lab1-python-data-types

October 5, 2024

[ ]:
```
print(2+2+2)
```

[ ]:
```
print
```

[ ]:
```

```

[ ]:
```
print('Hello, World!')
```

1 Introduction to python

This is where we can add text

We must designate this cell as a 'Markdown' cell.

With a single hashtag we get a big section header.

We can also add subsections.

1.1 Subsection

It is convenient to communicate how your code works and what it is doing with markdown cells!

You can also use LaTeX commands in-line with single dollar signs: $\int_0^\infty f(x)dx$

You can add LaTeX equations centered on their own lines with double dollar signs:

$$\sum_{i=1}^{\infty} \frac{1}{2^i}$$

2 Shortcuts

We can look at the existing keyboard shortcuts through the help menu, and we can make our own there

The most important shortcuts are 1. Shift + Enter 2. Ctrl + Enter 3. Alt + Enter 4. Tab 5. Shift + Tab

[ ]:
```
# we can add comments to code cells using a hashtag
# on windows you can comment a line with 'ctrl + /'
# on mac you can comment a line with 'cmd + /'
#
```

```
# assign a variable to a literal value
a=3 # assign variable a to literal 3
print('five times a is',5*a,'and 3 multiplied by a is',3*a)
a=2
print('five times a is',5*a,'and 3 multiplied by a is',3*a)
```

[ ]:
```
# we can also inspect the type of variables
print('a is of type',type(a))
b = 2.1
print('b is of type',type(b))
c = float(a)/b
print('c is of type',type(c), 'and its value is',c)
d = a/b
print('d is of type',type(c), 'and its value is',d)
```

3 Collections of variables

There are 4 main types of collections of variables in python 1. list 2. touple 3. set 4. dictionary

[1]:
```
# a list is an ordered and indexed collection of values that are changeable and␣
↪allows duplicates
simple_list = ['Dan',2,3,4,'python',2.71]
print(simple_list)
print(simple_list[0])
print(simple_list[3])
print(simple_list[-1])
print(simple_list[-2])
print(len(simple_list))
```

```
['Dan', 2, 3, 4, 'python', 2.71]
Dan
4
2.71
python
6
```

[2]:
```
# it is easy to change entries of a list
simple_list[3] = 52
print(simple_list[3]) # index is based on 0!!!
simple_list.append('to the back')
print(simple_list)
print(len(simple_list))
simple_list.pop(4)
print(simple_list)
```

```
52
['Dan', 2, 3, 52, 'python', 2.71, 'to the back']
```

```
7
['Dan', 2, 3, 52, 2.71, 'to the back']
```

[6]:
```
# it is easy to initialize a list of a particular size with all entries the same
repeated_list = [5]*4
print(repeated_list)
```

```
[5, 5, 5, 5]
```

[10]:
```
# we can even use lists as entries of a list
simple_list[1] = [1,2,3]
print(simple_list)
print(simple_list[1])
print(simple_list[1][2])
```

```
['Dan', [1, 2, 3], 3, 52, 2.71, 'to the back']
[1, 2, 3]
3
```

[11]:
```
# copying lists is a little tricky
list2 = simple_list
print(list2)
print(simple_list)
list2[1] = 0
print(list2)
print(simple_list)
```

```
['Dan', [1, 2, 3], 3, 52, 2.71, 'to the back']
['Dan', [1, 2, 3], 3, 52, 2.71, 'to the back']
['Dan', 0, 3, 52, 2.71, 'to the back']
['Dan', 0, 3, 52, 2.71, 'to the back']
```

[12]:
```
# we must use the copy method to prevent this behavior
list3 = simple_list.copy()
print(list3)
print(simple_list)
list3[0] = 'Mitchell'
print(list3)
print(simple_list)
```

```
['Dan', 0, 3, 52, 2.71, 'to the back']
['Dan', 0, 3, 52, 2.71, 'to the back']
['Mitchell', 0, 3, 52, 2.71, 'to the back']
['Dan', 0, 3, 52, 2.71, 'to the back']
```

[15]:
```
# a tuple is an ordered collection of values that are unchangeable and allows␣
 ↪duplicates
simple_tuple = (12,42,11,99,2351)
```

```
print(simple_tuple)
print(simple_tuple[1])
```

```
(12, 42, 11, 99, 2351)
42
```

[16]:
```
# it is not possible to change entries in a touple
simple_tuple[0] = 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-d8431e8d65c6> in <module>
      1 # it is not possible to change entries in a touple
----> 2 simple_tuple[0] = 5

TypeError: 'tuple' object does not support item assignment
```

[24]:
```
# you can work around this though...
dummy = list(simple_tuple)
dummy[0] = 5
simple_tuple = tuple(dummy)
print(simple_tuple)
```

```
(5, 42, 11, 99, 2351)
```

[25]:
```
# a set is an unordered collection of values that are changeable and does not␣
 ↪allow duplicates
simple_set = {11,-2,'water',-2}
print(simple_set)
```

```
{'water', 11, -2}
```

[26]:
```
print(simple_set[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-263fc76fcfaa> in <module>
----> 1 print(simple_set[1])

TypeError: 'set' object is not subscriptable
```

[27]:
```
print('water' in simple_set)
```

```
True
```

[28]:
```
# you can't change values but you can add and remove entries from a set
simple_set.add(72)
print(simple_set)
simple_set.remove('water')
print(simple_set)
```

```
{72, 'water', 11, -2}
{72, 11, -2}
```

[29]:
```
# a dictionary is a collection of values that are unordered (but indexed) and␣
 ↪changeable
simple_dict = {
    "brand": "Apple",
    "product": "iPhone",
    "model": "X"
}
print(simple_dict)
print('I bought␣
 ↪an',simple_dict['product'],"model",simple_dict['model'],'from',simple_dict['brand'])
```

```
{'brand': 'Apple', 'product': 'iPhone', 'model': 'X'}
I bought an iPhone model X from Apple
```

[30]:
```
simple_dict["model"] = "11 pro"
print('I bought␣
 ↪an',simple_dict['product'],"model",simple_dict['model'],'from',simple_dict['brand'])
```

```
I bought an iPhone model 11 pro from Apple
```

[31]:
```
# we can also add entries to the dictionary
simple_dict['color'] = 'red'
print('I bought␣
 ↪a',simple_dict["color"],simple_dict['product'],"model",simple_dict['model'],'from',simple_d
```

```
I bought a red iPhone model 11 pro from Apple
```

4 Accessing data in lists

A list is the main data type we will use for now and there are some rules we need to remember for accessing data in lists.

Later we'll get into NumPy arrays, but the same rules here apply to them.

With NumPy arrays we can access data in more ways.

[32]:
```
simple_list = [1,5,2,7,3,66,1923,11]
# we can access multiple values in the list using the :
# we must be careful though
```

```
print(simple_list[2:5])
# why isn't 66 in the output?
# the number to the left of the : is inclusive, but the number to the right is␣
  ↪exclusive
print(simple_list[2:3])
# we can also use negative numbers
print(simple_list[-5:-1])
```

```
[2, 7, 3]
[2]
[7, 3, 66, 1923]
```

numpy-intro-part1-class

October 5, 2024

[2]:
```
import numpy as np
```

0.0.1 What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy stands for Numerical Python.

0.0.2 Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

0.0.3 Installation of NumPy

If you have Python and pip already installed on a system, then installation of NumPy is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

[4]:
```
import numpy as np
```

[7]:
```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

[9]:
```
# Checking NumPy Version
# The version string is stored under __version__ attribute.

print(np.__version__)
```

```
1.24.4
```

0.0.4 Create a numPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

[10]:
```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

[11]:
```
#Use a tuple to create a NumPy array:
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

```
[1 2 3 4 5]
```

[12]:
```
#Create a 0-D array with value 42
arr = np.array(42)
print(arr)
```

```
42
```

[13]:
```
# Create a 1-D array containing the values 1,2,3,4,5:
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

[14]:
```
#Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

[15]:
```
#create a 3-D array with two 2-D arrays, both containing two arrays with the␣
 ↪values 1,2,3 and 4,5,6:
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

0.0.5 Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

[16]:
```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

0.0.6 Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

[17]:
```
#Create an array with 5 dimensions and verify that it has 5 dimensions:

arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

0.0.7 Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

[20]:
```
arr = np.array([1, 2, 3, 4])
print(arr[0])
print(arr[2])
print(arr[2] + arr[3])
```

```
1
3
```

7

0.0.8 Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

[21]:
```
#Access the element on the first row, second column:
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

[22]:
```
#Access the element on the 2nd row, 5th column:
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

```
5th element on 2nd row:  10
```

[5]:
```
#Access 3-D Arrays

#Access the third element of the second array of the first array:
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[1, 0, 2])
```

```
9
```

[24]:
```
#Print the last element from the 2nd dim:
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

```
Last element from 2nd dim:  10
```

0.0.9 NumPy Array Slicing

0.0.10 Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

[25]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
[2 3 4 5]
```

[27]:
```
#Slice elements from index 4 to the end of the array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

```
[5 6 7]
```

[28]:
```
#Slice elements from the beginning to index 4 (not included):
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

```
[1 2 3 4]
```

[32]:
```
### Negative Slicing
### Slice from the index 3 from the end to index 1 from the end:
### Use the minus operator to refer to an index from the end:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

```
[5 6]
```

[34]:
```
### STEP
#Use the step value to determine the step of the slicing:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

```
[2 4]
```

[35]:
```
#Return every other element from the entire array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

```
[1 3 5 7]
```

[36]:
```
### Slicing 2-D Arrays
#From the second element, slice elements from index 1 to index 4 (not included):
```

[37]:
```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

[38]:
```
#From both elements, return index 2:
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])
```

```
[3 8]
```

[39]:
```
###From both elements, slice index 1 to index 4 (not included), this will␣
 ↪return a 2-D array:
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

[ ]:

numpy-intro-part2-class

October 5, 2024

[5]:
```
import numpy as np
```

[8]:
```
print(np.cos(np.pi))
print(np.sqrt(1.21))
print(np.log(np.exp(5.2)))
```

```
-1.0
1.1
5.2
```

[12]:
```
# we can create numpy arrays by converting lists
# this is a vector
vec=np.array([1,2,3])
print(vec)
#we can create matrices by converting list of lists
mat=np.array([[1,2,3],[4,5,6],[6,8,1]])
print('')
print(mat)
print(mat.T)
```

```
[1 2 3]

[[1 2 3]
 [4 5 6]
 [6 8 1]]
[[1 4 6]
 [2 5 8]
 [3 6 1]]
```

[13]:
```
# there are lots of other ways to create numpy arrays
vec2=np.arange(0,15)# np.arange(start,stop)
print(vec2)
print('')
vec3=np.arange(3,21,6)#np.arange(start,stop,step)
print(vec3)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

```
[ 3  9 15]
```

vec2 = np.arange(0, 15) creates a 1D array of integers from 0 to 14:

vec3 = np.arange(3, 21, 6) creates a 1D array starting at 3, ending before 21, with steps of 6

[33]:
```
vec4=np.linspace(0,5,10)
print(vec4)
print('')
print(vec4.reshape(5,2))
vec4_reshaped=vec4.reshape(5,2)
print(vec4_reshaped)
print(vec4)
```

```
[0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]

[[0.         0.55555556]
 [1.11111111 1.66666667]
 [2.22222222 2.77777778]
 [3.33333333 3.88888889]
 [4.44444444 5.        ]]
[[0.         0.55555556]
 [1.11111111 1.66666667]
 [2.22222222 2.77777778]
 [3.33333333 3.88888889]
 [4.44444444 5.        ]]
[0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]
```

[19]:
```
mat2=np.zeros((5,3))#np.zeros(shape)
print(mat2)
mat3=np.ones((3,5))#np.ones(shape)
print('')
print(mat3)
mat4=np.eye(3)#np.eye(N)
mat4
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

[20]:
```
#we can +-*/ arrays together if they are right size
vec5=np.arange(1,6)
vec6=np.arange(3,8)
print(vec5)
print(vec6)
print(vec5+vec6)
print(1/vec5)
print(np.sqrt(vec6))
```

```
[1 2 3 4 5]
[3 4 5 6 7]
[ 4  6  8 10 12]
[1.         0.5        0.33333333 0.25       0.2       ]
[1.73205081 2.         2.23606798 2.44948974 2.64575131]
```

[21]:
```
#we can do matrix multiplication
print(mat)
print('')
print(vec)
print()
product=np.matmul(mat,vec)
print(product)
```

```
[[1 2 3]
 [4 5 6]
 [6 8 1]]

[1 2 3]

[14 32 25]
```

[22]:
```
print(np.linalg.solve(mat,product))
print('')
print(np.linalg.inv(mat))
```

```
[1. 2. 3.]

[[-1.59259259  0.81481481 -0.11111111]
 [ 1.18518519 -0.62962963  0.22222222]
 [ 0.07407407  0.14814815 -0.11111111]]
```

[25]:
```
#we can find the unique values in an array
vec7=np.array(['blue','red','orange','purple','purple','orange','red',6])
print(vec7)
print(np.unique(vec7))
```

```
['blue' 'red' 'orange' 'purple' 'purple' 'orange' 'red' '6']
['6' 'blue' 'orange' 'purple' 'red']
```

[27]:
```
#we can also use numpy to generate samples of a randam variables
rand_mat=np.random.rand(5,5)
print(rand_mat)
rand_mat2=np.random.randn(10,5)
print('')
print(rand_mat2)
```

```
[[0.35491376 0.31901766 0.74829223 0.17395797 0.79458383]
 [0.276769   0.71264721 0.92060162 0.39904183 0.47449625]
 [0.29687856 0.98653463 0.02155436 0.59915688 0.71097483]
 [0.90170019 0.46413194 0.7239314  0.26621443 0.55619685]
 [0.45308148 0.12287803 0.46185378 0.86596757 0.93506499]]

[[-2.43922848  0.25807142  0.79781244 -1.78471036 -0.67637784]
 [-1.37416289  2.592822    0.51008251 -1.06251643 -1.17001739]
 [ 0.17219319 -1.32932456  1.31610806  0.55663312 -0.19538025]
 [-0.36761458  0.16187897  1.41260083 -0.43588735  1.07165426]
 [-1.5377185  -0.66842103  0.95070663 -0.1744214   0.98607016]
 [-2.42582026 -1.10378979  1.52538208  0.54055068  0.36862647]
 [-0.65708852  0.74245707 -0.44827772  0.87054388 -0.39047218]
 [-0.46797044 -0.32347933 -0.52645743 -0.24327371  0.58093098]
 [ 0.62353088 -0.15568402 -0.26736913  1.01603786 -0.54849908]
 [ 0.19978715  0.90378081  0.12704016  0.73941304  0.38610682]]
```

[28]:
```
#we can also use numpy for statistical tools on arrays
print(np.mean(rand_mat))
print(np.std(rand_mat2))
```

```
0.5416176514357389
1.0005359897931545
```

[29]:
```
print(np.min(rand_mat))
print(np.max(rand_mat2))
```

```
0.021554355328848085
2.592821996203116
```

[31]:
```
# how do we access entries in numpy vector
rand_vec=np.random.randn(19)
print(rand_vec)
print('')
print(rand_vec[6])
```

```
[-1.06537087  0.9062574   0.70487132 -0.96276163  0.44197569  0.98556803
  1.74740933 -0.69744393 -1.2612578   0.7951034  -0.91465163  0.41791038
  0.05869775 -0.5696025  -0.42082141 -0.04771631 -0.65829216  0.13275246
 -0.28051368]
```

```
1.7474093309973002
```

numpy-intro-part3-class

October 5, 2024

[3]:
```
import numpy as np
```

[9]:
```
# how do we access entries in a numpy vector
rand_vec=np.random.rand(19)
print(rand_vec)
print(rand_vec[4])
```

```
[0.92571482 0.49755265 0.39998119 0.07877513 0.98093024 0.85484515
 0.16199059 0.84263044 0.89194671 0.91686246 0.03103067 0.68878533
 0.14305289 0.62031372 0.62797773 0.69853633 0.46764185 0.46297885
 0.88172779]
0.9809302447259455
```

[10]:
```
# we can access multiple entries at once using :
print(rand_vec[4:9])
```

```
[0.98093024 0.85484515 0.16199059 0.84263044 0.89194671]
```

[11]:
```
# we can also access multiple non-consecutive entries using np.arange
print(np.arange(0,15,3))
print(rand_vec[np.arange(0,15,3)])
```

```
[ 0  3  6  9 12]
[0.92571482 0.07877513 0.16199059 0.91686246 0.14305289]
```

[12]:
```
# what about matrices
rand_mat=np.random.rand(5,5)
print(rand_mat)
print(rand_mat[1][2])
print(rand_mat[1,2])
```

```
[[0.09323531 0.12728828 0.52843048 0.78773523 0.32181119]
 [0.66898479 0.73449362 0.00592468 0.62746032 0.51244137]
 [0.36171297 0.31426886 0.27444488 0.02173348 0.34220812]
 [0.58006888 0.60795789 0.8779869  0.82395548 0.53420109]
 [0.1469786  0.62784201 0.9579042  0.3979218  0.37770637]]
0.005924675173620497
0.005924675173620497
```

[13]:
```
print(rand_mat[0:2,1:3])
```

```
[[0.12728828 0.52843048]
 [0.73449362 0.00592468]]
```

[14]:
```
# lets change some values in an array
print(rand_vec)
rand_vec[3:5]=4
print('')
print(rand_vec)
rand_vec[3:5]=[1,2]
print('')
print(rand_vec)
```

```
[0.92571482 0.49755265 0.39998119 0.07877513 0.98093024 0.85484515
 0.16199059 0.84263044 0.89194671 0.91686246 0.03103067 0.68878533
 0.14305289 0.62031372 0.62797773 0.69853633 0.46764185 0.46297885
 0.88172779]

[0.92571482 0.49755265 0.39998119 4.         4.         0.85484515
 0.16199059 0.84263044 0.89194671 0.91686246 0.03103067 0.68878533
 0.14305289 0.62031372 0.62797773 0.69853633 0.46764185 0.46297885
 0.88172779]

[0.92571482 0.49755265 0.39998119 1.         2.         0.85484515
 0.16199059 0.84263044 0.89194671 0.91686246 0.03103067 0.68878533
 0.14305289 0.62031372 0.62797773 0.69853633 0.46764185 0.46297885
 0.88172779]
```

[15]:
```
print(rand_mat)
rand_mat[1:3,3:5]=0
print('')
print(rand_mat)
```

```
[[0.09323531 0.12728828 0.52843048 0.78773523 0.32181119]
 [0.66898479 0.73449362 0.00592468 0.62746032 0.51244137]
 [0.36171297 0.31426886 0.27444488 0.02173348 0.34220812]
 [0.58006888 0.60795789 0.8779869  0.82395548 0.53420109]
 [0.1469786  0.62784201 0.9579042  0.3979218  0.37770637]]

[[0.09323531 0.12728828 0.52843048 0.78773523 0.32181119]
 [0.66898479 0.73449362 0.00592468 0.         0.        ]
 [0.36171297 0.31426886 0.27444488 0.         0.        ]
 [0.58006888 0.60795789 0.8779869  0.82395548 0.53420109]
 [0.1469786  0.62784201 0.9579042  0.3979218  0.37770637]]
```

[16]:
```
print(rand_mat[0:2,0:3])
print('')
sub_mat=rand_mat[0:2,0:3]
print(sub_mat)
print('')
sub_mat[:]=2
print(sub_mat) # note: we are adding the columns present at index number 0,1,2
```

```
[[0.09323531 0.12728828 0.52843048]
 [0.66898479 0.73449362 0.00592468]]

[[0.09323531 0.12728828 0.52843048]
 [0.66898479 0.73449362 0.00592468]]

[[2. 2. 2.]
 [2. 2. 2.]]
```

[17]:
```
print(rand_mat)
```

```
[[2.         2.         2.         0.78773523 0.32181119]
 [2.         2.         2.         0.         0.        ]
 [0.36171297 0.31426886 0.27444488 0.         0.        ]
 [0.58006888 0.60795789 0.8779869  0.82395548 0.53420109]
 [0.1469786  0.62784201 0.9579042  0.3979218  0.37770637]]
```

[18]:
```
rand_mat2=rand_mat[0:2,0:3].copy()
rand_mat2[:]=99
print(rand_mat2)
print(rand_mat)
```

```
[[99. 99. 99.]
 [99. 99. 99.]]
[[2.         2.         2.         0.78773523 0.32181119]
 [2.         2.         2.         0.         0.        ]
 [0.36171297 0.31426886 0.27444488 0.         0.        ]
 [0.58006888 0.60795789 0.8779869  0.82395548 0.53420109]
 [0.1469786  0.62784201 0.9579042  0.3979218  0.37770637]]
```

[ ]:

and-pandas-series-and-dataframes

October 5, 2024

[3]:
```
import pandas as pd
import numpy as np # numpy is not necessary for pandas,but we will use some np␣
 ↪code in this example
# in general its good pratice to import all packages at the begining
```

[9]:
```
# first lets look at series-think of this as a singel column of a spreadsheet
# each entry in a series corresponds to an individual row in a spreadsheet
# we can convert a series by converting a list, or numpy array

mylist=[5.4,6.1,1.7,99.8]
myarray=np.array(mylist)
```

[10]:
```
myseries1=pd.Series(data=mylist)
print(myseries1)
myseries2=pd.Series(data=myarray)
print(myseries2)
```

```
0     5.4
1     6.1
2     1.7
3    99.8
dtype: float64
0     5.4
1     6.1
2     1.7
3    99.8
dtype: float64
```

[11]:
```
# we can access individual entries the same way as with lists and arrays
print(myseries1[2])
```

```
1.7
```

[13]:
```
# we can add labels to the entries of a series

mylabels=['first','second','third','fourth']
myseries3=pd.Series(data=mylist,index=mylabels)
print(myseries3)
```

```
first      5.4
second     6.1
third      1.7
fourth    99.8
dtype: float64
```

[14]:
```
# we need not be explicit about the entries od pd.series
myseries4=pd.Series(mylist,mylabels)
print(myseries4)
```

```
first      5.4
second     6.1
third      1.7
fourth    99.8
dtype: float64
```

[15]:
```
# we can also access entries using index labels
print(myseries4['second'])
```

```
6.1
```

[19]:
```
# we can do math on series
myseries5=pd.Series([5.5,1.1,8.8,1.6],['first','third','fourth','fifth'])
print(myseries5)
print('')
print(myseries4+myseries5)
```

```
first     5.5
third     1.1
fourth    8.8
fifth     1.6
dtype: float64

fifth       NaN
first      10.9
fourth    108.6
second      NaN
third       2.8
dtype: float64
```

[17]:
```
# we can combine series to create  a dataframe using the concat function
df1=pd.concat([myseries4,myseries5],axis=1,sort=False)
df1
```

[17]:

	0	1
first	5.4	5.5
second	6.1	1.1
third	1.7	8.8

```
fourth  99.8  1.6
```

[18]:
```
# we can create a new dataframe
df2=pd.DataFrame(np.random.randn(5,5))
df2
```

[18]:

	0	1	2	3	4
0	0.827666	0.937177	-1.359387	0.533784	-0.236861
1	-0.623482	-0.620307	-0.956728	-0.685777	-0.287111
2	0.004876	1.186497	0.193192	-1.042031	-0.426312
3	1.066617	-0.344778	2.223597	0.472575	0.266702
4	0.482640	-0.236198	0.111004	0.197922	-0.607358

[22]:
```
# lets give labels to rows and columns
df3=pd.DataFrame(np.random.randn(5,5),index=['first row','second row','third␣
↪row','fourth row','fifth row'],
                 columns=['first col','second col','third col','fourth␣
↪col','fifth col'])
df3
```

[22]:

	first col	second col	third col	fourth col	fifth col
first row	1.572157	-0.621958	1.063805	1.121372	-1.668425
second row	-1.105208	1.287097	0.263949	0.299693	0.255133
third row	-1.625136	1.396537	0.654898	-3.475062	-0.728533
fourth row	-1.774371	-1.485051	-0.997328	-1.320681	0.882620
fifth row	0.719182	1.509969	0.488716	-0.486814	-0.768529

[23]:
```
# we can access individual series in a dataframe
print(df3['second col'])
print('')
print(['third col','first col'])
```

```
first row    -0.621958
second row    1.287097
third row     1.396537
fourth row   -1.485051
fifth row     1.509969
Name: second col, dtype: float64

['third col', 'first col']
```

[ ]:

lab4-pandas-creating-dataframes

October 5, 2024

1 Pandas

Pandas is a package used for managing data.

Pandas main use is that it creates 2 new data types for storing data: series and dataframe.

Think of a pandas dataframe like an excel spreadsheet that is storing some data. One column can have customer name, one column can have product sold name, another column can have price or quantity… Then the rows could be individual sales.

A dataframe is made up of several series. Each column of a dataframe is a series.

We can name each column and row of a dataframe.

A pandas dataframe is very similar to a data.frame in R.

Similar to numpy arrays, a dataframe is a more robust data type for storing data than lists of lists. Dataframes are more flexible than numpy arrays.

A numpy array can create a matrix with all entries of the same data type. In a dataframe each column can have its own datatype.

That's not to say numpy arrays aren't useful. It is often easiest to convert some subset of a dataframe to a numpy array and then use that to do some math.

Pandas also has SQL-like functions for merging, joining, and sorting dataframes.

[1]:
```
import pandas as pd
import numpy as np  # numpy is not necessary for pandas, but we will use some␣
 ↪np code in this example
# in general it's good practice to import all pacakages at the beginning
```

2 Combining data frames

The ways dataframes are combined in pandas is similar to SQL

We will examine 3 methods for combining dataframes

1. concat
2. join
3. merge

[7]:
```
df1 = pd.DataFrame({"customer":['101','102','103','104'],
                    'category': ['cat2','cat2','cat1','cat3'],
                    'important': ['yes','no','yes','yes'],
                    'sales': [123,52,214,663]},index=[0,1,2,3])
df1
```

[7]:

	customer	category	important	sales
0	101	cat2	yes	123
1	102	cat2	no	52
2	103	cat1	yes	214
3	104	cat3	yes	663

[6]:
```
df2 = pd.DataFrame({"customer":['101','103','104','105'],
                    'color': ['yellow','green','green','blue'],
                    'distance': [12,9,44,21],
                    'sales': [123,214,663,331]},index=[4,5,6,7])
df2
```

[6]:

	customer	color	distance	sales
4	101	yellow	12	123
5	103	green	9	214
6	104	green	44	663
7	105	blue	21	331

[10]:
```
df3 = pd.DataFrame({'Q1': [101,102,103],
                    'Q2': [201,202,203]},
                   index=['I0','I1','I2'])


df3
```

[10]:

	Q1	Q2
I0	101	201
I1	102	202
I2	103	203

[11]:
```
df4 = pd.DataFrame({'Q3': [301,302,303],
                    'Q4': [401,402,403]},
                   index=['I0','I2','I3'])
df4
```

[11]:

	Q3	Q4
I0	301	401
I2	302	402
I3	303	403

[12]:
```
# if some series has multiple of the same value then we can group all the␣
 ↪unique entries together
mydict = {'customer': ['Customer 1','Customer␣
 ↪1','Customer2','Customer2','Customer3','Customer3'],
          'product1': [1.1,2.1,3.8,4.2,5.5,6.9],
          'product2': [8.2,9.1,11.1,5.2,44.66,983]}
df6 = pd.DataFrame(mydict,index=['Purchase 1','Purchase 2','Purchase␣
 ↪3','Purchase 4','Purchase 5','Purchase 6'])
df6
```

[12]:

	customer	product1	product2
Purchase 1	Customer 1	1.1	8.20
Purchase 2	Customer 1	2.1	9.10
Purchase 3	Customer2	3.8	11.10
Purchase 4	Customer2	4.2	5.20
Purchase 5	Customer3	5.5	44.66
Purchase 6	Customer3	6.9	983.00

lab2-creating-a-csv-file

October 5, 2024

[6]:
```
# How to write a CSV file in Python?
# To export (or write) data to a CSV file in Python, you typically use the csv␣
 ↪module.
import csv



data = [
    ['Name', 'Age', 'City'],
    ['John Doe', 30, 'New York'],
    ['Jane Smith', 25, 'Los Angeles'],
    ['Bob Johnson', 45, 'Chicago']
]

# Specify the file name
file_name = 'employee.csv'

# Writing to the CSV file
with open(file_name, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print(f'{file_name} has been created with data.')
```

```
employee.csv has been created with data.
```

[7]:
```
# How to write and append a CSV file in Python?
# You can write and append to a CSV file using the csv module.
# To append data, open the file in append mode ('a') instead of write mode␣
 ↪('w'):

import csv

# Data to append
new_data = [
    ['Alice Brown', 35, 'Boston'],
    ['Eve Green', 28, 'San Francisco']
```

```
]

# Specify the file name
file_name = 'employee.csv'

# Append data to CSV file
with open(file_name, mode='a', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(new_data)

print(f'New data appended to CSV file {file_name}')
```

```
New data appended to CSV file employee.csv
```

[11]:
```
# How to write dict to CSV in Python?
import csv

data = [
    {'Name': 'John Doe', 'Age': 30, 'City': 'New York'},
    {'Name': 'Jane Smith', 'Age': 25, 'City': 'Los Angeles'},
    {'Name': 'Bob Johnson', 'Age': 45, 'City': 'Chicago'}
]

# Specify the file name
file_name = 'employee_dict.csv'

# Write dictionary to CSV file
with open(file_name,mode='w', newline='') as csvfile:
    fields = ['Name', 'Age', 'City']
    writer = csv.DictWriter(csvfile, fieldnames=fields)
    writer.writeheader()
    writer.writerows(data)

print(f'Dictionary data created to CSV file {file_name}')
```

```
Dictionary data created to CSV file employee_dict.csv
```

[1]:
```
import numpy as np
import pandas as pd # pandas library to work with dataframes
```

[3]:
```
df = pd.read_csv('Iris_data_sample .csv')
df
```

[3]:

	Unnamed: 0	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	NaN	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	

```
3             4            ??           3.1           1.5           0.2
4             5             5           3.6           ###           0.2
..          …             …           …           …           …
145         146           6.7           3.0           5.2           2.3
146         147           6.3           2.5             5           1.9
147         148           6.5           3.0           5.2           2.0
148         149           6.2           3.4           5.4           2.3
149         150           5.9           3.0           5.1           1.8

            Species
0       Iris-setosa
1               NaN
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              …
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

[15]:
```
df.head()
```

[15]:
```
   Unnamed: 0 SepalLengthCm  SepalWidthCm PetalLengthCm  PetalWidthCm  \
0           1           5.1           3.5           1.4           0.2
1           2           4.9           NaN           1.4           0.2
2           3           4.7           3.2           1.3           0.2
3           4            ??           3.1           1.5           0.2
4           5             5           3.6           ###           0.2

       Species
0  Iris-setosa
1          NaN
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa
```

[5]:
```
df=pd.read_csv('Iris_data_sample .csv',index_col=0)
df.head()
```

[5]:
```
   SepalLengthCm  SepalWidthCm PetalLengthCm  PetalWidthCm      Species
1           5.1           3.5           1.4           0.2  Iris-setosa
2           4.9           NaN           1.4           0.2          NaN
3           4.7           3.2           1.3           0.2  Iris-setosa
```

```
4            ??           3.1           1.5           0.2  Iris-setosa
5             5           3.6           ###           0.2  Iris-setosa
```

[18]:
```
df=pd.read_csv('Iris_data_sample .csv',index_col=0,na_values=["??"])
df
```

[18]:
```
     SepalLengthCm  SepalWidthCm PetalLengthCm  PetalWidthCm         Species
1              5.1           3.5           1.4           0.2     Iris-setosa
2              4.9           NaN           1.4           0.2             NaN
3              4.7           3.2           1.3           0.2     Iris-setosa
4              NaN           3.1           1.5           0.2     Iris-setosa
5              5.0           3.6           ###           0.2     Iris-setosa
..             …             …             …             …               …
146            6.7           3.0           5.2           2.3  Iris-virginica
147            6.3           2.5             5           1.9  Iris-virginica
148            6.5           3.0           5.2           2.0  Iris-virginica
149            6.2           3.4           5.4           2.3  Iris-virginica
150            5.9           3.0           5.1           1.8  Iris-virginica

[150 rows x 5 columns]
```

[20]:
```
df=pd.read_csv('Iris_data_sample .csv',index_col=0,na_values=["??","###"])
df.head()
```

[20]:
```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
1            5.1           3.5            1.4           0.2  Iris-setosa
2            4.9           NaN            1.4           0.2          NaN
3            4.7           3.2            1.3           0.2  Iris-setosa
4            NaN           3.1            1.5           0.2  Iris-setosa
5            5.0           3.6            NaN           0.2  Iris-setosa
```

[8]:
```
df_xlsx=pd.read_excel('Iris_data_sample .xlsx')
df_xlsx
```

[8]:
```
     Unnamed: 0 SepalLengthCm  SepalWidthCm PetalLengthCm  PetalWidthCm  \
0             1           5.1           3.5           1.4           0.2
1             2           4.9           NaN           1.4           0.2
2             3           4.7           3.2           1.3           0.2
3             4            ??           3.1           1.5           0.2
4             5             5           3.6           ###           0.2
..          …             …             …             …             …
145         146           6.7           3.0           5.2           2.3
146         147           6.3           2.5             5           1.9
147         148           6.5           3.0           5.2           2.0
148         149           6.2           3.4           5.4           2.3
149         150           5.9           3.0           5.1           1.8
```

```
            Species
0       Iris-setosa
1               NaN
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              …
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

[5]:
```
df_xlsx=pd.read_excel('Iris_data_sample .
 ↪xlsx',sheet_name='Iris_data',index_col=0,na_values=["??","###"])
df_xlsx.head()
```

[5]:
```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
1            5.1           3.5            1.4           0.2  Iris-setosa
2            4.9           NaN            1.4           0.2          NaN
3            4.7           3.2            1.3           0.2  Iris-setosa
4            NaN           3.1            1.5           0.2  Iris-setosa
5            5.0           3.6            NaN           0.2  Iris-setosa
```

[5]:
```
df_txt=pd.read_table('Iris_data_sample .txt')
df_txt.head()
```

[5]:
```
   "SepalLengthCm" "SepalWidthCm" "PetalLengthCm" "PetalWidthCm" "Species"
0                 1 1 5.1 3.5 1.4 0.2 "Iris-setosa"
1                   2 2 4.9 3 1.4 0.2 "Iris-setosa"
2                 3 3 4.7 3.2 1.3 0.2 "Iris-setosa"
3                 4 4 4.6 3.1 1.5 0.2 "Iris-setosa"
4                   5 5 5 3.6 1.4 0.2 "Iris-setosa"
```

[6]:
```
df_txt=pd.read_table('Iris_data_sample .txt',delimiter=" ")
df_txt.head()
```

[6]:
```
   Unnamed: 0  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
1           1            5.1           3.5            1.4           0.2
2           2            4.9           3.0            1.4           0.2
3           3            4.7           3.2            1.3           0.2
4           4            4.6           3.1            1.5           0.2
5           5            5.0           3.6            1.4           0.2

       Species
```

```
1  Iris-setosa
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa
5  Iris-setosa
```

[4]:
```
df.shape # Check number of columns and rows in data frame
```

[4]:
```
(150, 6)
```

[5]:
```
df.isnull().values.any() # If there are any null values in data set
```

[5]:
```
False
```

[7]:
```
df.isnull().sum() # If there are any null values in data set
```

[7]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

[ ]:
```

```

lab3-basic-plots-1

October 5, 2024

1 Line Plot

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

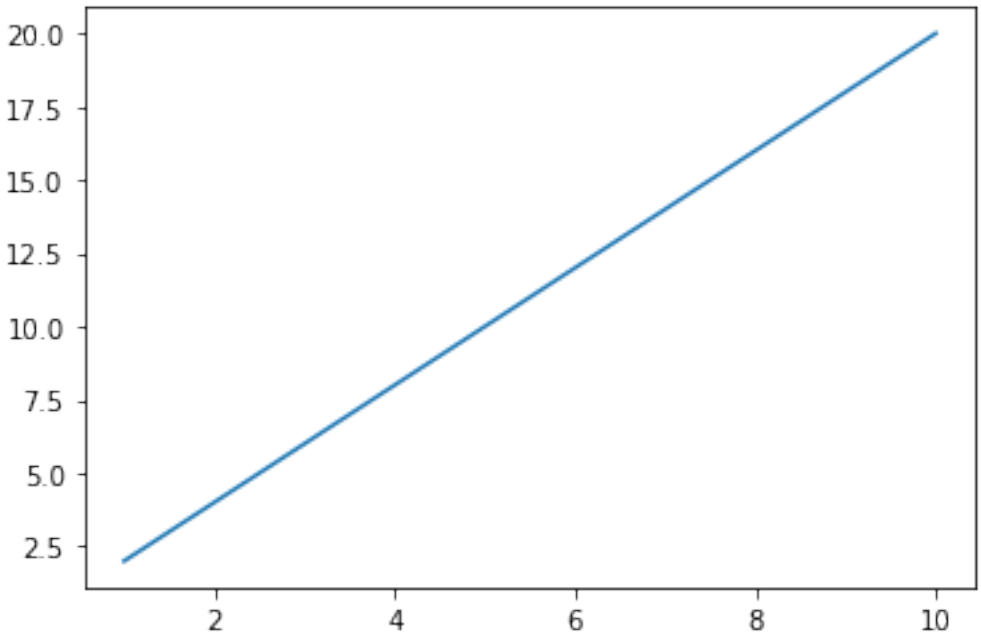
[2]:
```
x=np.arange(1,11)
x
```

[2]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[3]:
```
y=2*x
y
```

[3]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

[5]:
```
plt.plot(x,y)
plt.show()
```



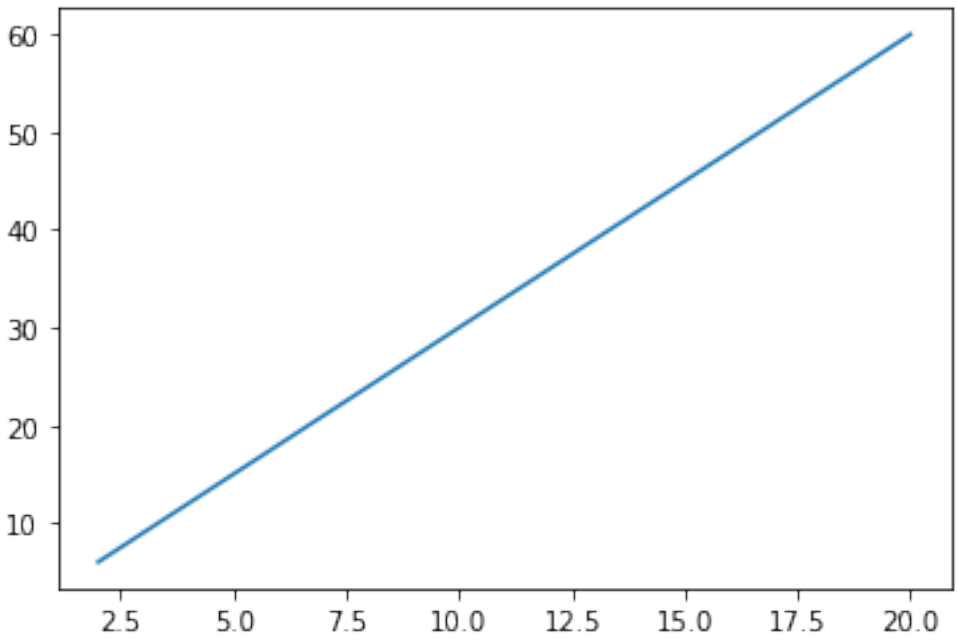
[8]:
```
x=np.arange(2,21)
x
```

[8]:
```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20])
```

[9]:
```
y=3*x
y
```

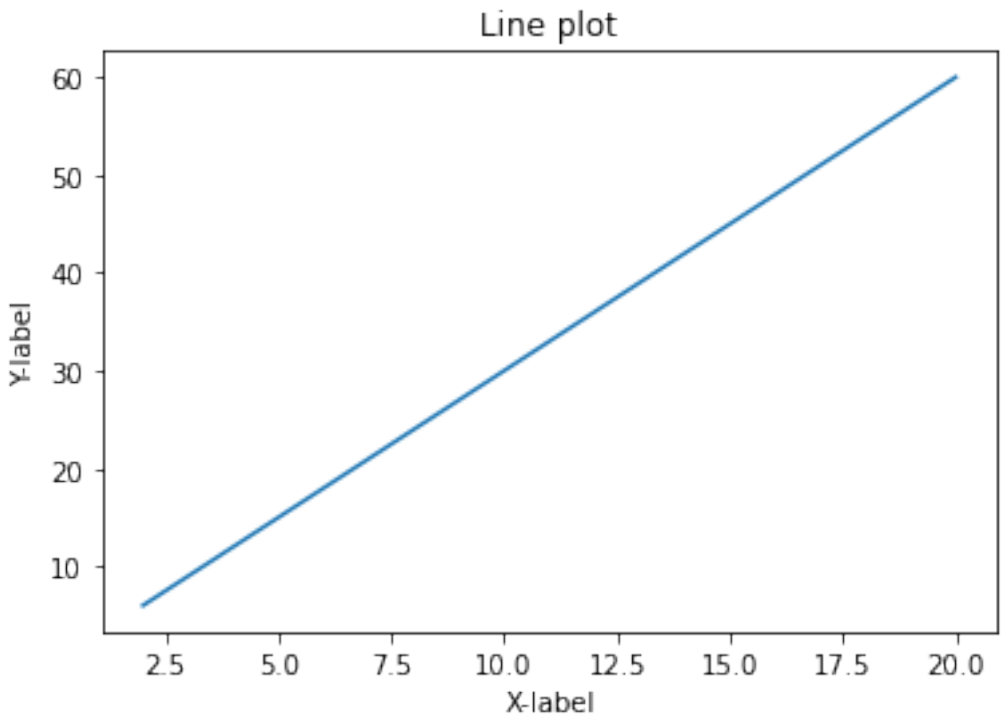
[9]:
```
array([ 6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54,
       57, 60])
```

[10]:
```
plt.plot(x,y)
plt.show()
```

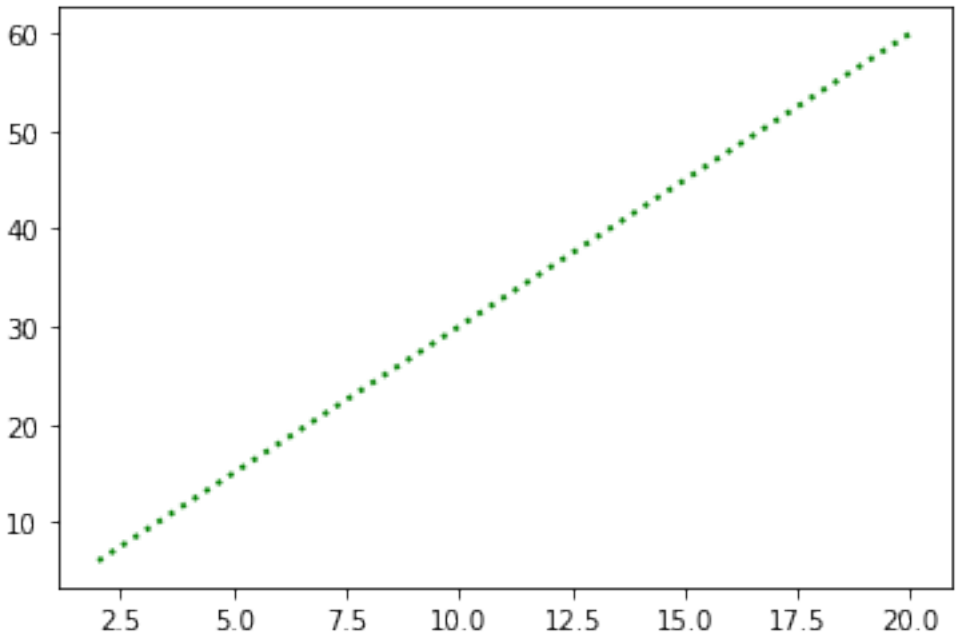


[10]:
```
plt.plot(x,y)
plt.title("Line plot")
plt.xlabel("X-label")
plt.ylabel("Y-label")
```

[10]:
```
Text(0, 0.5, 'Y-label')
```

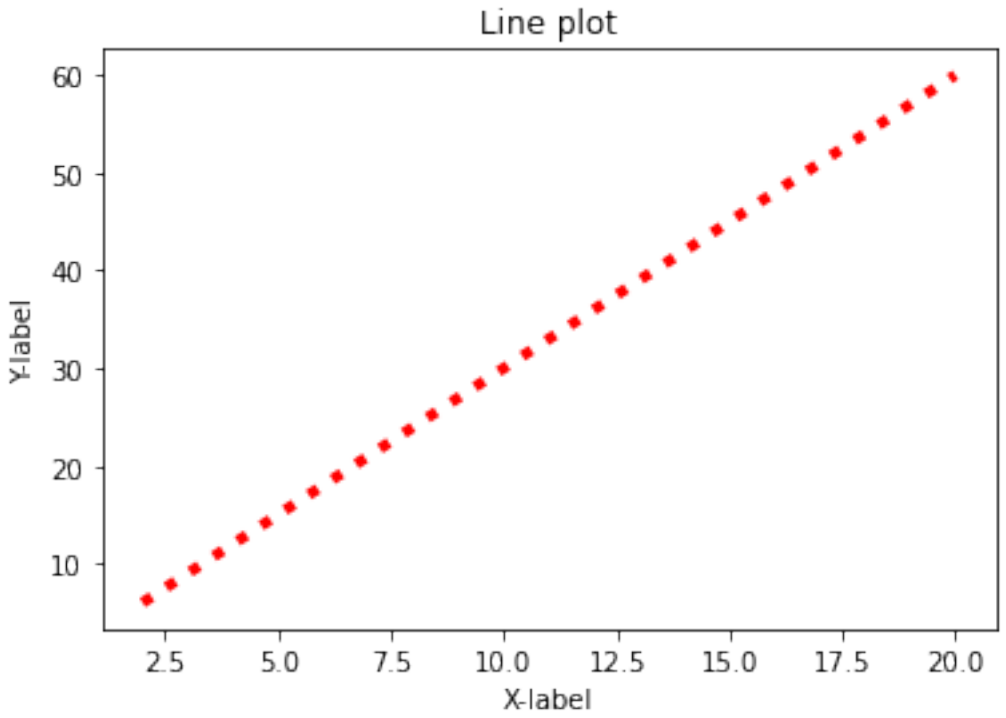


[11]:
```
plt.plot(x,y,color='g',linestyle=':',linewidth=2)
plt.show()
```



[12]:
```
plt.plot(x,y,color='r',linestyle=':',linewidth=4)
plt.title("Line plot")
plt.xlabel("X-label")
plt.ylabel("Y-label")
```

[12]:
```
Text(0, 0.5, 'Y-label')
```



1.1 Adding 2 lines in the same plot

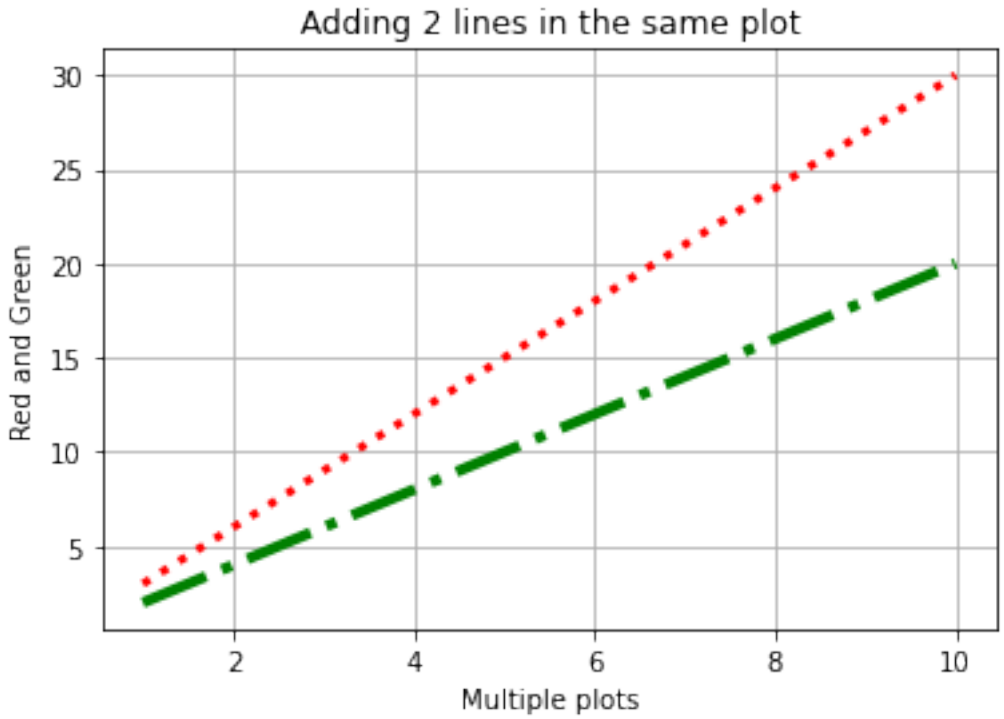
[14]:
```
x=np.arange(1,11)
x
```

[14]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[15]:
```
y1=2*x
y2=3*x
```

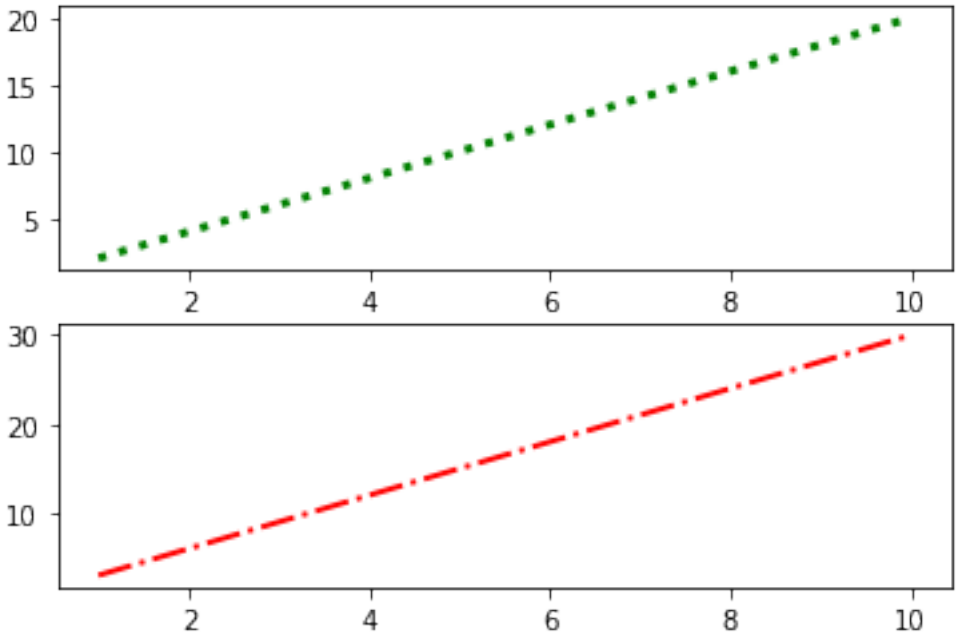
[19]:
```
plt.plot(x,y1,color='g',linestyle='-.',linewidth=4)
plt.plot(x,y2,color='r',linestyle=':',linewidth=3)
plt.title("Adding 2 lines in the same plot")
plt.xlabel("Multiple plots")
plt.ylabel("Red and Green")
plt.grid(True)
```

```
plt.show()
```



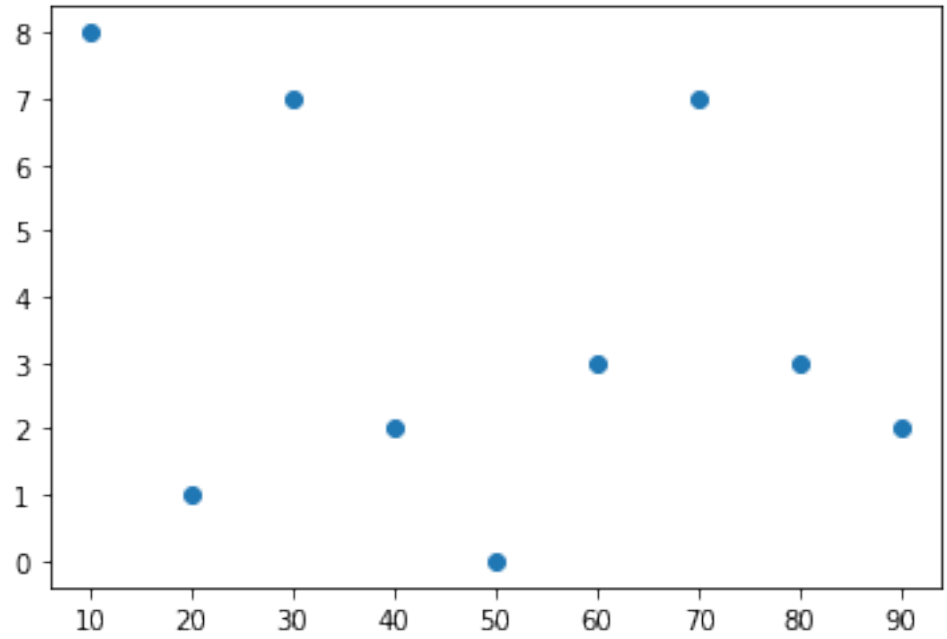
1.2 Subplots

[21]:
```
x=np.arange(1,11)
y1=2*x
y2=3*x
plt.subplot(2,1,1)
plt.plot(x,y1,color='g',linestyle=':',linewidth=3)

plt.subplot(2,1,2)
plt.plot(x,y2,color='r',linestyle='-.',linewidth=2)

plt.show()
```



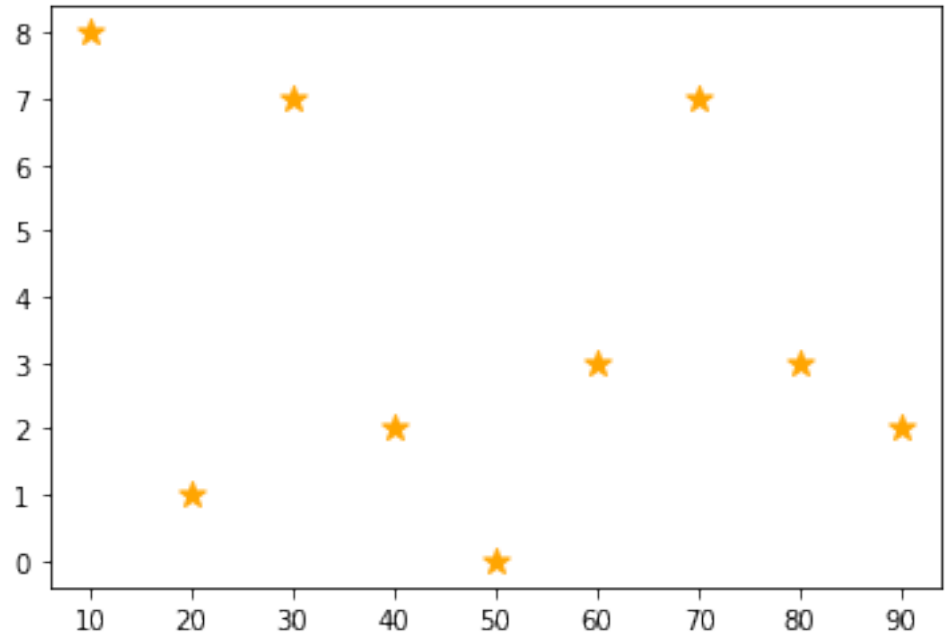
1.3 Scatter Plot

[22]:
```
x=[10,20,30,40,50,60,70,80,90]
y=[8,1,7,2,0,3,7,3,2]

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



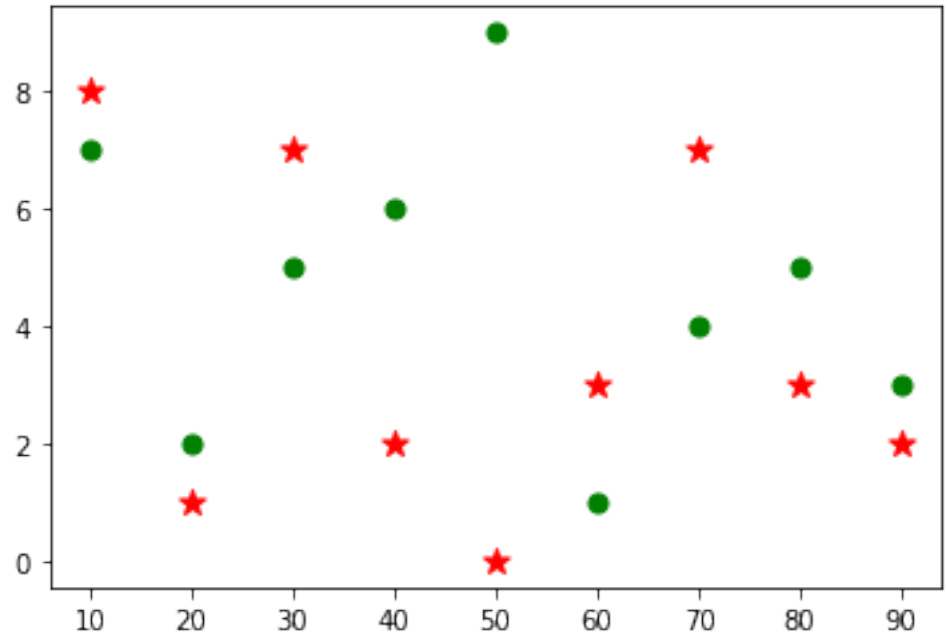
1.4 Adding marker in the scatter plot

[23]:
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]

plt.scatter(x,a,marker='*',c='orange',s=100)
plt.show()
```



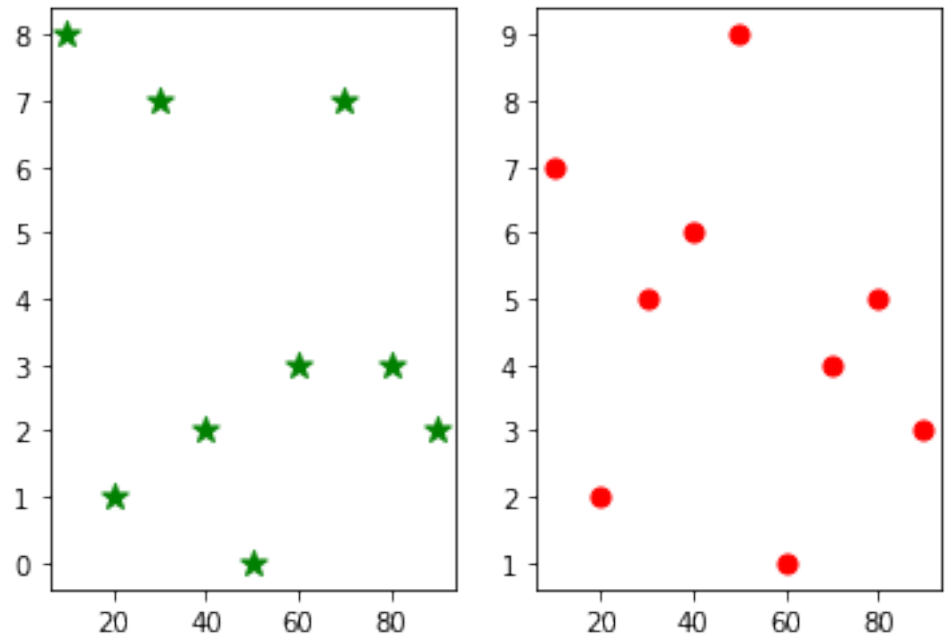
1.5 Adding 2 markers in the same plot

[24]:
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]
b=[7,2,5,6,9,1,4,5,3]

plt.scatter(x,a,marker='*',c='r',s=100)
plt.scatter(x,b,marker='.',c='g',s=200)
plt.show()
```



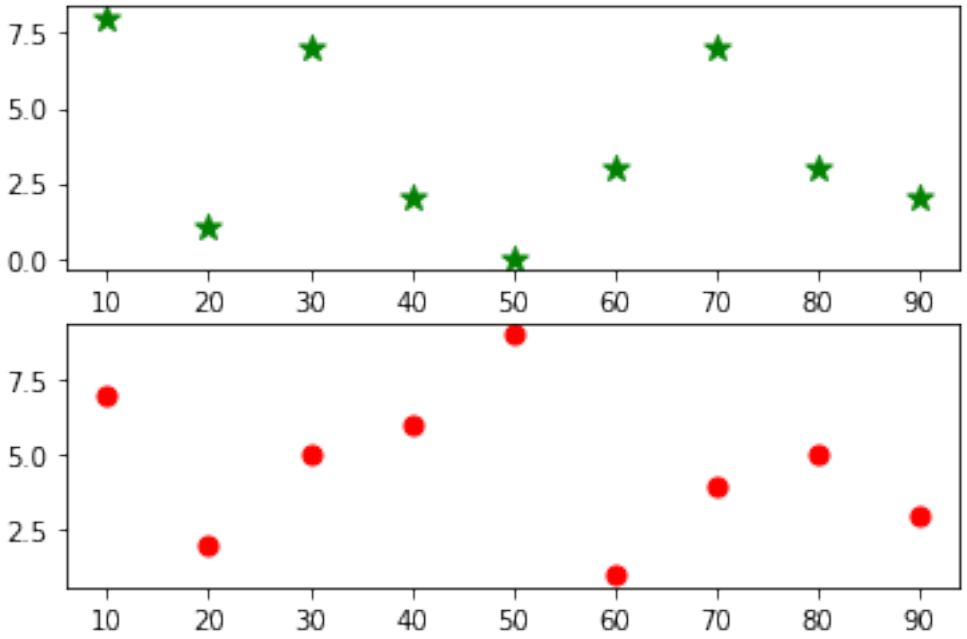
1.6 Adding Subplots

[26]:
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]
b=[7,2,5,6,9,1,4,5,3]

plt.subplot(1,2,1)
plt.scatter(x,a,marker='*',c='g',s=100)

plt.subplot(1,2,2)
plt.scatter(x,b,marker='.',c='r',s=200)

plt.show()
```



[ ]:
```
Instead of having them in 2 columns if you want in 2 rows then
```

[63]:
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]
b=[7,2,5,6,9,1,4,5,3]

plt.subplot(2,1,1)
plt.scatter(x,a,marker='*',c='g',s=100)

plt.subplot(2,1,2)
plt.scatter(x,b,marker='.',c='r',s=200)

plt.show()
```


7.5
5.0
2.5
0.0
10
20
30
40
50
60
70
80
90
7.5
5.0
2.5
10
20
30
40
50
60
70
80
90

[ ]:

lab-3-basic-plots-2

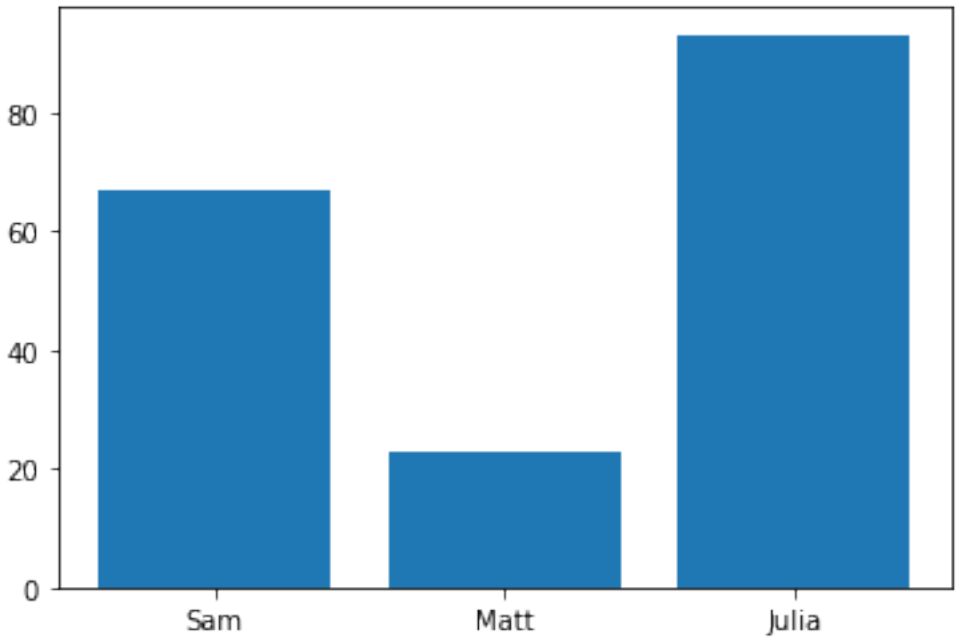
October 5, 2024

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

0.1 Bar Plot

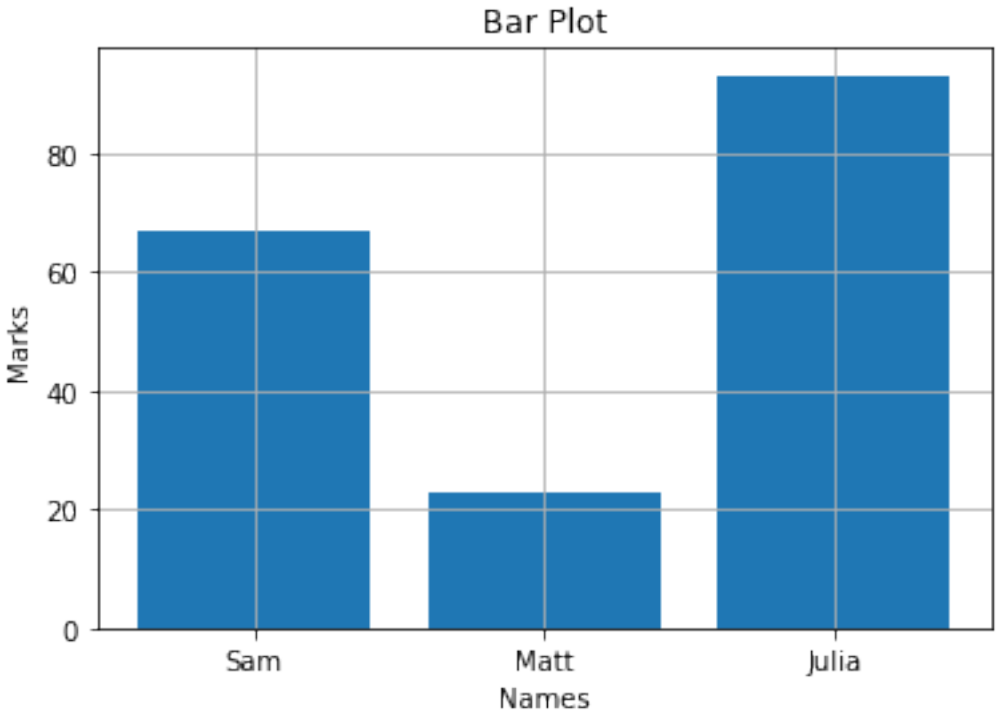
Bar Plot

[2]:
```
student={'Sam':67,'Matt':23,'Julia':93}
Names=list(student.keys())
Values=list(student.values())
plt.bar(Names,Values)
plt.show()
```



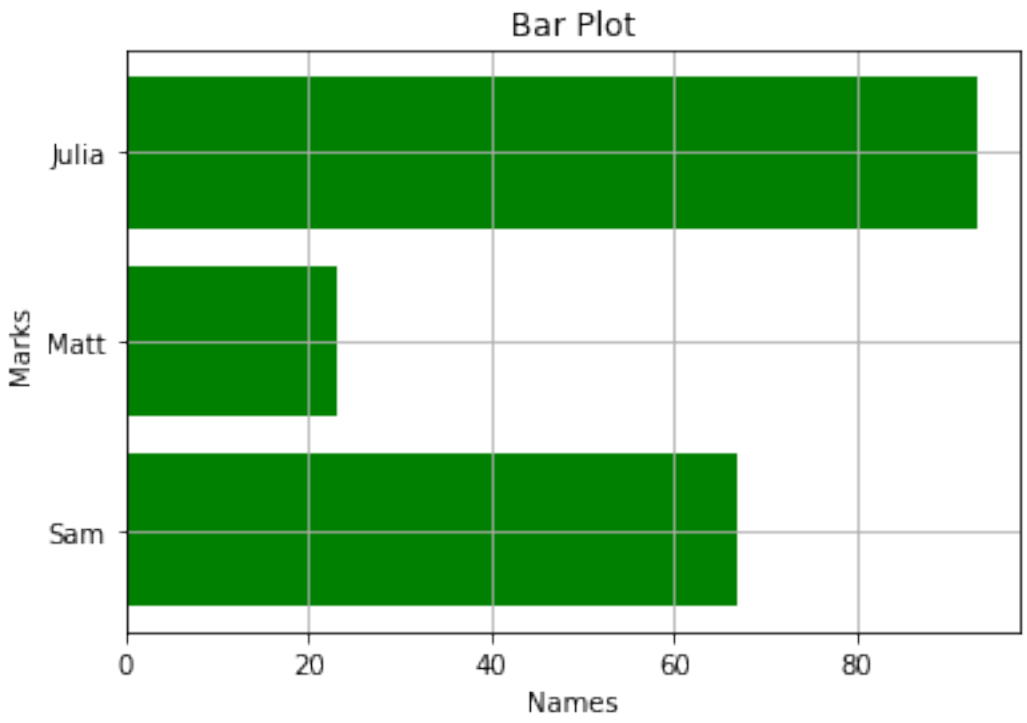
0.2 Adding Titles and Labels

[6]:
```
plt.bar(Names,Values)
plt.title("Bar Plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```



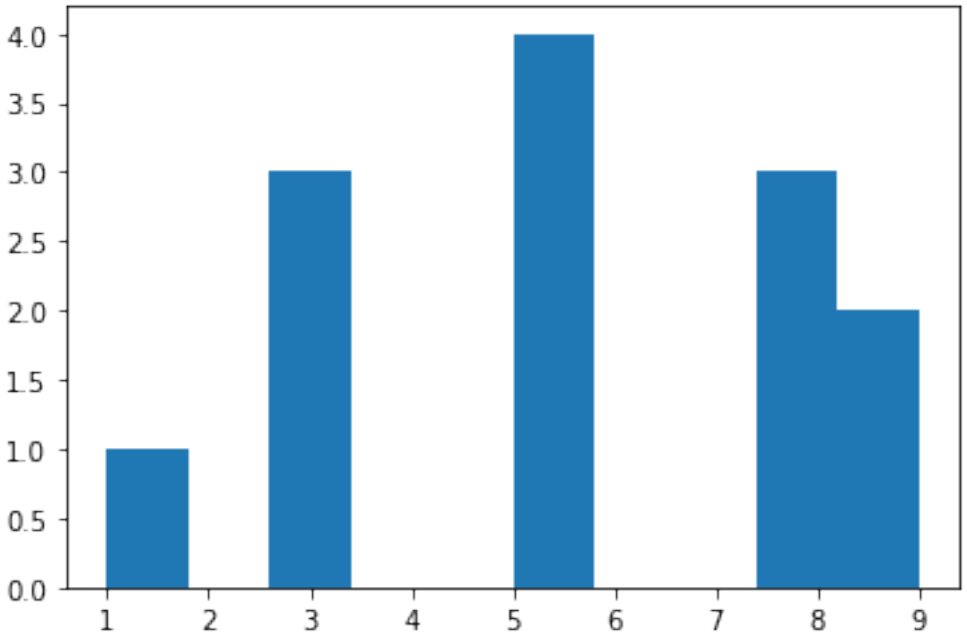
0.3 Horizontal Bar Plot

[7]:
```
plt.barh(Names,Values,color='g')
plt.title("Bar Plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```



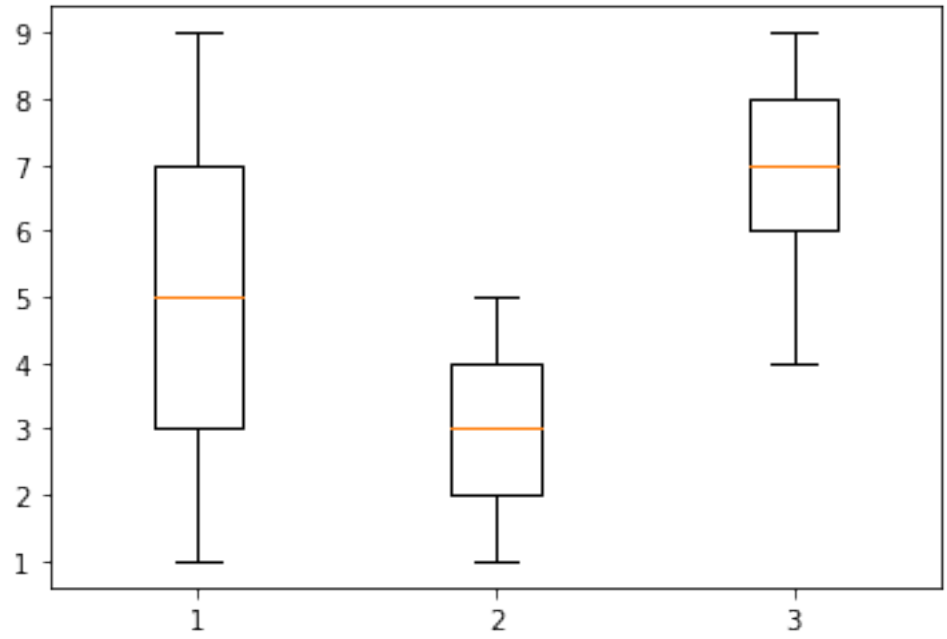
0.4 Histogram

[8]:
```
data=[1,3,3,3,9,9,5,5,5,5,8,8,8]  #creating data
plt.hist(data)                       # Making Histogram
plt.show()
```



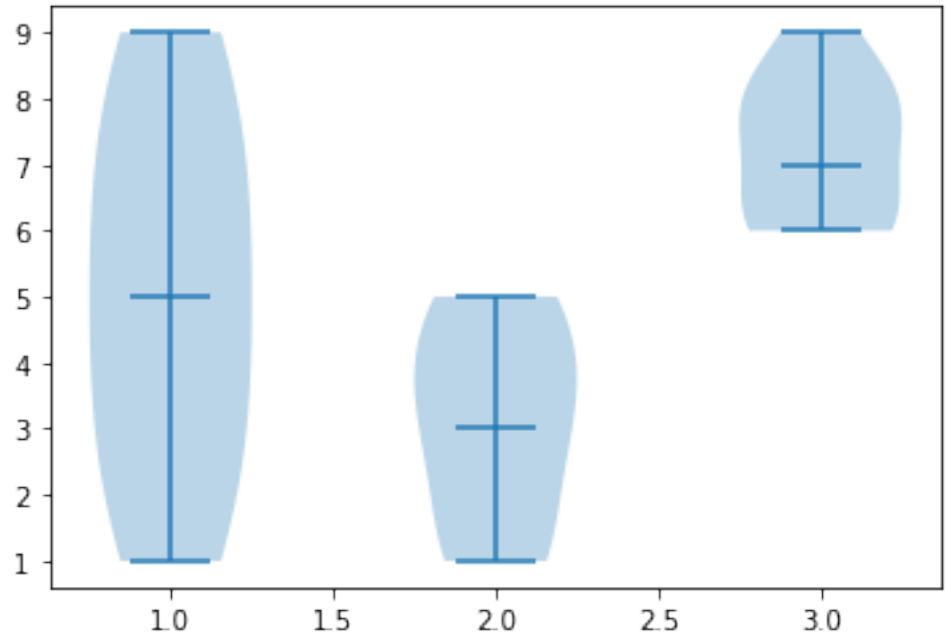
0.5 Box Plot

[10]:
```
one=[1,2,3,4,5,6,7,8,9]  # Creating data
two=[1,2,3,4,5,4,3,2,1]
three=[6,7,8,9,8,7,6,5,4]

data=list([one,two,three])
plt.boxplot(data)  # making plot
plt.show()
```



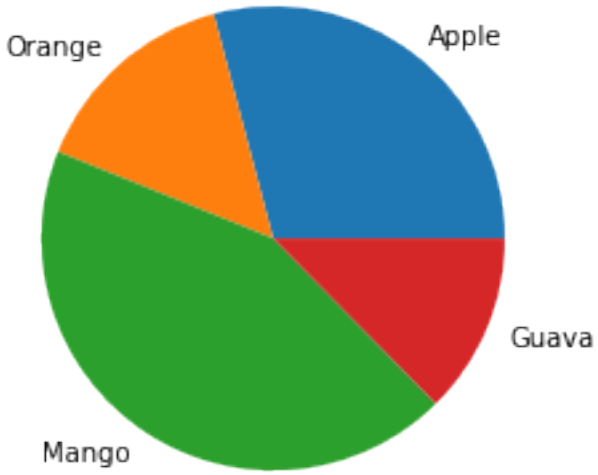
0.6 Violin Plot

[13]:
```
one=[1,2,3,4,5,6,7,8,9]  # Creating data
two=[1,4,3,5,2,3,1,5,4]
three=[6,8,9,7,8,6,8,6,7]

data=list([one,two,three])
plt.violinplot(data,showmedians=True)  # making plot
plt.show()
```



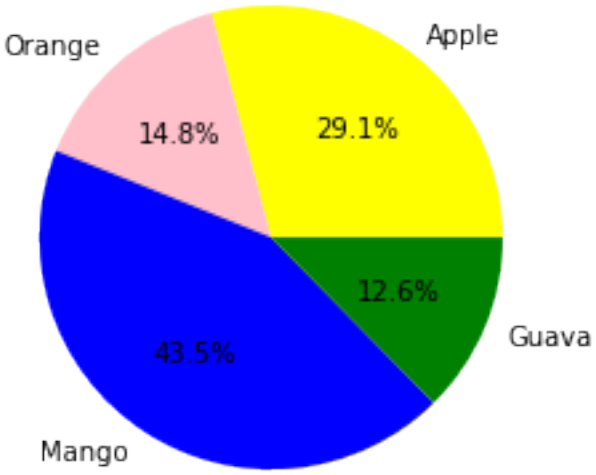
0.7 Pie chart

[15]:
```
fruit=['Apple','Orange','Mango','Guava'] # Creating data
quantity=[67,34,100,29]
plt.pie(quantity,labels=fruit)  # making plot
plt.show()
```



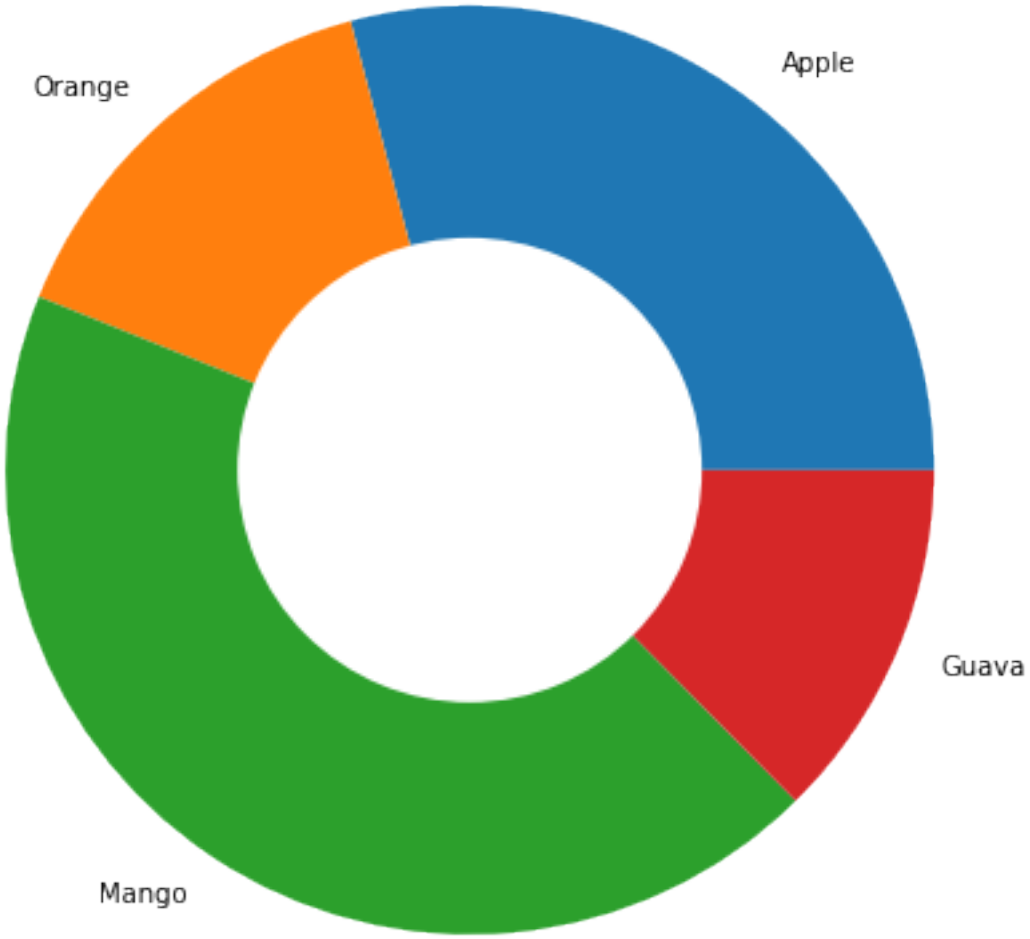
0.8 Changing Aesthetics

[14]:
```
plt.pie(quantity,labels=fruit,autopct='%0.
  ↪1f%%',colors=['yellow','pink','blue','green'])
plt.show()
```

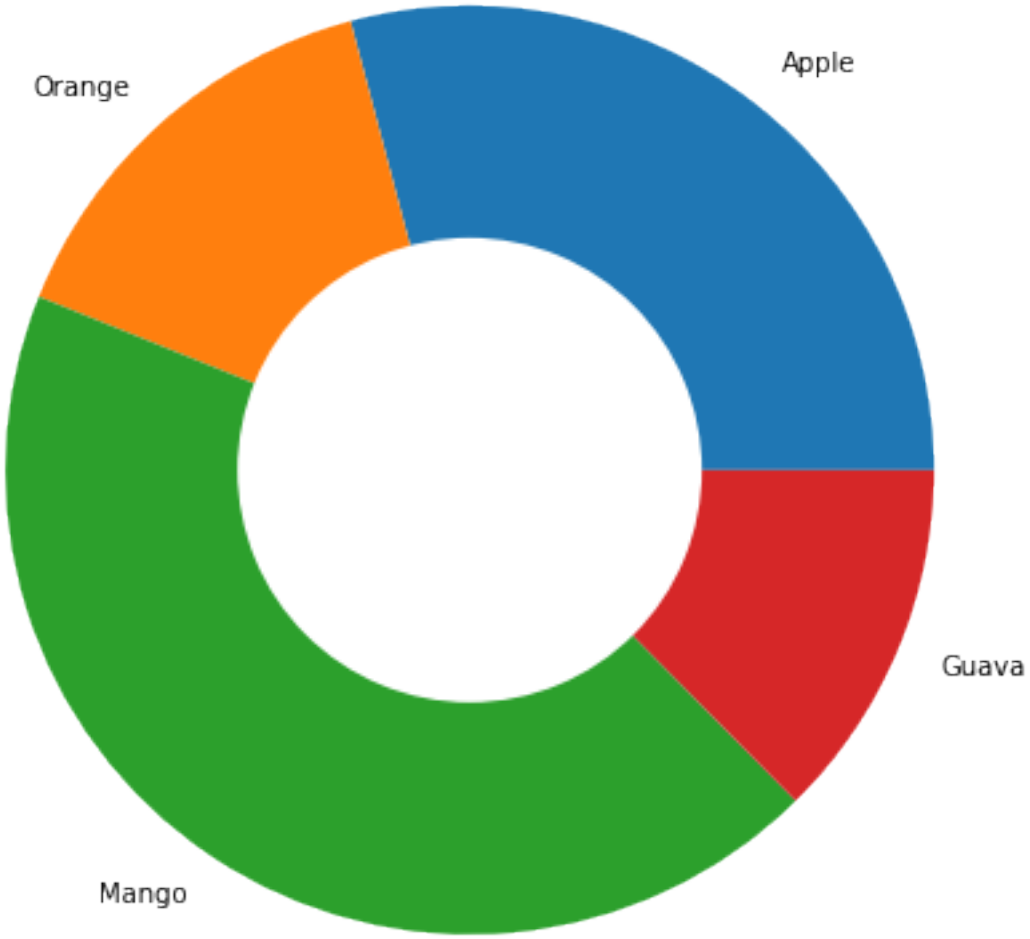


0.9 Donut Chart

[17]:
```
fruit=['Apple','Orange','Mango','Guava'] # Creating data
quantity=[67,34,100,29]
plt.pie(quantity,labels=fruit,radius=2)
plt.pie([2],colors=['w'],radius=1)
plt.show()
```

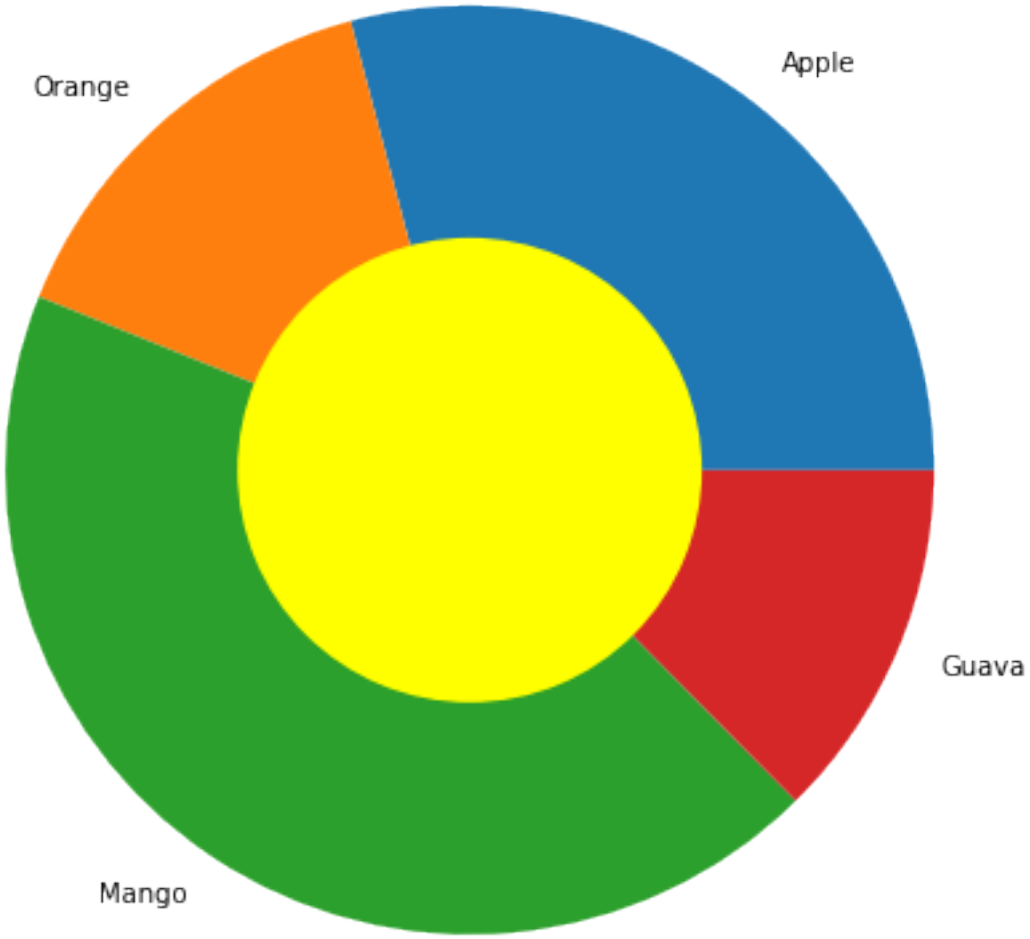


[17]:
```
plt.pie(quantity,labels=fruit,radius=2)
plt.pie([1],colors=['w'],radius=1)
plt.show()
```



[18]:
```
plt.pie(quantity,labels=fruit,radius=2)
plt.pie([2],colors=['yellow'],radius=1)
plt.show
```

[18]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```


Apple
Orange
Guava
Mango

[ ]: