10)

In [25]: plt.xlim()
Out[25]: (0.0, 10.0)
```

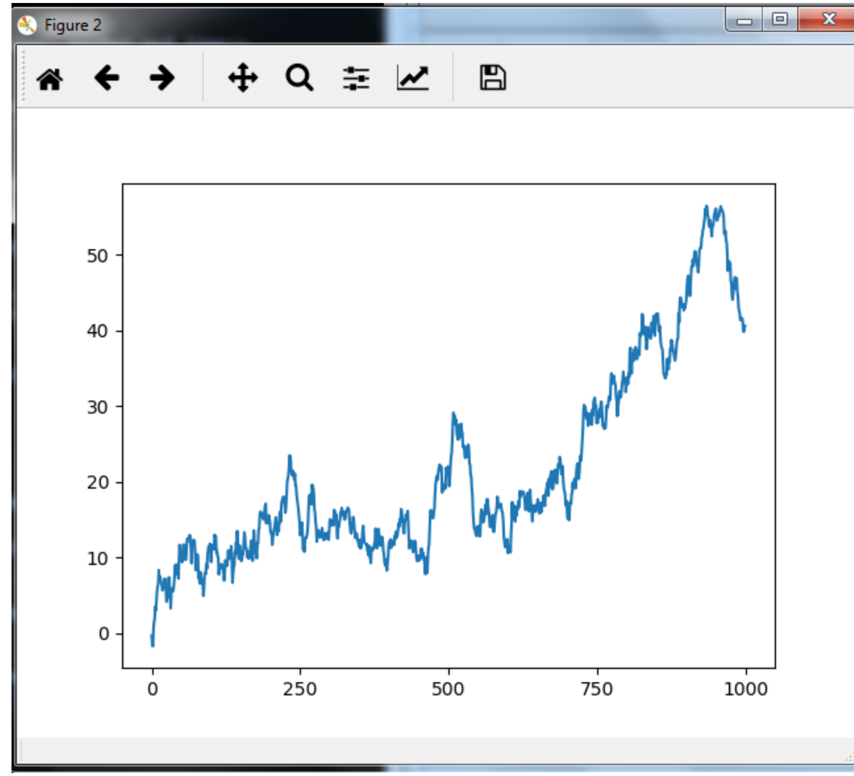
Setting the title, axis labels, ticks, and ticklabels

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
```

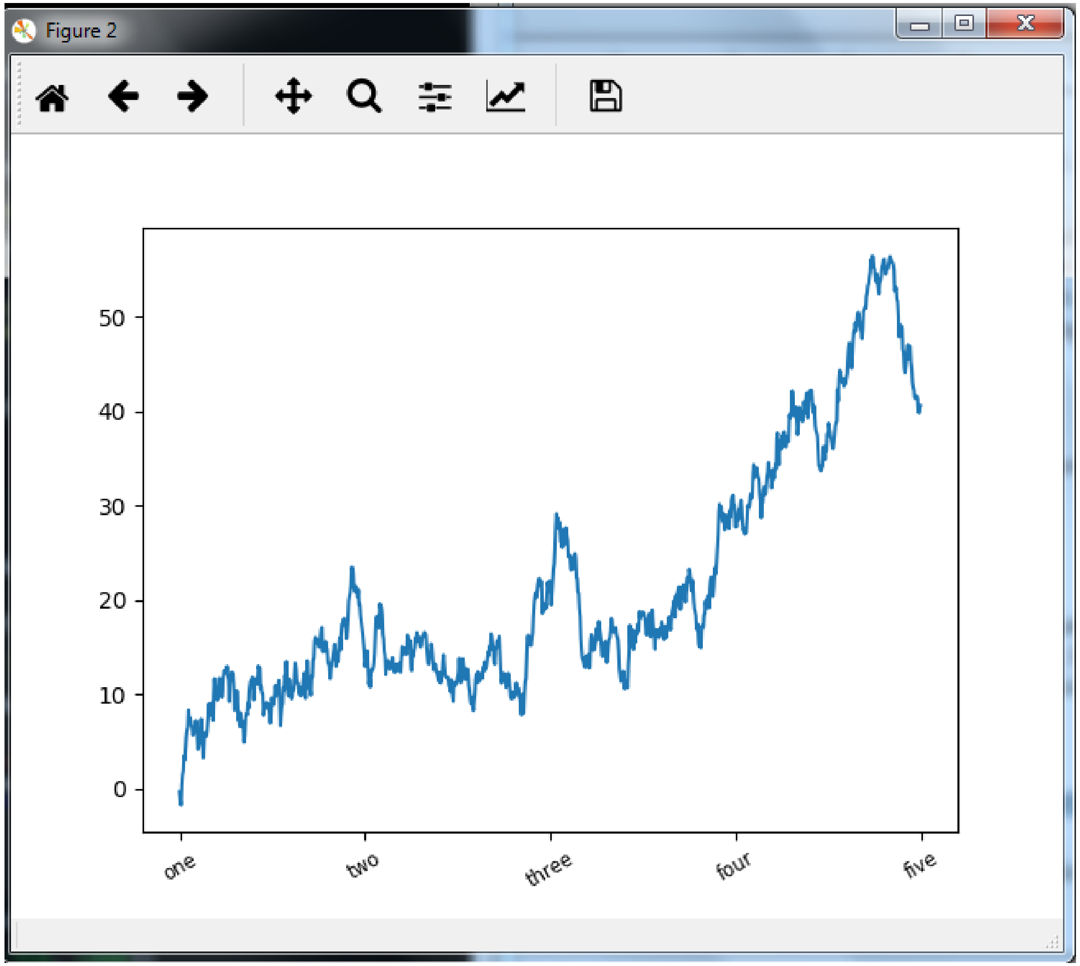




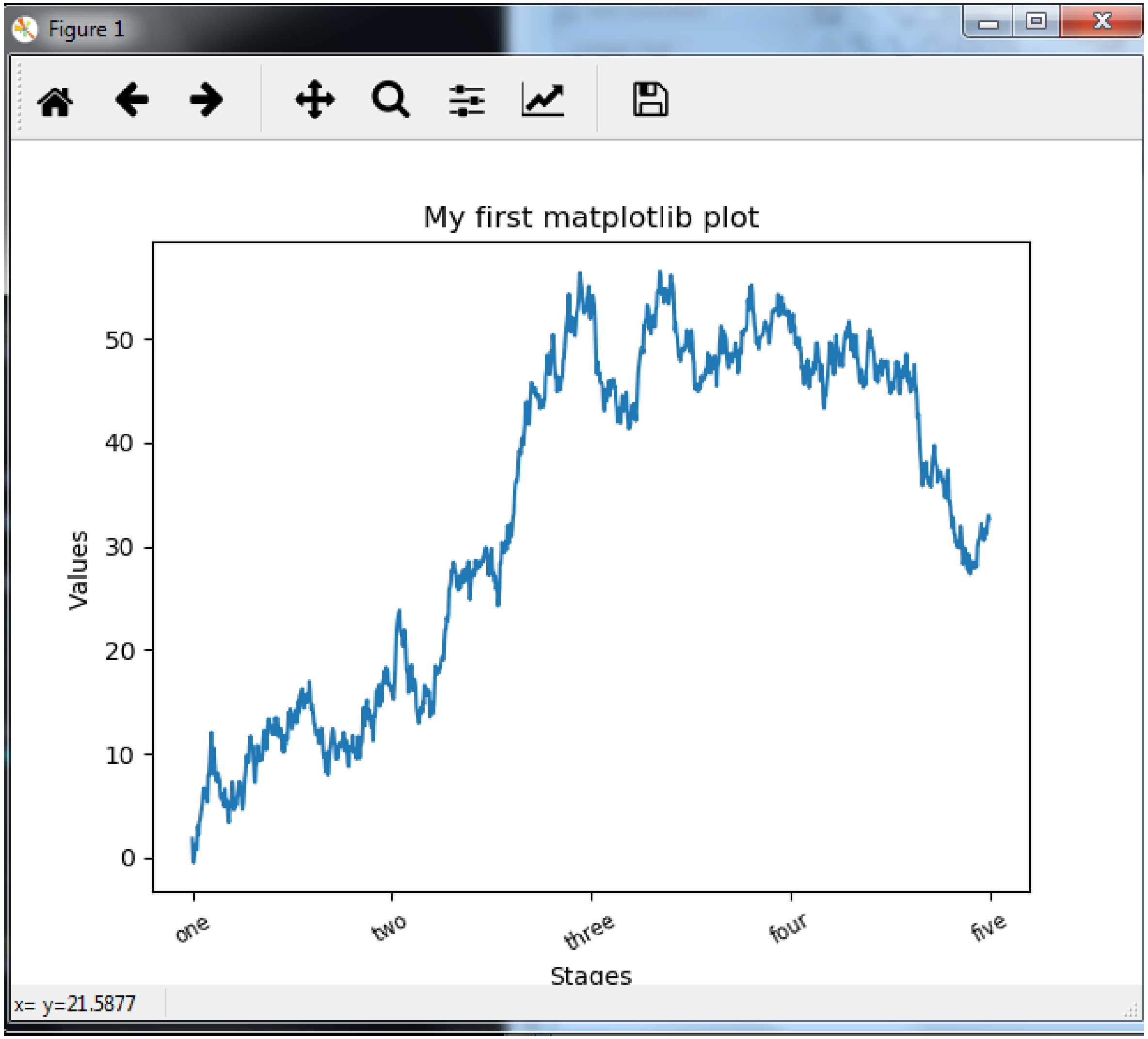
ax.plot(randn(1000).cumsum())



ticks = ax.set_xticks([0, 250, 500, 750, 1000])



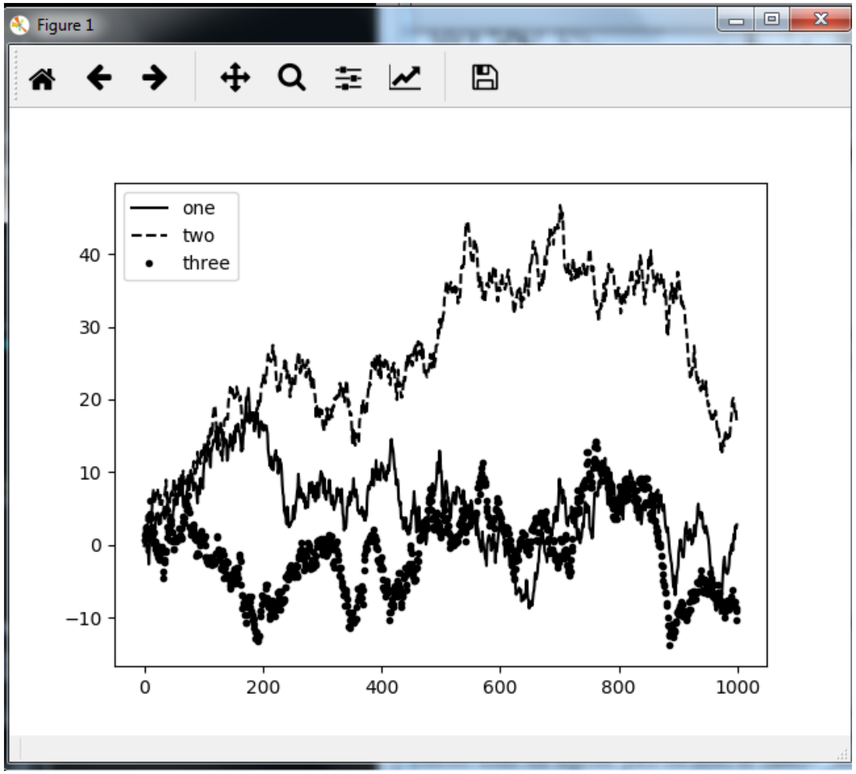
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], rotation=30, fontsize='small')



ax.set_title('My first matplotlib plot')
ax.set_xlabel('Stages')
ax.set_ylabel('Values')

Adding legends

Legends are element for identifying plot elements.




```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)

ax.plot(randn(1000).cumsum(), 'k', label='one')
[<matplotlib.lines.Line2D at 0x51db230>]

ax.plot(randn(1000).cumsum(), 'k--', label='two')
[<matplotlib.lines.Line2D at 0x51e3f30>]

ax.plot(randn(1000).cumsum(), 'k.', label='three')
[<matplotlib.lines.Line2D at 0x51cf1d0>]

ax.legend(loc='best')
<matplotlib.legend.Legend at 0x51d2ef0>
```

Working with CSV files

Pandas

It contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python.

pandas is built on top of NumPy and makes it easy to use in NumPy-centric applications

Introduction to pandas Data Structures

Series and DataFrame

Series

A Series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its *index*.

```
In [4]: obj = Series([4, 7, -5, 3])

In [5]: obj
Out[5]:
0    4
1    7
2   -5
3    3
```

Index on the left and the values on the right.

Index for the data has not specified, so a default one consisting of the integers 0 through N - 1 (where N is the length of the data) is created.

```
In [6]: obj.values
Out[6]: array([ 4,  7, -5,  3])

In [7]: obj.index
Out[7]: Int64Index([0, 1, 2, 3])
```

Access the array representation and index object of the Series via its values and index attributes

```
In [8]: obj2 = Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [9]: obj2
Out[9]:
d    4
b    7
a   -5
c    3
```

Creating a Series with an index identifying each data point

DataFrame

A DataFrame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).

The DataFrame has both a row and column index; it can be thought of as a dict of Series (one for all sharing the same index)

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
```

```
>>> import pandas as pd
>>> data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
'year': [2000, 2001, 2002, 2001, 2002],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
>>> frame=pd.DataFrame(data)
>>> frame
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
>>>
```

The resulting DataFrame will have its index assigned automatically as with Series, and the columns are placed in sorted order.

pandas have the following functions for reading tabular data as a DataFrame object.

Function	Description
read_csv	Load delimited data from a file, URL, or file-like object. Use comma as default delimiter
read_table	Load delimited data from a file, URL, or file-like object. Use tab ('\t') as default delimiter
read_fwf	Read data in fixed-width column format (that is, no delimiters)
read_clipboard	Version of read_table that reads data from the clipboard. Useful for converting tables from web pages

Options for the functions are

Indexing: can treat one or more columns as the returned DataFrame, and whether to get column names from the file, the user, or not at all.

Type inference and data conversion: this includes the user-defined value conversions and custom list of missing value markers.

Datetime parsing: includes combining capability, including combining date and time information spread over multiple columns into a single column in the result.

Iterating: support for iterating over chunks of very large files.

Unclean data issues: skipping rows or a footer, comments, or other minor things like numeric data with thousands separated by commas.

A CSV (comma-separated values) file is a text file that has a specific format which allows data to be saved in a table structured format.

These files are often used for exchanging data between different applications.

1. List each item on its own row.
Each row in CSV file must describe a single entity.

2. Include a column header row.
The first row in a CSV file is a column header row.

3. Check formatting within columns.
Some columns require certain formatting. If a single cell in a column has multiple values, separate the values with semicolons.

4. Check the file format and encoding

These files may sometimes be called Character Separated Values or Comma Delimited files.

They mostly use the comma character to separate (or delimit) data, but sometimes use other characters, like semicolons.

It is easy to export complex data from one application to a CSV file, and then import the data in that CSV file into another application.

To view the contents of a CSV file in Notepad, right-click it in File Explorer or Windows Explorer, and then select the “Edit” command.

Or use a spread sheet program to open the csv file.

Use read_csv to read it into a DataFrame

CSV File

df = pd.read_csv('ch06/ex1.csv')

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.read_csv('D:\\Programs\\python\\Attendance.csv')
>>> df
     Uni Reg No  Roll No    ...      Total  Percentage
0    TKR19CS001      1.0    ...     74/106        70.0
1    TKR19CS002      2.0    ...     75/109        69.0
2    TKR19CS003      3.0    ...     77/106        73.0
3    TKR19CS004      4.0    ...     74/106        70.0
4    TKR19CS005      5.0    ...    102/106        96.0
5    TKR19CS006      6.0    ...    101/109        93.0
6    TKR19CS007      7.0    ...    101/106        95.0
7    TKR19CS008      8.0    ...     97/106        92.0
8    TKR19CS009      9.0    ...     67/130        52.0
9    TKR19CS010     10.0    ...     92/106        87.0
10   TKR19CS011     11.0    ...     81/106        76.0
11   TKR19CS012     12.0    ...     64/109        59.0
12   TKR19CS013     13.0    ...    102/106        96.0
13   TKR19CS014     14.0    ...     79/106        75.0
14   TKR19CS015     15.0    ...     67/106        63.0
15   TKR19CS016     16.0    ...     88/106        83.0
16   TKR19CS017     17.0    ...    101/106        95.0
```

read_table and specifying the delimiter

pd.read_table('ch06/ex1.csv', sep=',')

```
>>> pd.read_table('D:\\Programs\\python\\Attendance.csv', sep=',')
     Uni Reg No  Roll No     ...        Total Percentage
0    TKR19CS001      1.0     ...       74/106       70.0
1    TKR19CS002      2.0     ...       75/109       69.0
2    TKR19CS003      3.0     ...       77/106       73.0
3    TKR19CS004      4.0     ...       74/106       70.0
4    TKR19CS005      5.0     ...      102/106       96.0
5    TKR19CS006      6.0     ...      101/109       93.0
6    TKR19CS007      7.0     ...      101/106       95.0
7    TKR19CS008      8.0     ...       97/106       92.0
8    TKR19CS009      9.0     ...       67/130       52.0
9    TKR19CS010     10.0     ...       92/106       87.0
10   TKR19CS011     11.0     ...       81/106       76.0
```

Assign default column names

pd.read_csv('ch06/ex2.csv', header=None)

```
>>> pd.read_csv('D:\\Programs\\python\\Attendance.csv', header=None)
              0         1    ...         9          10
0    Uni Reg No   Roll No    ...     Total  Percentage
1    TKR19CS001         1    ...    74/106          70
2    TKR19CS002         2    ...    75/109          69
3    TKR19CS003         3    ...    77/106          73
4    TKR19CS004         4    ...    74/106          70
5    TKR19CS005         5    ...   102/106          96
6    TKR19CS006         6    ...   101/109          93
7    TKR19CS007         7    ...   101/106          95
8    TKR19CS008         8    ...    97/106          92
9    TKR19CS009         9    ...    67/130          52
10   TKR19CS010        10    ...    92/106          87
11   TKR19CS011        11    ...    81/106          76
12   TKR19CS012        12    ...    64/109          59
13   TKR19CS013        13    ...   102/106          96
14   TKR19CS014        14    ...    79/106          75
15   TKR19CS015        15    ...    67/106          63
```

pd.read_csv('ch06/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])

specify column names

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

File: ch06/ex2.csv

column at index 4 or named 'message' as index column

```
In [853]: names = ['a', 'b', 'c', 'd', 'message']

In [854]: pd.read_csv('ch06/ex2.csv', names=names, index_col='message')
Out[854]:
         a   b   c   d
message
hello    1   2   3   4
world    5   6   7   8
foo      9  10  11  12
```

A hierarchical index from multiple columns

```
In [855]: !cat ch06/csv_mindex.csv
key1,key2,value1,value2
one,a,1,2
one,b,3,4
one,c,5,6
one,d,7,8
two,a,9,10
two,b,11,12
two,c,13,14
two,d,15,16
```

parsed = pd.read_csv('ch06/csv_mindex.csv', index_col=['key1', 'key2'])

parsed

```
           value1  value2
key1 key2
one  a          1       2
     b          3       4
     c          5       6
     d          7       8
two  a          9      10
     b         11      12
     c         13      14
     d         15      16
```

Table might not have a fixed delimiter, use whitespace or some other pattern to separate fields

regular expression \s+,

pass a regular expression as a delimiter for read_table

```
In [859]: result = pd.read_table('ch06/ex3.txt', sep='\s+')

In [860]: result
Out[860]:
            A         B         C
aaa -0.264438 -1.026059 -0.619500
bbb  0.927272  0.302904 -0.032399
ccc -0.264273 -0.386314 -0.217601
ddd -0.871858 -0.348382  1.100491
```

skiprows

For example, to skip the first, third, and fourth rows of a file with skiprows:

pd.read_csv('ch06/ex4.csv', skiprows=[0, 2, 3])

CSV file

```
>>> pd.read_csv("D:\\Programs\\python\data.csv",skiprows=[9,10,11])
  Student Activity Points Report          ...                     Unnamed: 2
0                Register Number          ...    Total Activity Points Earned
1                     TKR19CS001          ...                             71
2                     TKR19CS002          ...                             50
3                     TKR19CS003          ...                             50
4                     TKR19CS004          ...                             60
5                     TKR19CS005          ...                            106
6                     TKR19CS006          ...                             60
7                     TKR19CS007          ...                             80

[8 rows x 3 columns]
```

Handling missing values

Missing data is usually either not present (empty string) or marked by some *sentinel value.*

By default, pandas uses a set of commonly occurring sentinels, such as NA, -1.#IND, and NULL:

```
>>> import pandas as pd
>>> result = pd.read_csv('d:\\programs\\python\\data1.csv')
>>> result
  Student Activity Points Report          ...                 Unnamed: 2
0                Register Number          ...  Total Activity Points Earned
1                     TKR19CS001          ...                          71
2                     TKR19CS002          ...                          50
3                     TKR19CS003          ...                         NaN
4                     TKR19CS004          ...                          60
5                     TKR19CS005          ...                         NaN
6                            NaN          ...                         NaN
7                     TKR19CS007          ...                          80

[8 rows x 3 columns]
>>> 
```

isnull()??

```
>>> pd.isnull(result)
   Student Activity Points Report   Unnamed: 1  Unnamed: 2
0                            False       False       False
1                            False       False       False
2                            False       False       False
3                            False       False        True
4                            False       False       False
5                            False       False        True
6                             True       False        True
7                            False       False       False
```

Reading Text Files in Pieces

To read in a small piece of a file or iterate through smaller chunks of the file.

```
>>> pd.read_csv('d:\\programs\\python\\data1.csv', nrows=2)
  Student Activity Points Report          ...                  Unnamed: 2
0                Register Number          ...  Total Activity Points Earned
1                     TKR19CS001          ...                          71

[2 rows x 3 columns]
```

Writing Data Out to Text Format

Data can also be exported to delimited format

Using DataFrame's to_csv method, the data can out to a comma-separated file:

data.to_csv('d:\\programs\\python\\data3.csv')

Other delimiters can be used

```
>>> import sys
>>> data.to_csv(sys.stdout, sep='|')
|Student Activity Points Report |Unnamed: 1|Unnamed: 2
0|Register Number|Student Name|Total Activity Points Earned
1|TKR19CS001|ADARSH C V|71
2|TKR19CS002|ADARSH JAYACHANDRAN|50
3|TKR19CS003|ADARSH PRAKASAN|50
4|TKR19CS004|ADITHYA SATHYAN|60
5|TKR19CS005|AISWARYA K V|106
6|TKR19CS006|ALVIN ANTONY|60
7|TKR19CS007|AMRUTHA SATHEESH|80
8|#this is a comment||
9|to see the working ||
10|of skip||
```

Missing values appear as empty strings in the output.

It can be denoted by some other sentinel value:

```
>>> data.to_csv(sys.stdout, na_rep='NULL')
,Student Activity Points Report ,Unnamed: 1,Unnamed: 2
0,Register Number,Student Name,Total Activity Points Earned
1,TKR19CS001,ADARSH C V,71
2,TKR19CS002,ADARSH JAYACHANDRAN,50
3,TKR19CS003,ADARSH PRAKASAN,50
4,TKR19CS004,ADITHYA SATHYAN,60
5,TKR19CS005,AISWARYA K V,106
6,TKR19CS006,ALVIN ANTONY,60
7,TKR19CS007,AMRUTHA SATHEESH,80
8,#this is a comment,NULL,NULL
9,to see the working ,NULL,NULL
10,of skip,NULL,NULL
```

Both the row and column labels can be disabled as

data.to_csv(sys.stdout, index=False, header=False)

```
>>> data.to_csv(sys.stdout, index=False, header=False)
Register Number,Student Name,Total Activity Points Earned
TKR19CS001,ADARSH C V,71
TKR19CS002,ADARSH JAYACHANDRAN,50
TKR19CS003,ADARSH PRAKASAN,50
TKR19CS004,ADITHYA SATHYAN,60
TKR19CS005,AISWARYA K V,106
TKR19CS006,ALVIN ANTONY,60
TKR19CS007,AMRUTHA SATHEESH,80
#this is a comment,,
to see the working ,,
of skip,,
```

Handling Missing Data

Missing data is common in most data analysis applications. One of the goals in designing pandas was to make working with missing data.

pandas uses the floating point value NaN (Not a Number) to represent missing data in both floating as well as in non-floating point arrays

```
>>> import numpy as np
>>> string_data = Series(['aardvark', 'artichoke', np.nan, 'avocado'])
>>> string_data
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object
>>> string_data.isnull()
0    False
1    False
2     True
3    False
dtype: bool
```

The built-in Python None value is also treated as NA in object arrays:

string_data[0] = None

NA handling methods

Argument	Description
`dropna`	Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.
`fillna`	Fill in missing data with some value or using an interpolation method such as `'ffill'` or `'bfill'`.
`isnull`	Return like-type object containing boolean values indicating which values are missing / NA.
`notnull`	Negation of `isnull`.

Filtering Out Missing Data

```
>>> from numpy import nan as NA
>>> data = Series([1, NA, 3.5, NA, 7])
>>> data
0     1.0
1     NaN
2     3.5
3     NaN
4     7.0
dtype: float64
>>> data.dropna()
0     1.0
2     3.5
4     7.0
dtype: float64
>>>
```

```
>>> data[data.notnull()]
0     1.0
2     3.5
4     7.0
dtype: float64
>>>
```

With DataFrame objects, these are a bit more complex.

Data in DataFrame has to delete from row or column which are all NA or just those containing any NAs.

dropna by default drops any row containing a missing value:

```
>>> data = DataFrame([[1., 6.5, 3.], [1., NA, NA],[NA, NA, NA], [NA, 6.5, 3.]])
>>> data
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
>>> cleaned = data.dropna()
>>> cleaned
     0    1    2
0  1.0  6.5  3.0
>>>
```

Passing how='all' will only drop rows that are all NA:

```
>>> data
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
>>> cleaned = data.dropna()
>>> cleaned
     0    1    2
0  1.0  6.5  3.0
>>> data.dropna(how='all')
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
3  NaN  6.5  3.0
```

Dropping columns in the same way is only a matter of passing axis=1:

```
>>> data[4] = NA
>>> data
     0    1    2   4
0  1.0  6.5  3.0 NaN
1  1.0  NaN  NaN NaN
2  NaN  NaN  NaN NaN
3  NaN  6.5  3.0 NaN
>>> data.dropna(axis=1, how='all')
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
```

Filling in Missing Data

Rather than filtering out missing data to fill in the NA there are number of ways.

The fillna method.

Calling fillna with a constant replaces missing values with that value:

```
>>> data
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
>>> data.fillna(0)
     0    1    2
0  1.0  6.5  3.0
1  1.0  0.0  0.0
2  0.0  0.0  0.0
3  0.0  6.5  3.0
>>>
```

Calling fillna with a dict you can use a different fill value for each column

```
>>> data = DataFrame([[1., 6.5, 3.], [1., NA, NA],[NA, NA, NA], [NA, 6.5, 3.]])

>>> data

     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
>>> data.fillna({1: 0.5, 2: -1})

     0    1    2
0  1.0  6.5  3.0
1  1.0  0.5 -1.0
2  NaN  0.5 -1.0
3  NaN  6.5  3.0
>>>
```

Data Transformation

Removing Duplicates

The DataFrame method duplicated returns a boolean Series indicating whether each row is a duplicate or not:

```
>>> data = DataFrame({'k1': ['one'] * 3 + ['two'] * 4,'k2': [1, 1, 2, 3, 3, 4, 4]})

>>> data

    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
>>> data.duplicated()

0     False
1      True
2     False
3     False
4      True
5     False
6      True
dtype: bool
```

drop_duplicates returns a DataFrame where the duplicated array is True

```
>>> data.duplicated()

0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
>>> data.drop_duplicates()

    k1  k2
0  one   1
2  one   2
3  two   3
5  two   4
```

Vectorized string functions

```
>>> data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com','Rob': 'rob@gmail.com', 'Wes': np.nan}

>>> data

{'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com', 'Rob': 'rob@gmail.com', 'Wes': nan}
>>> data = Series(data)

>>> data

Dave     dave@google.com
Steve    steve@gmail.com
Rob        rob@gmail.com
Wes                  NaN
dtype: object
>>> data.str.contains('gmail')

Dave     False
Steve     True
Rob       True
Wes        NaN
dtype: object
```

Introduction to Micro services using Flask.

Flask is a micro framework for Python web development.

A framework is a library or collection of libraries to solve a part of a generic problem instead of a complete specific one.

To build web applications, there are issues to be solved,

- Routing from URLs to resources
- Inserting dynamic data into HTML
- Interacting with an end user

Micro framework

Implements only core functionality, leaves more advanced functionality to extensions.

The “micro” in micro framework means Flask aims to keep the core simple but extensible.

There is no native support in Flask for

Accessing databases
Validating web forms
Authenticating users, or other high-level tasks.

These and many other key services most web applications need are available through extensions that integrate with the core packages.

Using Virtual Environments

Virtual environment is a private copy of the Python interpreter.

Onto which packages can be installed privately, without affecting the global Python interpreter installed.

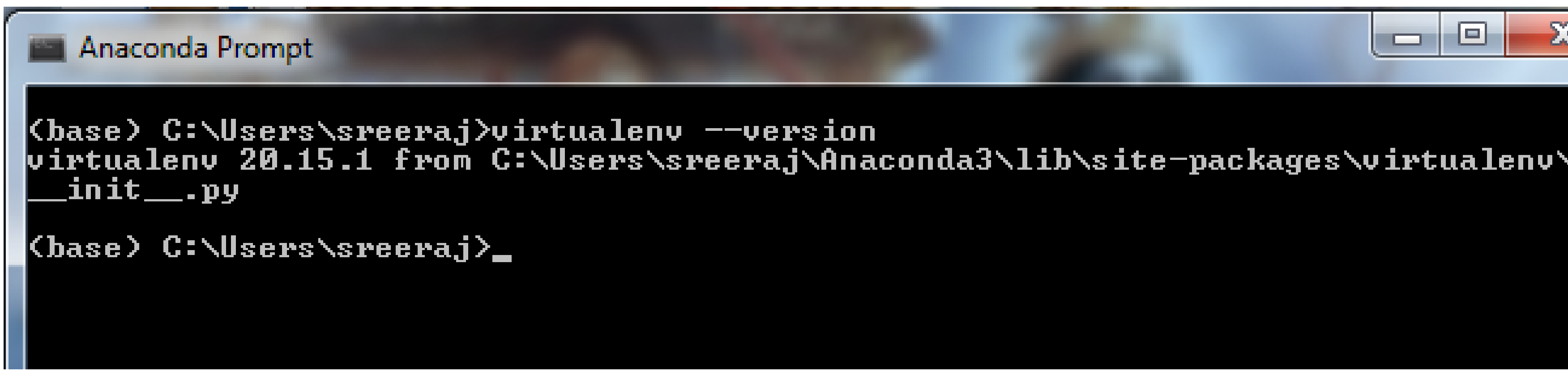
Creating a virtual environment for each application ensures that applications have access to only the packages that they use.

Global interpreter remains neat and clean.

Virtual environments are created with the third-party virtualenv utility.

check

virtualenv --version



pip install virtualenv CREATE DIRECTORY ACTIVATE

<name of environment>\Scripts\activate

pip install flask >>import flask

Running a web server on local machine that client can make requests to local machine.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    print("Hello")
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

from flask import Flask

Imports Flask from the package flask.

app = Flask(__name__)

Creates an instance of the Flask object using module's name as a parameter.

Flask uses this to resolve resources.

__name__ links module to the Flask object.

```
@app.route("/")
```

Line 3 is a Python decorator.

Flask uses decorators for URL routing.

Function directly below it should be called whenever a user visits the main *root page of web application (which is defined by the single forward slash).*

Here calls a function that takes the function defined under the decorator (in our case, index()) and returns a modified function.

Decorators in Python

Allows programmers to modify the behaviour of a function or class.

Decorators allow to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it.

```python
def hello():
    print("Hello")
    return "Hello World!"
```

Defines a very simple function that returns a message.

As this function is called by Flask when a user visits the application, the return value of this will be what is sent in response to a user who requests the landing page.

```python
if __name__ == "__main__":
    app.run()
```

This is a simple conditional statement that evaluates to True if the application is run directly .

It is used to prevent Python scripts from being unintentionally run when they are imported into other Python files.

Running the code

From command prompt (Python's Home)

>cd newproj
newproj> python hello.py

visit http://127.0.0.1:5000 in web browser

Anaconda Prompt - python hello.py

```
 Directory of C:\Users\sreeraj\newproj

17-07-2022  23:15    <DIR>          .
17-07-2022  23:15    <DIR>          ..
17-07-2022  00:08               171 hello.py
15-07-2022  23:04               154 hello.py.txt
17-07-2022  23:27    <DIR>          templates
17-07-2022  23:59               462 test.py
15-07-2022  22:57    <DIR>          venv
17-07-2022  00:08    <DIR>          __pycache__
               3 File(s)            787 bytes
               5 Dir(s)   3,448,061,952 bytes free

(base) C:\Users\sreeraj\newproj>python hello.py
 * Serving Flask app "hello" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Hello
127.0.0.1 - - [25/Jul/2022 11:08:15] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Jul/2022 11:08:16] "GET /favicon.ico HTTP/1.1" 404 -
```

End