COMP7035

Python for Data Analytics and Artificial Intelligence

Numpy/Matplotlib

Renjie Wan, Xue Wei

17/10/2022

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

What we will learn?

Topic		Hours
I.	Python Fundamentals A. Program control and logic B. Data types and structures C. Function D. File I/O	12
II.	Numerical Computing and Data Visualization Tools and libraries such as A. NumPy **B. Matplotlib** C. Seaborn	9
III.	Exploratory Data Analysis (EDA) with Python Tools and libraries such as A. Pandas B. Sweetviz	9
IV.	Artificial Intelligence and Machine Learning with Python Tools and libraries such as A. Keras B. Scikit-learn	9

Numpy Broadcasting

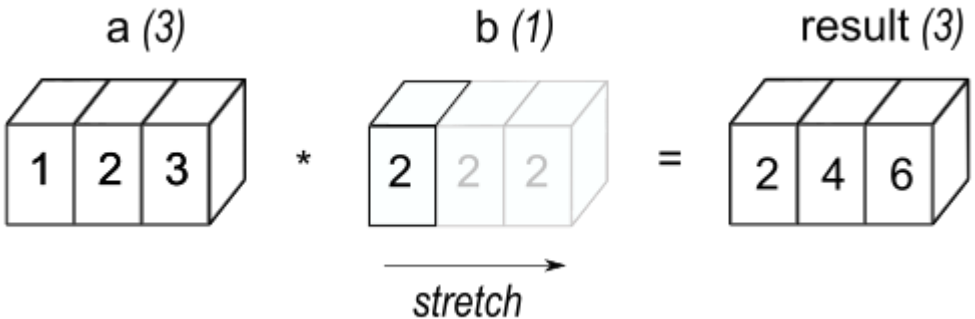
- The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.
- Scalar will be broadcasted to all elements in an array with different shapes.

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([2.,  4.,  6.])
```

With same shapes

```
import numpy as np
a = np.array([1.0, 2.0, 3.0])
b = 2.0
print(a * b)
```

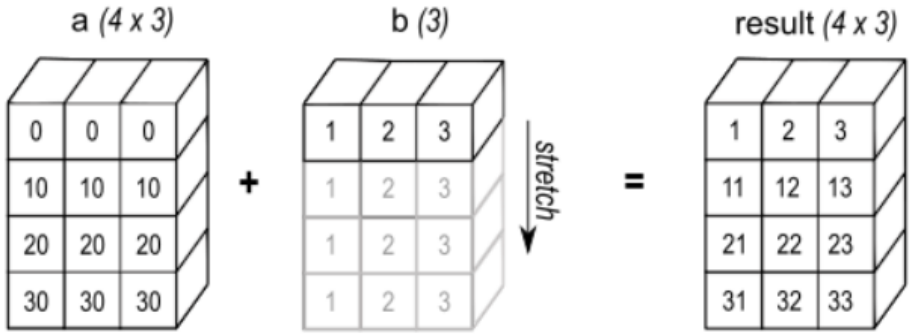
With different shapes



General broadcasting rule

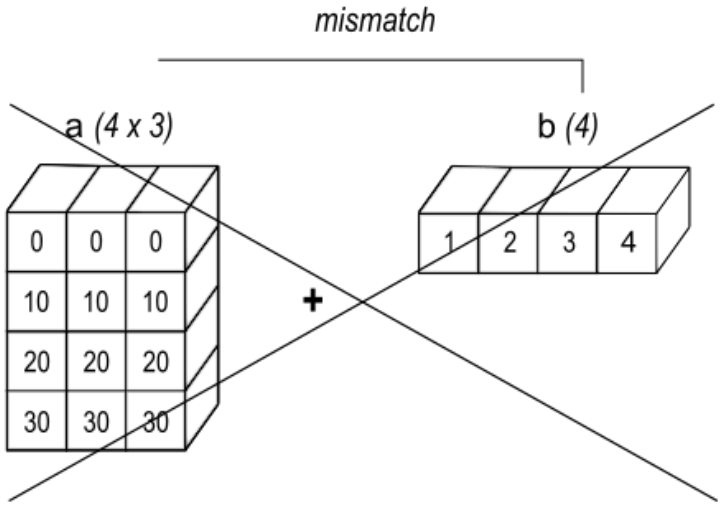
- Broadcasting for two arrays
  - *A one-dimensional array added to a two-dimensional array results in broadcasting if number of 1-d array elements matches the number of 2-d array columns.*

```
import numpy
a = np.array([[ 0.0,  0.0,  0.0],
              [10.0, 10.0, 10.0],
              [20.0, 20.0, 20.0],
              [30.0, 30.0, 30.0]])
b = np.array([1.0, 2.0, 3.0])
print(a + b)
```



General broadcasting rule

- Broadcasting for two arrays
  - *A one-dimensional array added to a two-dimensional array results in broadcasting if number of 1-d array elements matches the number of 2-d array columns.*

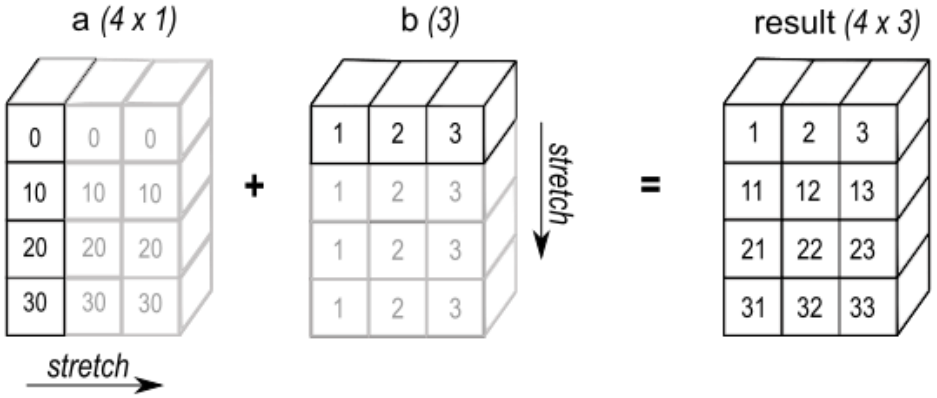


Different column numbers. Not ALLOWED!!!

General broadcasting rule

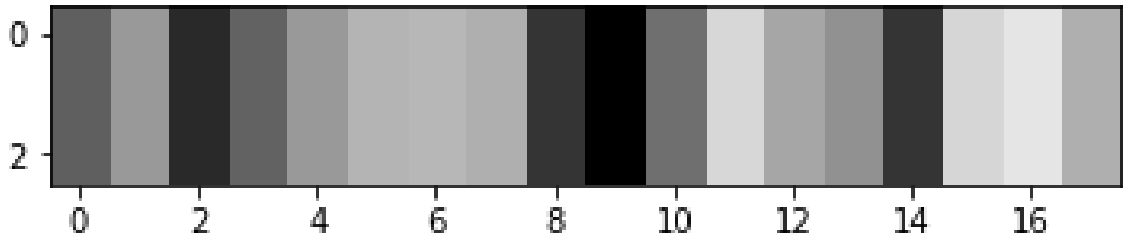
- Broadcasting provides a convenient way of taking the outer product (or any other outer operation) of two arrays. The following example shows an outer addition operation of two 1-d arrays:

```
a = np.array([0.0, 10.0, 20.0, 30.0])
b = np.array([1.0, 2.0, 3.0])
print(a[:, np.newaxis])
a[:, np.newaxis] + b
```



`np.newaxis: Create a new axis for array`

Let us make it more clear

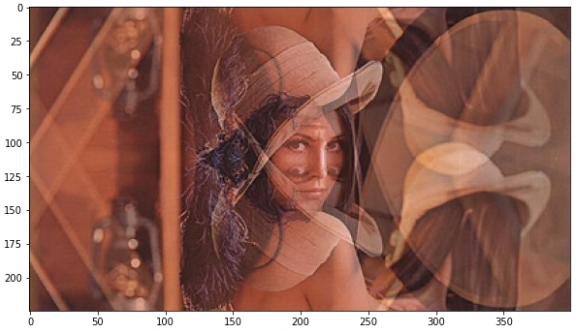



```
img = np.zeros((3, 18, 3))
arr = np.random.random((1, 18, 1))*1000
kkk = (img + arr).astype(np.uint8)
plt.imshow(kkk)
```

Practical examples for broadcasting

- Image blending

After image flip

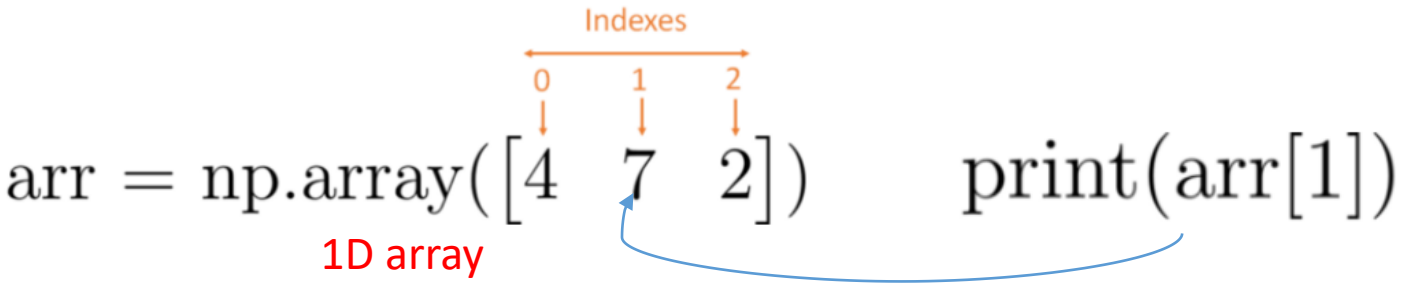


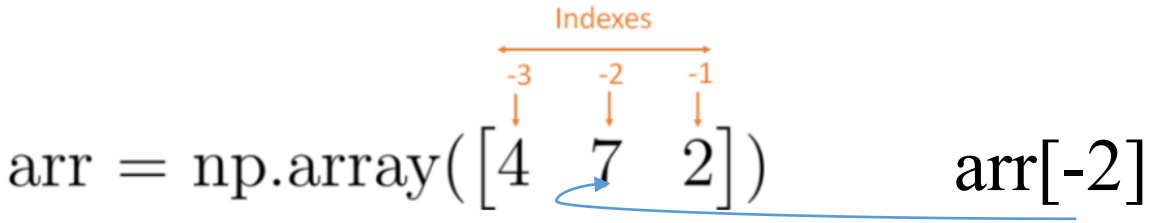
Blended image

```
img = plt.imread('lena.jpg')
img0 = plt.imread('lena.jpg')
img0 = np.flipud(img0)
print(img.dtype)
# uint8
dst = (img * 0.6 + img0 * 0.4).astype(np.uint8)    # Blending them in
plt.figure(figsize=(10, 10))
plt.imshow(dst)
```

Array Element Access

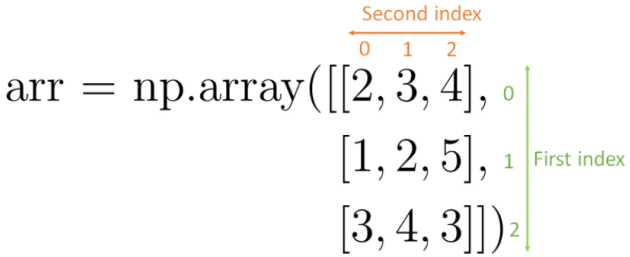
- From 1D to 2D

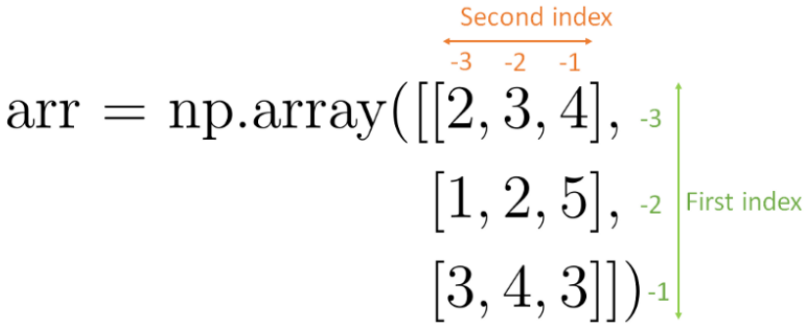


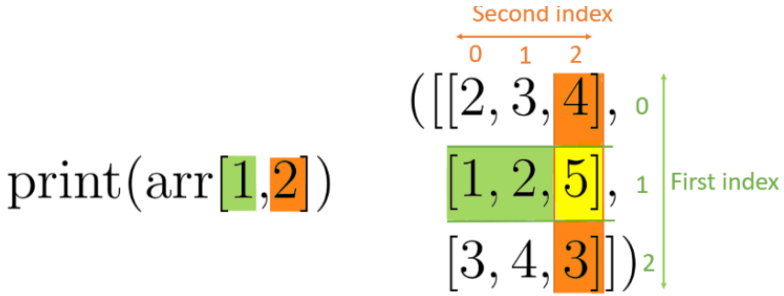


Array Element Access

- From 1D to 2D

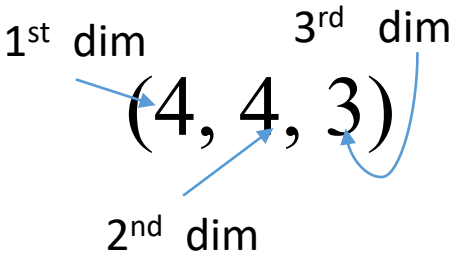


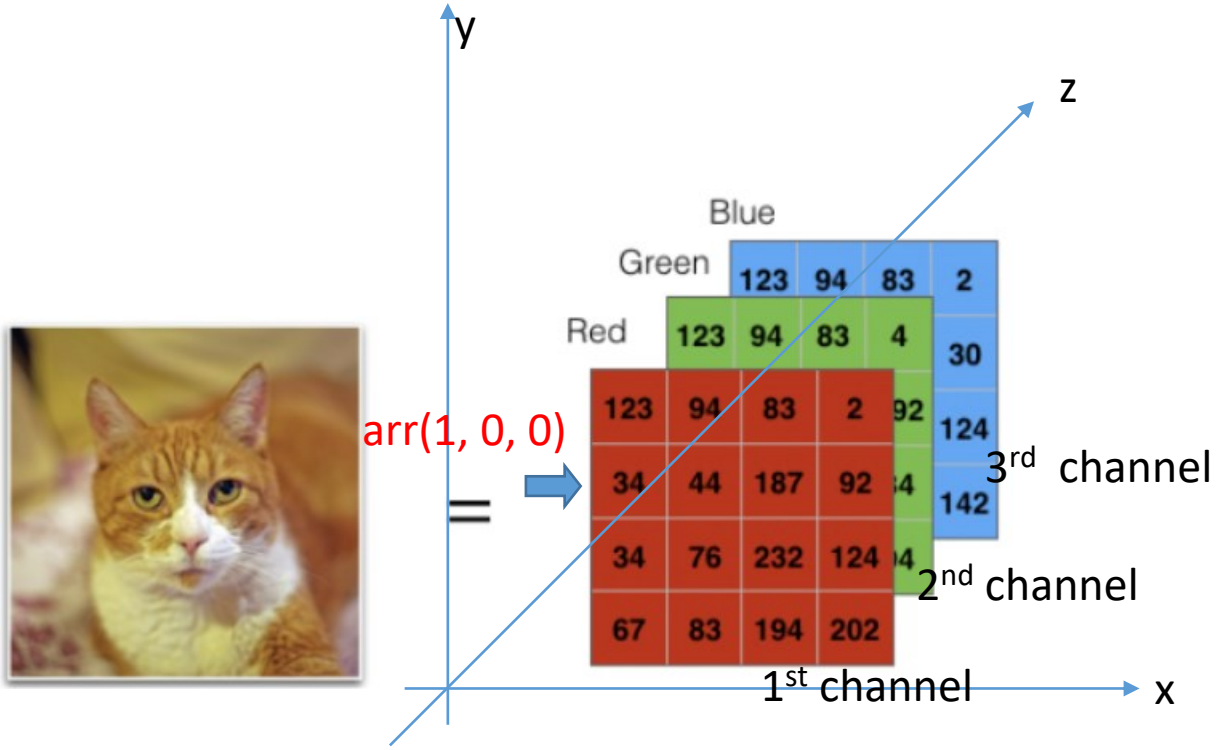




print(arr[-2, -3])

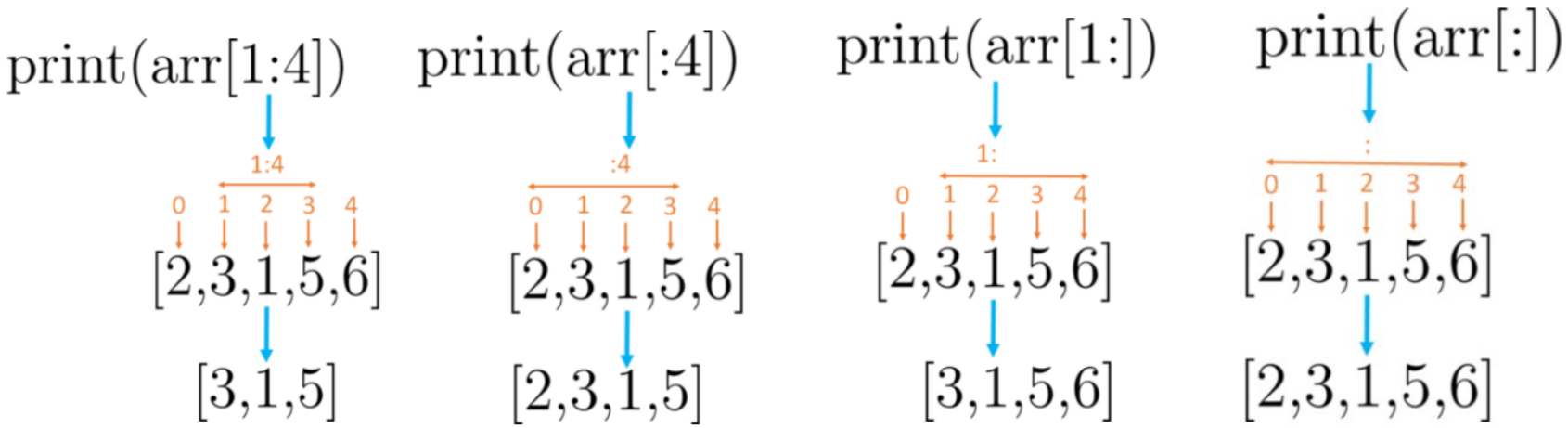
Array Element Access





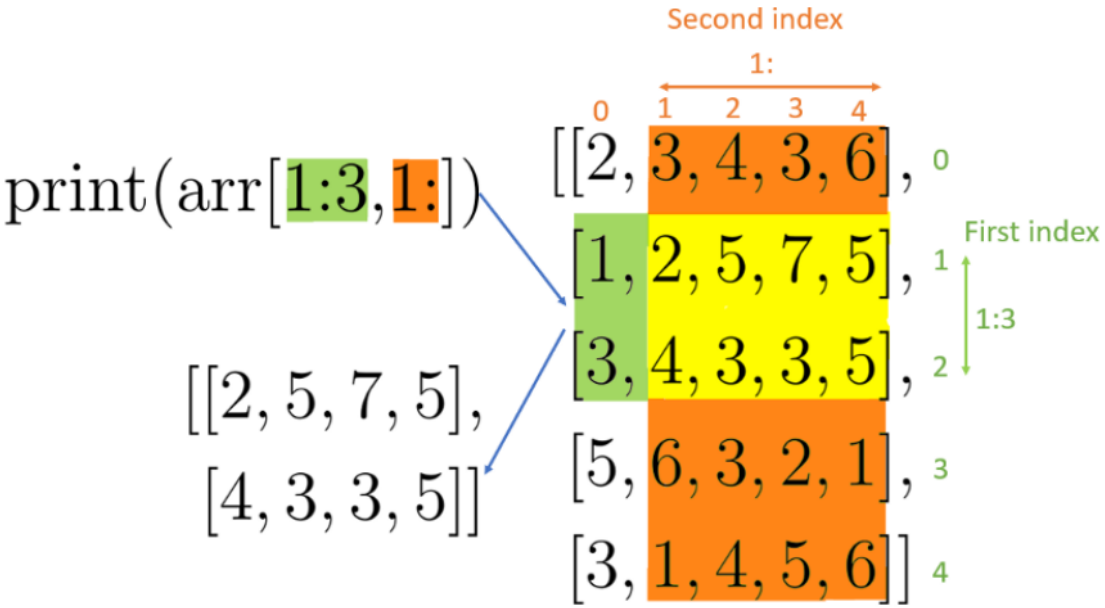
NumPy Array Slicing

- To slice a one dimensional array, we provide a start and an end number separated by a semicolon (:). The range then starts at the start number and **one before the end number**.
- When you want to get the whole array from the start until the element with index 3, I could write: print(arr[0:4]).
- To get from the first index all the way to the end of the array, I can write it without providing a slicing end



NumPy Array Slicing

- For 2D Array



Examples about array slicing

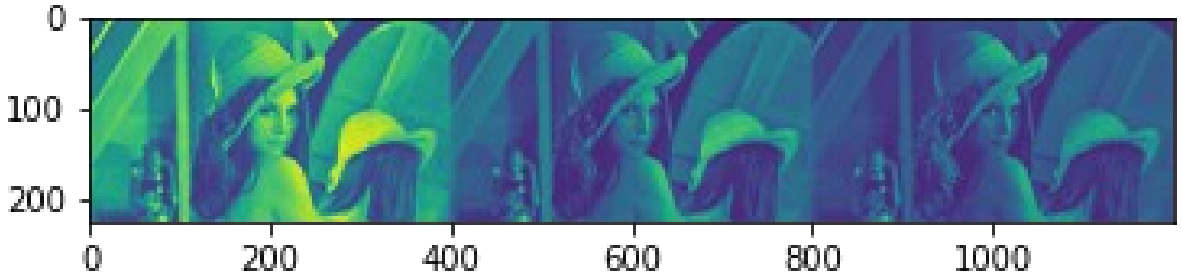
- Image slicing

```
import numpy as np
import matplotlib.pylab as plt

im = plt.imread('lena.jpg')

plt.imshow(im)
print(im.shape) #(225, 400, 3)

im_trim1 = im[128:225, 128:384]
plt.imshow(im_trim1)
print(im_trim1.shape) #(97, 256, 3)
```

We do not specify the third axis which means we take them all

A small exercise for you

Original image



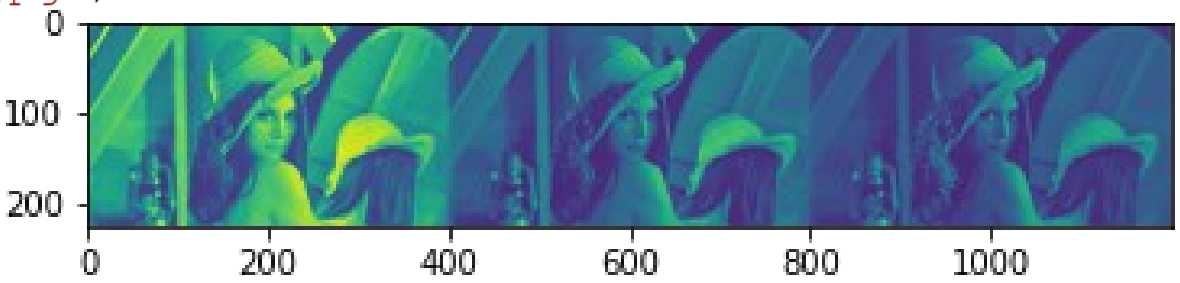
First channel Second channel Third channel

Please write a program to transfer the image from its original status to the separated R G B channel, and then show them in the above forms.

The functions you may need: np.concatenate((arr1, arr2, arr3), axis = decided by you)

A small exercise for you

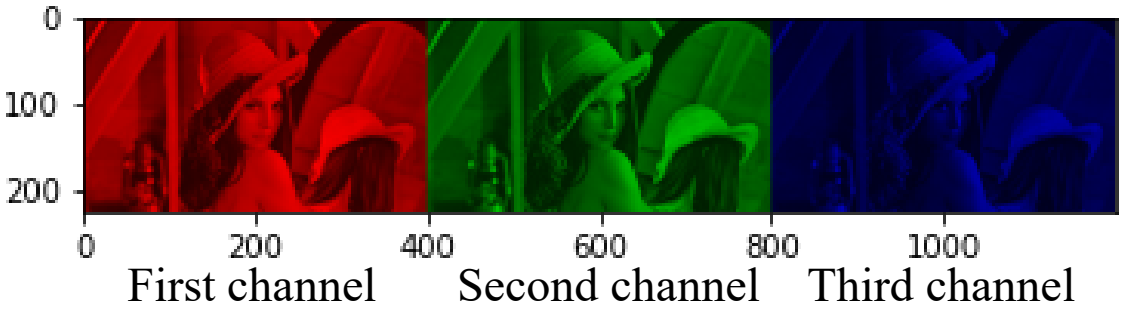
```
import numpy as np
import matplotlib.pylab as plt

im = plt.imread('lena.jpg')

im_R = im[:,:,0]
im_G = im[:,:,1]
im_B = im[:,:,2]

im_RGB = np.concatenate((im_R, im_G, im_B), axis=0)

plt.imshow(im_RGB)
```



Make them red, green, and blue

Make other channels be zero, but still ensure it to be a three-channel matrix

```
import numpy as np
import matplotlib.pylab as plt

im = plt.imread('lena.jpg')

im_R = im.copy()
im_R[:,:,1]=0
im_R[:,:,2]=0
im_G = im.copy()
im_G[:,:,0] = 0
im_G[:,:,2] = 0
im_B = im.copy()
im_B[:,:, 0] = 0
im_B[:,:, 1] = 0

im_RGB = np.concatenate((im_R, im_G, im_B), axis=1)

plt.imshow(im_RGB)
```



Pyplot

- Pyplot is a collection of functions to plot figures
- We only introduce some of its main functions.
- If your feel interested in more, please refer to:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Import it like this!!!

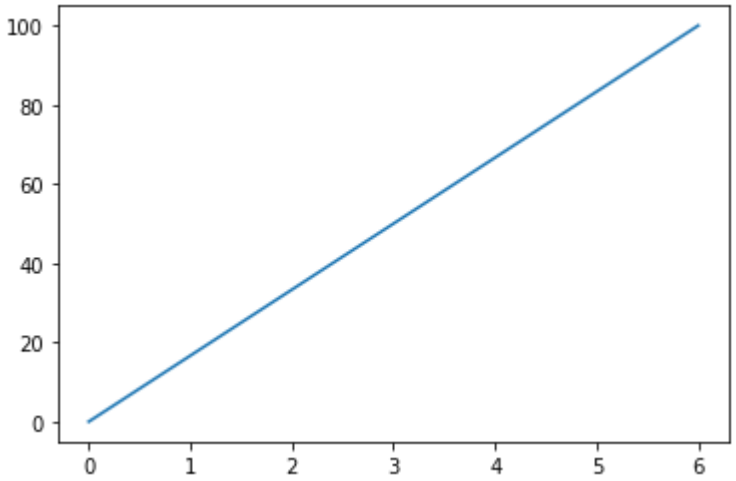
```
import matplotlib.pyplot as plt
```

Pyplot

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 6])
y = np.array([0, 100])

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

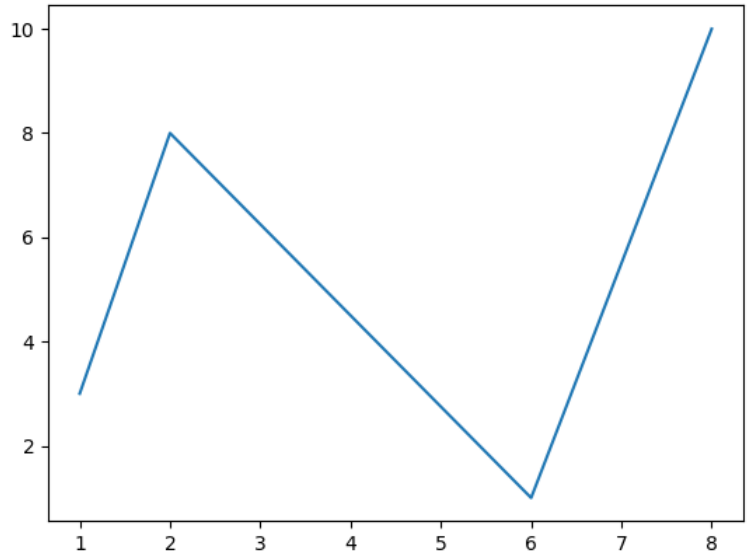
First point

Second point



Exercise 2

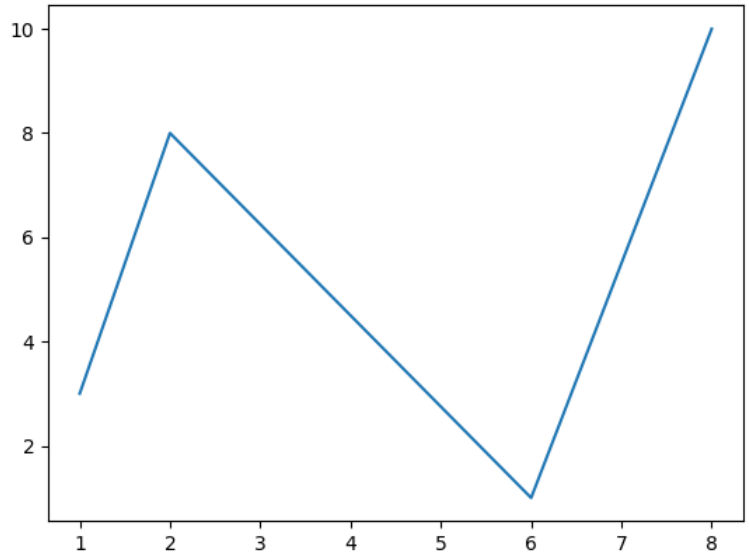
- We have four points (1, 3) , (2, 8) , (6, 1) , (8, 10). Please plot them.



Answer

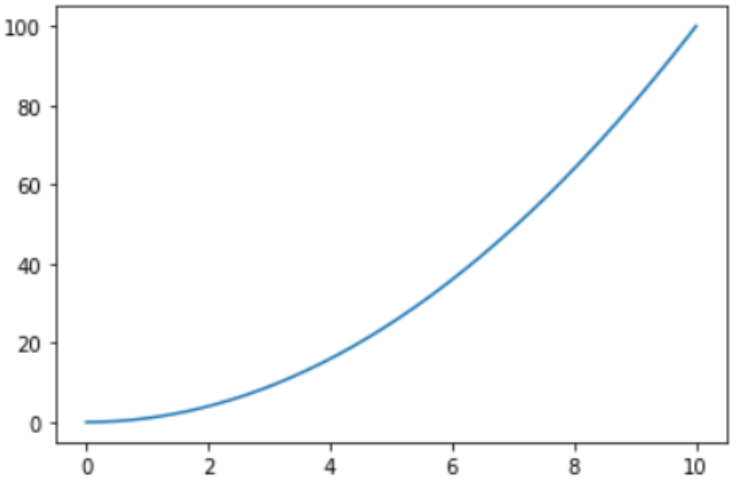
- We have four points (1, 3) , (2, 8) , (6, 1) , (8, 10). Please plot them.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



PyPlot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



To plot the curve of the following mathematical function:

$y = x^2$

```
np.linspace(start, stop, num=50)
```

Return evenly spaced numbers over a specified interval [*start*, *stop*]. Star and stop are both inclusive

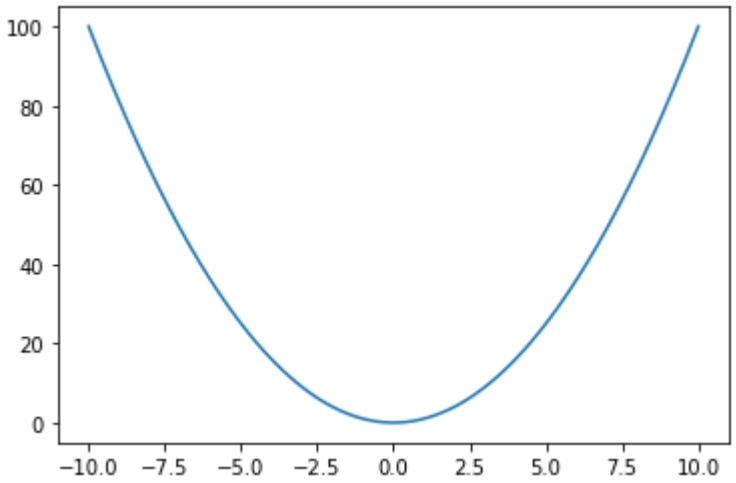
Can we change the x axis to negative value?

PyPlot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (-10, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```

To plot the curve of the following mathematical function:

$y = x^2$



A small exercise for you

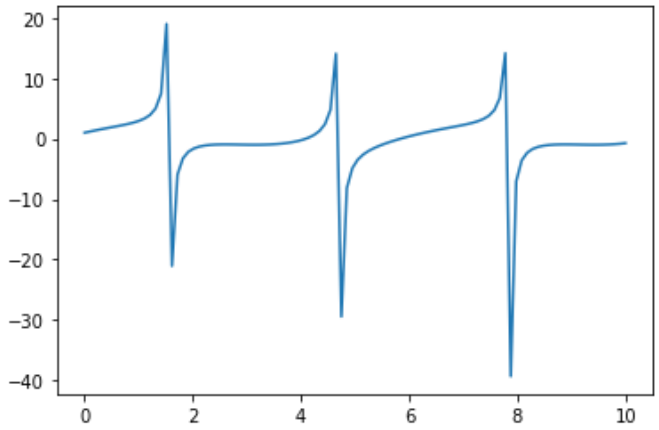
1. Please plot a figure for $y = cos(x) + sin(x) + tan(x)$ between 0 to 10000
2. Please plot a figure for $f = sin^2(x-2)e^{-x^2}$ between -10 to 10

`np.e`

Answer 1

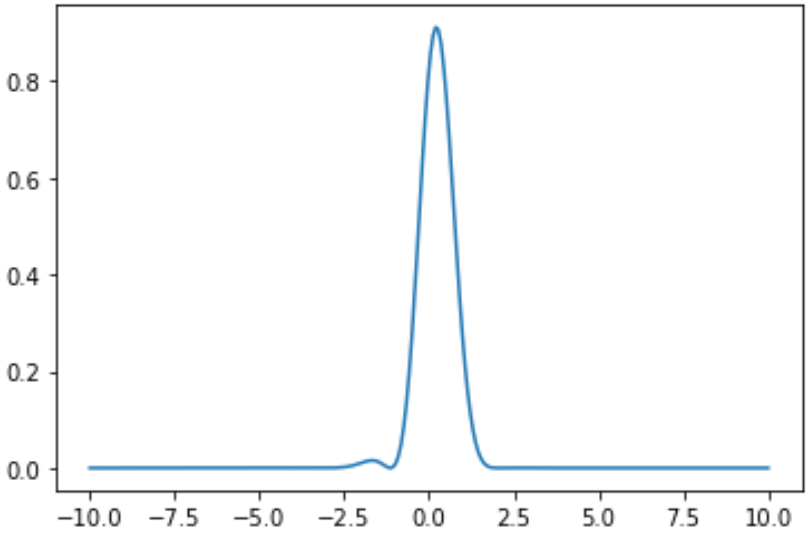
- Please plot a figure for $y = \sin(x) + \cos(x) + \tan(x)$

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (0, 10, 100)
y = np.cos(x) + np.sin(x) + np.tan(x)
plt.plot(x, y)
plt.show()
```



Answer 2

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10,10,256)
f =(np.sin(x-2))**2*(np.e)**(-x**2)

plt.plot(x,f)
plt.show()
```



More about Pyplot

- `linspace` returns evenly spaced numbers over a specified interval.
  - Can plot any figures under Cartesian coordinate system
- It can be used to implement more kinds of coordinate system.

Can you try them?

```
theta = np.linspace(0, 2 * np.pi, 100)
```

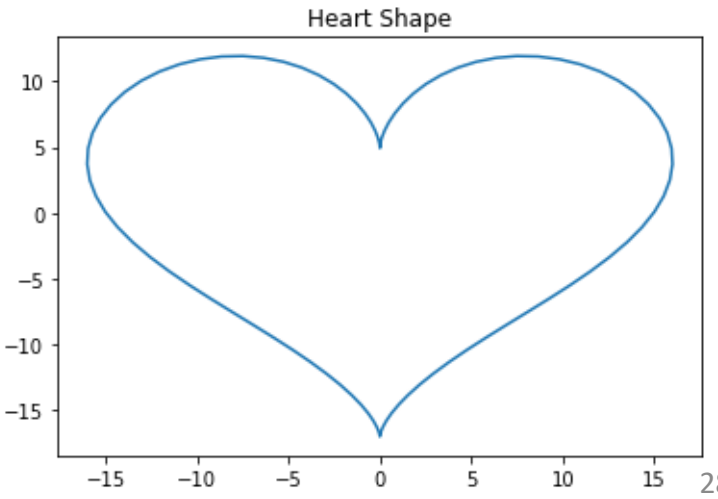
$$x = 16 * sin(\theta)^3$$

$$y = 13 * cos(\theta) - 5 * cos(2\theta) - 2 * cos(3\theta) - cos(4\theta)$$

Answer

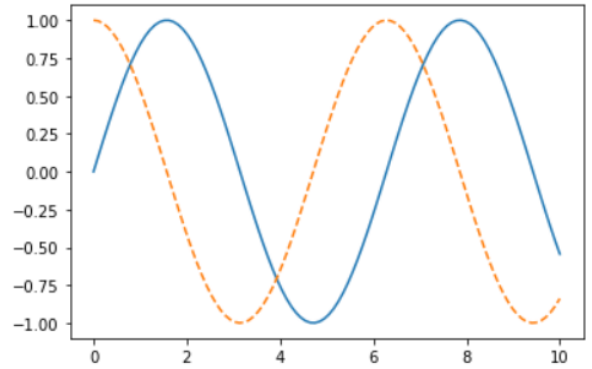
This is a figure under the polar coordinate system

```
import numpy as np
from matplotlib import pyplot as plt

# Creating equally spaced 100 data in range 0 to 2*pi
theta = np.linspace(0, 2 * np.pi, 100)

# Generating x and y data
x = 16 * ( np.sin(theta) ** 3 )
y = 13 * np.cos(theta) - 5* np.cos(2*theta) -
 2 * np.cos(3*theta) - np.cos(4*theta)

# Plotting
plt.plot(x, y)
plt.title('Heart Shape')
plt.show()
```

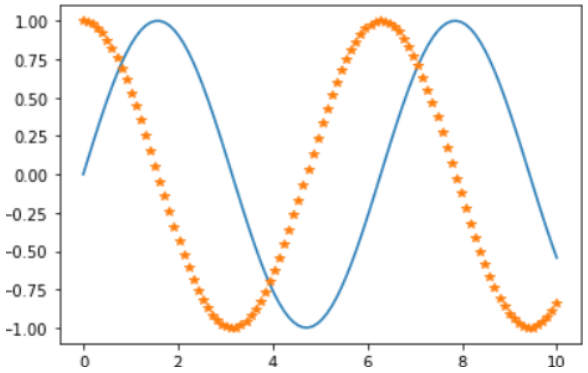


Plot with different styles

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```




```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '*');
```



Change style

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), 'g', marker='o')
plt.plot(x, np.cos(x), 'r', marker='s')
```

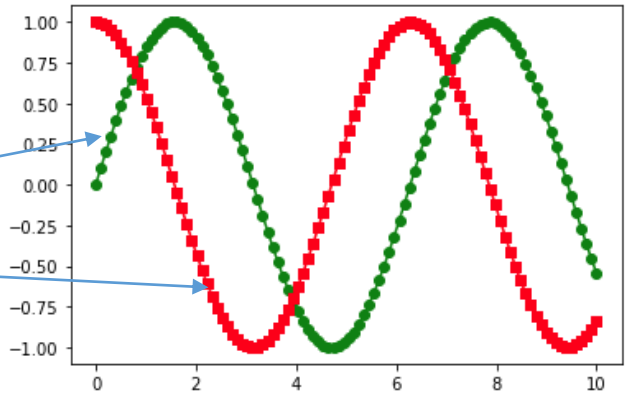
```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), 'm', marker='X')
plt.plot(x, np.cos(x), 'y', marker='$...$')
```

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

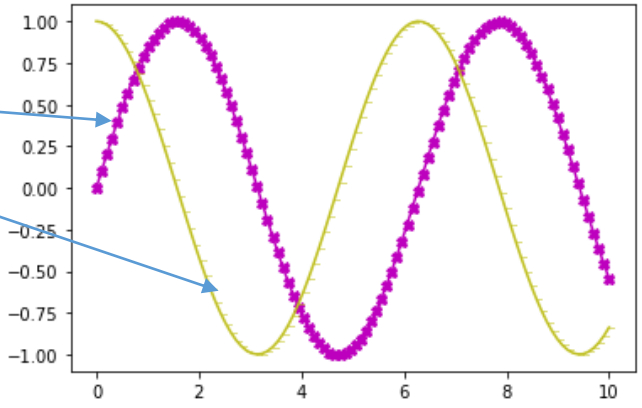
Marker style can be found below:
https://matplotlib.org/stable/api/markers_api.html

Change style

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), 'g', marker='o')
plt.plot(x, np.cos(x), 'r', marker='s')
```




```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), 'm', marker='X')
plt.plot(x, np.cos(x), 'y', marker='$...$')
```



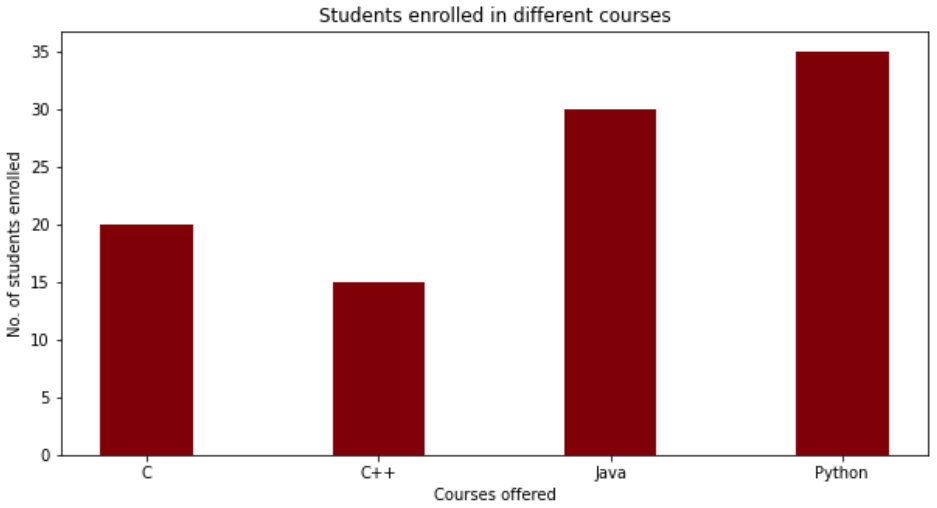
Marker style can be found below:
https://matplotlib.org/stable/api/markers_api.html

Plot Bar

- The bars are positioned at *x* with the given *align*ment. Their dimensions are given by *height* and *width*.

matplotlib.pyplot.bar(*x*, *height*, *width=0.8)*

```
# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

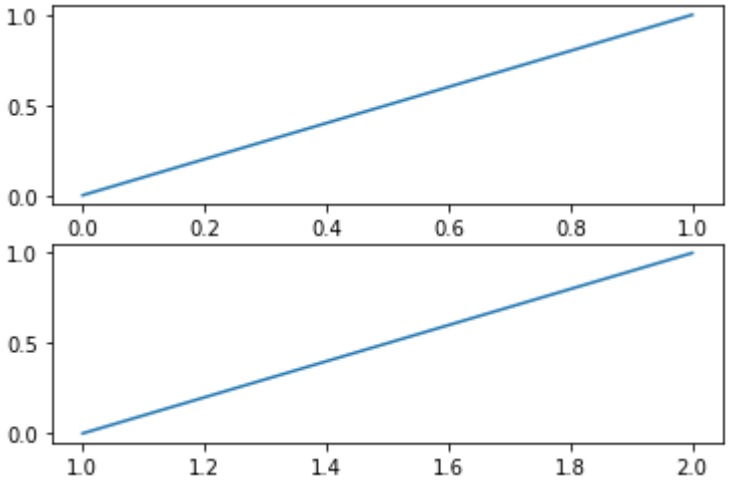


subplots(*nrows=1, ncols=1*)

- subplots creates a figure and a grid of subplots with a single call, while providing reasonable control over how the individual plots are created
- Return two values: fig: the figure to be ploted ax: ***array of Axes***

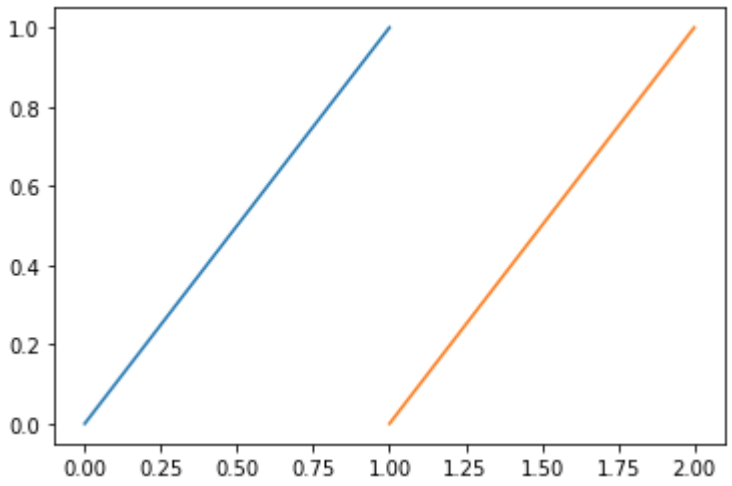
A class

```
import matplotlib.pyplot as plt

fig = plt.figure()

# Plot first figure
ax = fig.subplots(2)
ax[0].plot([0, 1], [0, 1])
ax[1].plot([1, 2], [0, 1])
plt.show()
```



subplots()

```
import matplotlib.pyplot as plt

fig = plt.figure()

# Plot first figure
ax = fig.subplots()

ax.plot([0, 1], [0, 1])
ax.plot([1, 2], [0, 1])
plt.show()
```



Animation

When data becomes available to replace the existing output, the *clear_output* will be called immediately before the new data is added.

```
import matplotlib.pyplot as plt
from matplotlib import animation
import numpy as np
from IPython.display import display, clear_output

fig = plt.figure() #Create a canvas to be painted
x = np.linspace(0, 10, 5)
y = x**2

ax = fig.subplots() #Create the subcanvas
l = ax.plot(x, y) #Define the data in x and y axix
l = l[0]

def animate(i):
  l.set_data(x[:i],y[:i])
  return l

for i in range(len(x)):
    animate(i)
    clear_output(wait=True) #clear_output to clear the output of a cell.
    display(fig)

plt.show()
```

```
x:[0.  2.5  5.  7.5  10. ]
y:[0.  6.25 25. 56.25 100.]
```

1st iteration: i=0 x[:0] y[:0]
2nd iteration: i=1 x[:1] y[:1]
3rd iteration: i=2 x[:2] y[:2]
4th iteration: i=3 x[:3] y[:3]
5th iteration: i=4 x[:4] y[:4]

Function in Lambda Style

- A lambda function can take any number of arguments, but can only have one expression.

```
def g(x):
    return x+1
```

```
g = lambda x:x+1
```

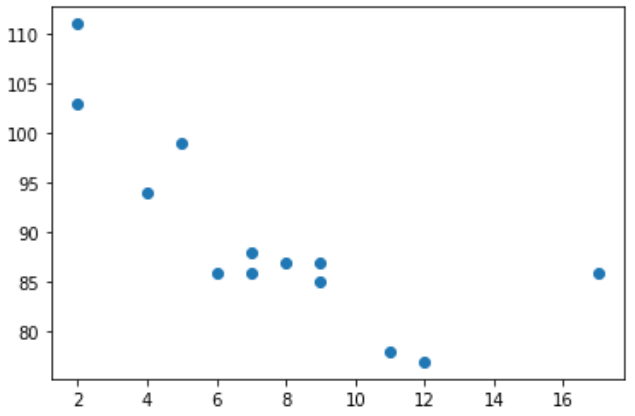
```
def animate(i):
    l.set_data(x[:i],y[:i])
    return l
```

```
animate = lambda i: l.set_data(t[:i], x[:i])
```

PyPlot: Scatter

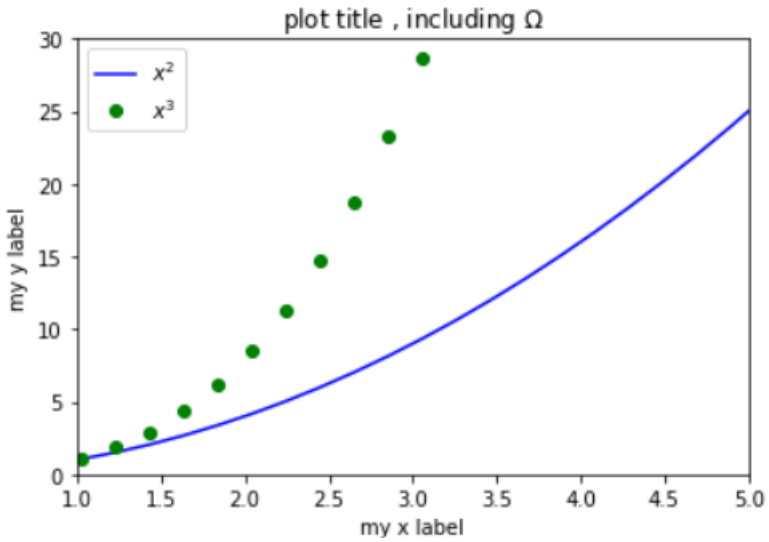
- The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

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



Adding multiple lines and a legend

```
#Adding multiple lines and a legend

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace (0, 10, 50)
y1 = np.power(x, 2)
y2 = np.power(x, 3)
plt.plot(x, y1, 'b-', label='$x^2$')
plt.plot(x, y2, 'go', label='$x^3$')
plt.xlim((1, 5))
plt.ylim((0, 30))
plt.xlabel('my x label')
plt.ylabel('my y label')
plt.title('plot title , including $\Omega$')
plt.legend()
```



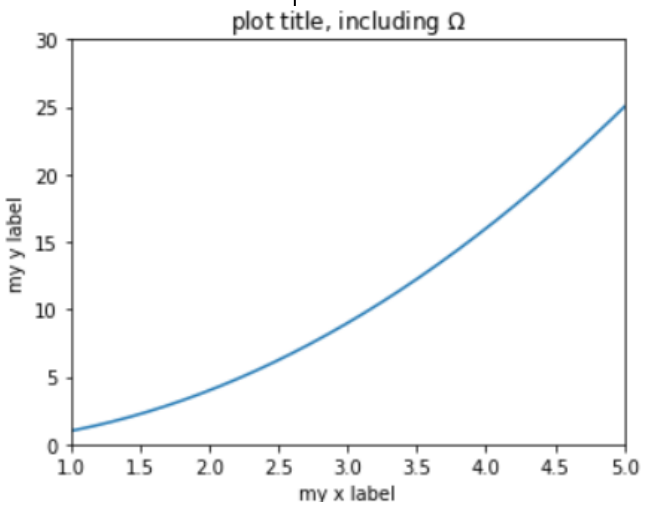
To plot the curve of the following mathematical function:

$y_1 = x^2 \quad y_2 = x^3$

Make your figure more clear

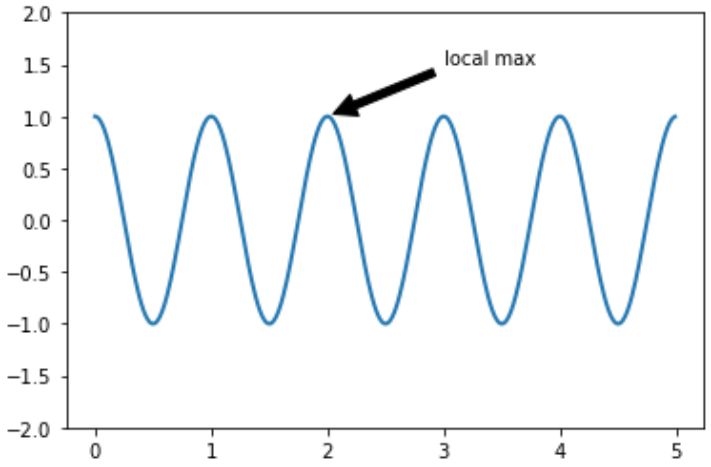
```
#Adding titles and labels
import numpy as np
import matplotlib.pyplot as plt

f, ax = plt.subplots(1, 1, figsize=(5,4))
x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
ax.plot(x, y)
ax.set_xlim((1, 5)) #Set the x-axis view limits.
ax.set_ylim((0, 30)) #Set the y-axis view limits.
ax.set_xlabel('my x label') #set the y label here
ax.set_ylabel('my y label') #set the x label here
ax.set_title('plot title, including $\Omega$') # set the title here
plt.tight_layout()
```

Give your figure some labels!



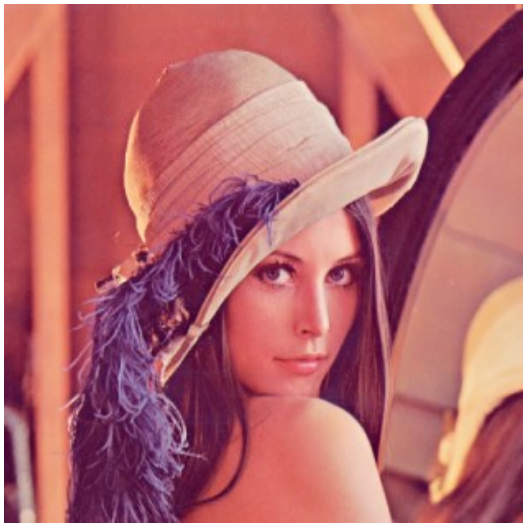
Annotate your figure

```
ax = plt.subplot()

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2) #lw = Linewidth

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```



At last, A story about Lena figure

- Lenna or Lena is a standard test image used in the field of image processing since 1973
- It is a picture of the Swedish model Lena Söderberg captured in 1960s for Playboy magazine.



1960s

Lenna attended a research conference in 1997

She attended ICIP 2015 in Québec