COMP7035

Python for Data Analytics and Artificial Intelligence

Numpy/Matplotlib

Renjie Wan/Jun Qi

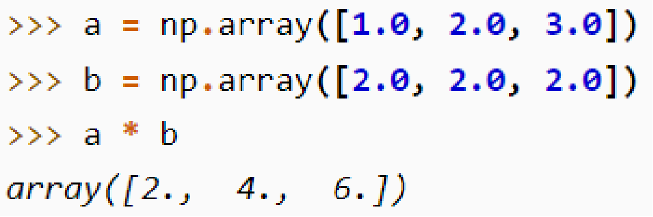
22/10/2024

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

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.

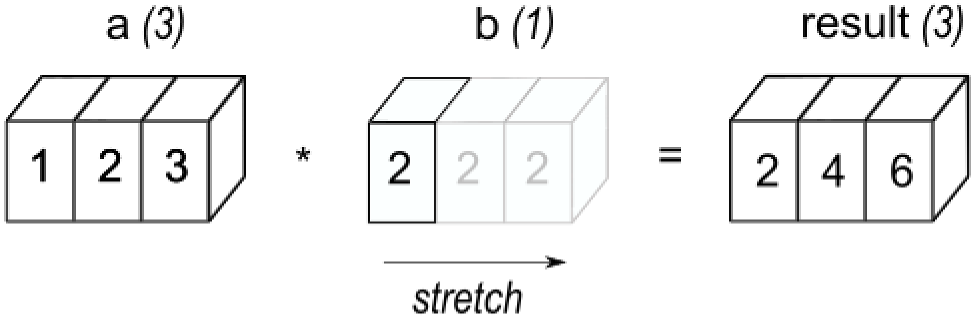


```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([2.,  4.,  6.])
```

With same shapes

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

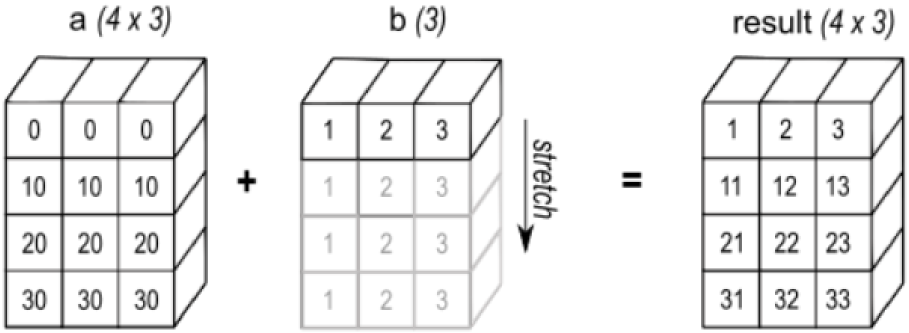
With different shapes



General broadcasting rule

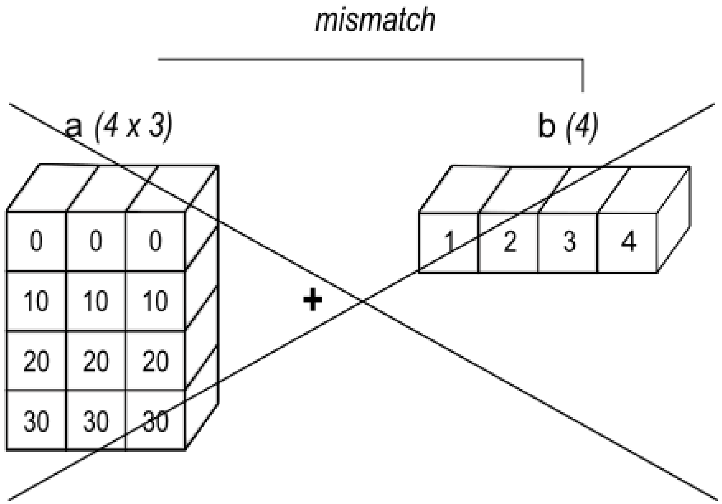
- Broadcasting for two arrays
  - *A one-dimensional array added to a two-dimensional array results in broadcasting if number of 1-d array elements matches the number of 2-d array columns.*

```
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.*

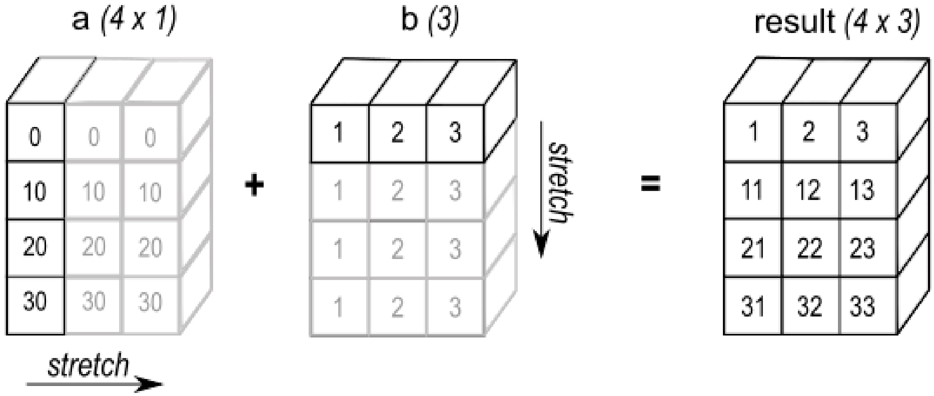


Different column numbers. Not ALLOWED!!!

General broadcasting rule

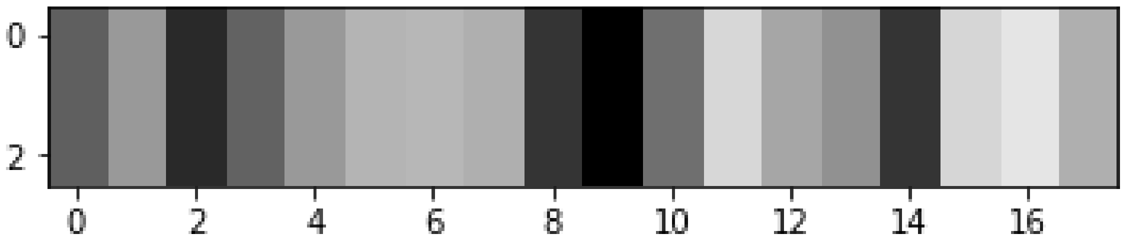
- Broadcasting provides a convenient way of taking the outer product (or any other outer operation) of two arrays. The following example shows an outer addition operation of two 1-d arrays:

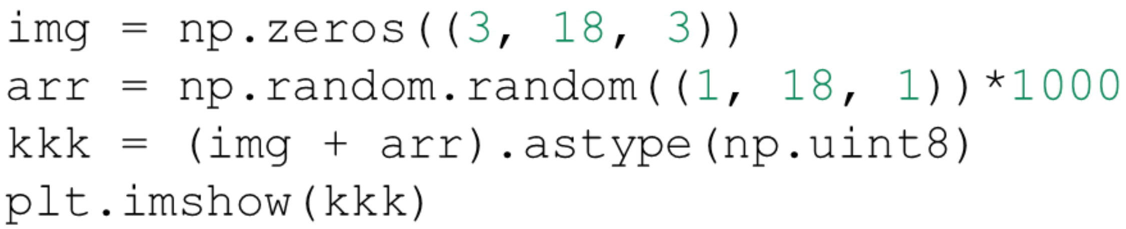
```
a = np.array([0.0, 10.0, 20.0, 30.0])
b = np.array([1.0, 2.0, 3.0])
print(a[:, np.newaxis])
a[:, np.newaxis] + b
```



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

Let us make it more clear

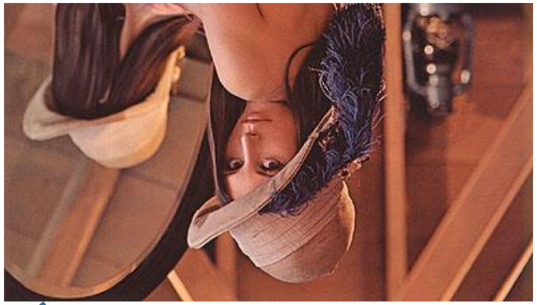




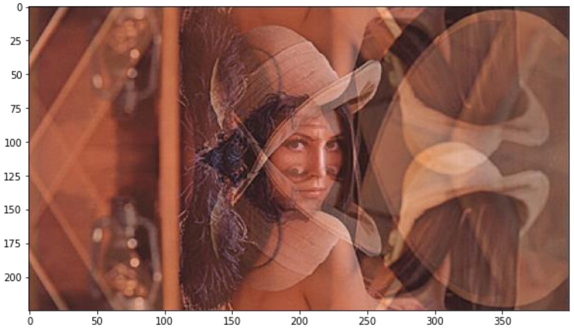
```
img = np.zeros((3, 18, 3))
arr = np.random.random((1, 18, 1))*1000
kkk = (img + arr).astype(np.uint8)
plt.imshow(kkk)
```

Practical examples for broadcasting

- Image blending



After image flip

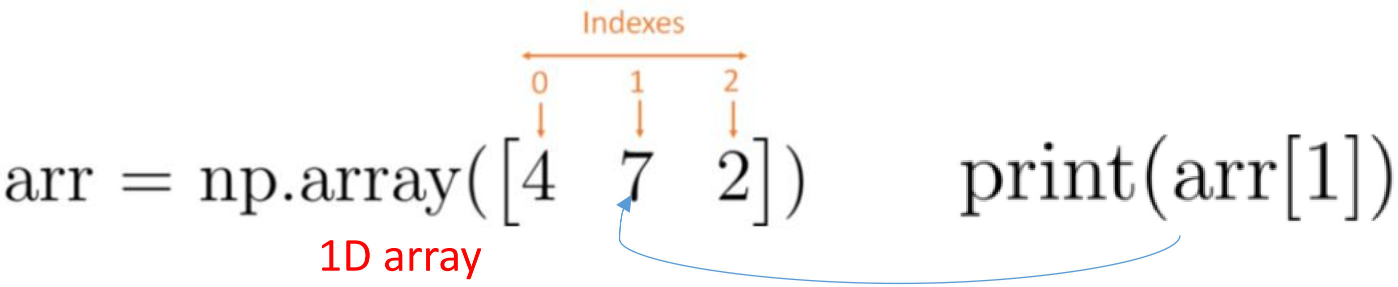


Blended image

```
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)
```

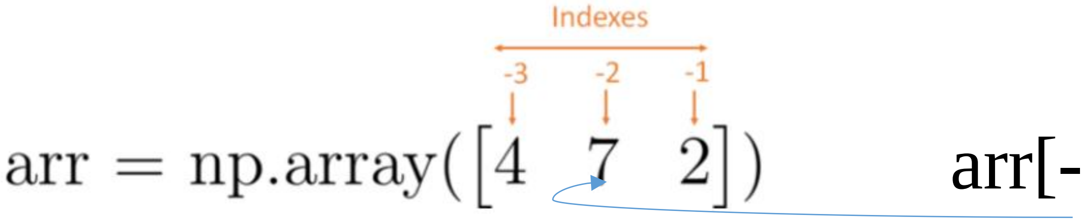
Array Element Access

- From 1D to 2D



```
arr = np.array([4  7  2])        print(arr[1])
```

1D array

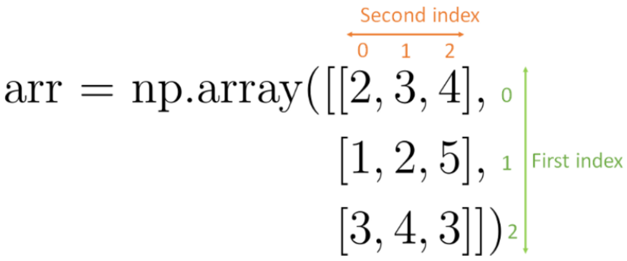


```
arr = np.array([4  7  2])
```

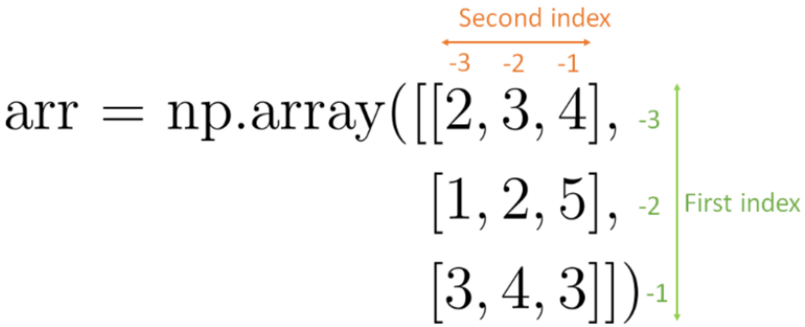
arr[-2]

Array Element Access

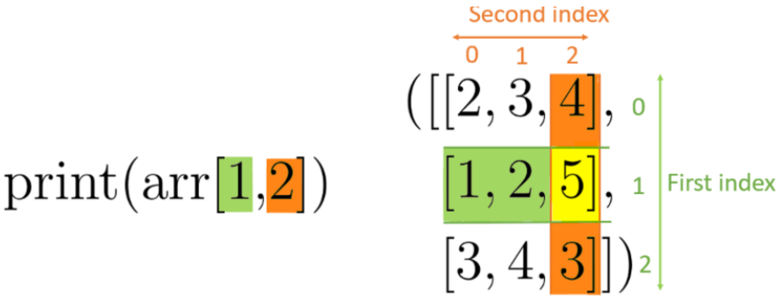
- From 1D to 2D



$$\text{arr} = \text{np.array}([[2, 3, 4], [1, 2, 5], [3, 4, 3]])$$



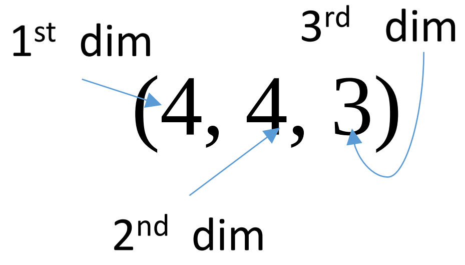
$$\text{arr} = \text{np.array}([[2, 3, 4], [1, 2, 5], [3, 4, 3]])$$

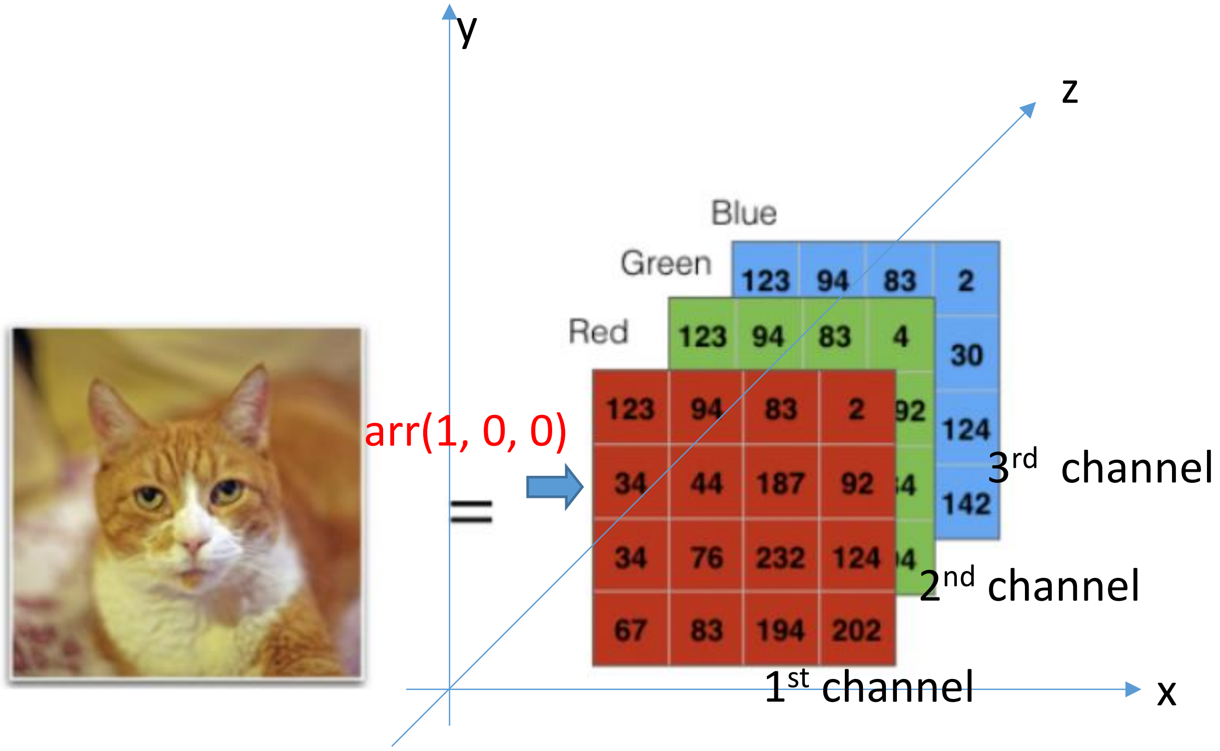


$$\text{print}(\text{arr}[1,2]) \qquad ([[2, 3, 4], [1, 2, 5], [3, 4, 3]])$$

print(arr[-2, -3])

Array Element Access

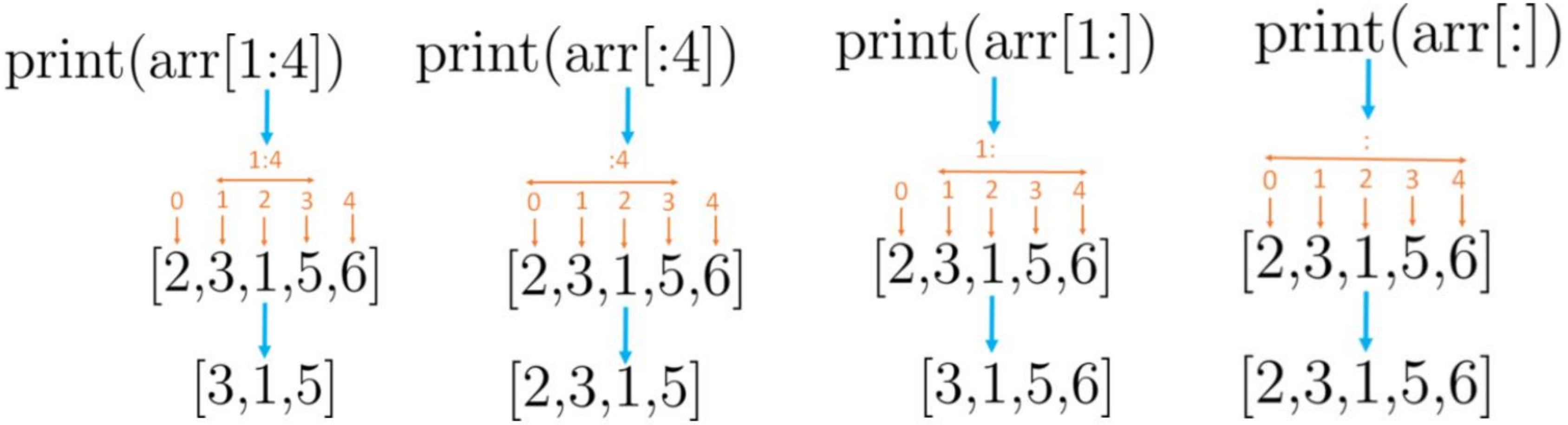




123	94	83	2
34	44	187	92
34	76	232	124
67	83	194	202

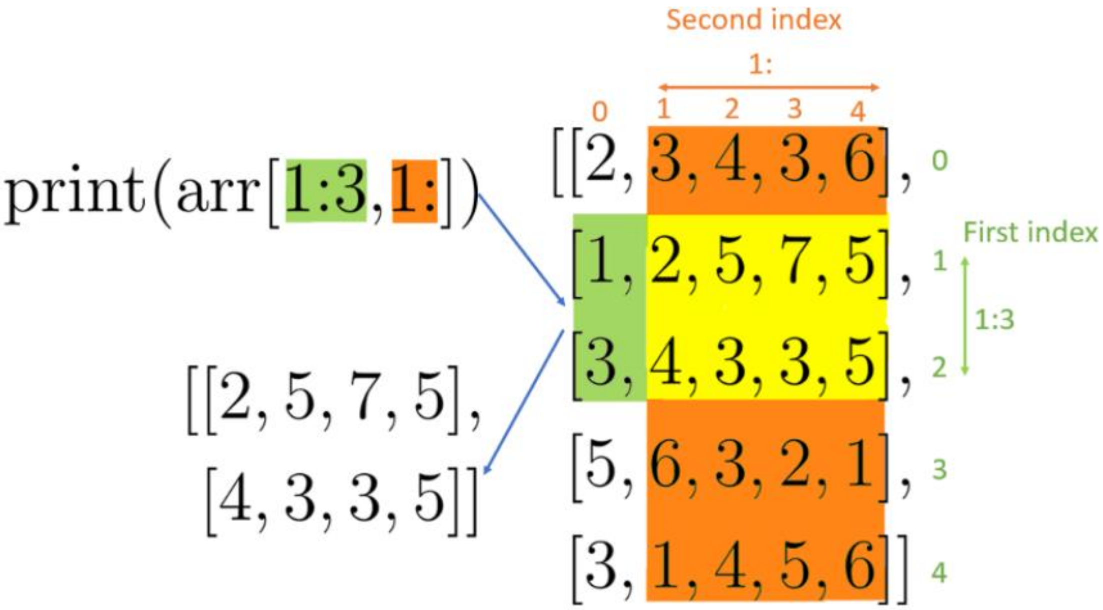
NumPy Array Slicing

- 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



```
print(arr[1:3,1:])
```

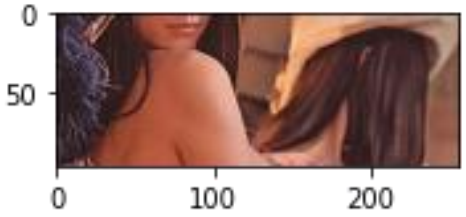
```
[[2,3,4,3,6],
 [1,2,5,7,5],
 [3,4,3,3,5],
 [5,6,3,2,1],
 [3,1,4,5,6]]
```

```
[[2,5,7,5],
 [4,3,3,5]]
```

Examples about array slicing

- Image slicing

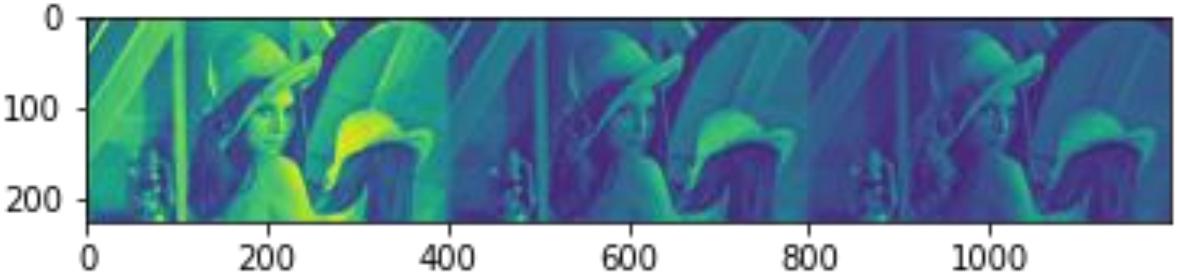
```
import numpy as np
import matplotlib.pylab as plt

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

Exercise-1

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)

Exercise-1

Access and concatenate different channels of Lena

Make them red, green, and blue

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

[redacted]

```
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

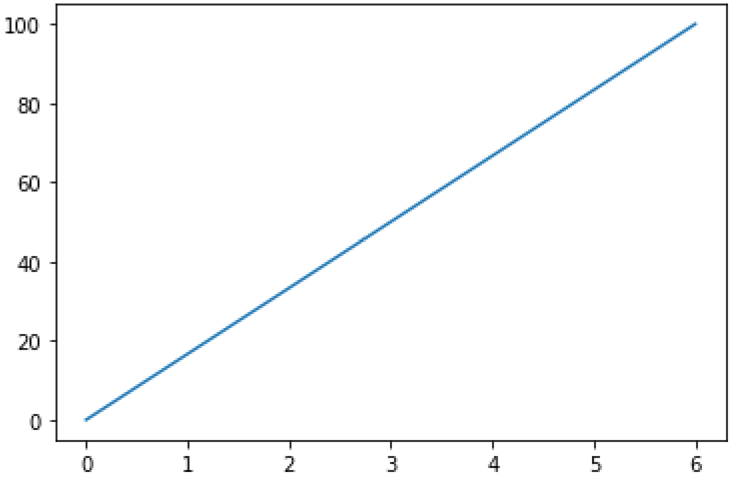
```
import matplotlib.pyplot as plt
import numpy as np
```

First point

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

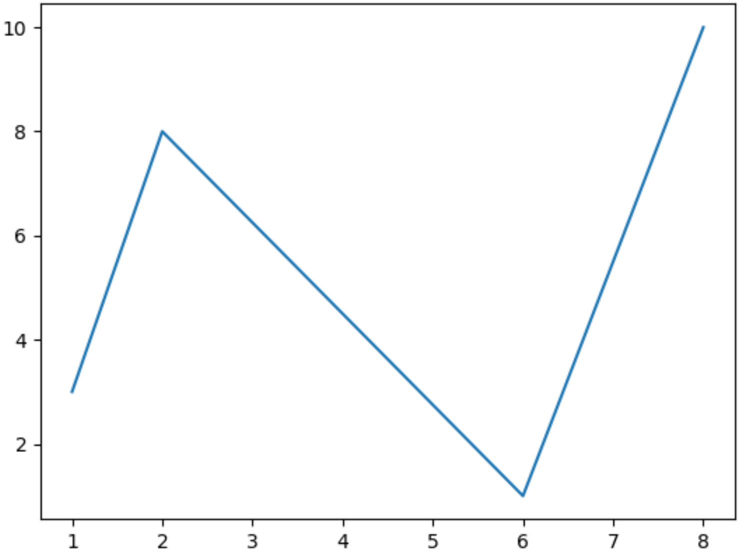
Second point

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



Exercise 2

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

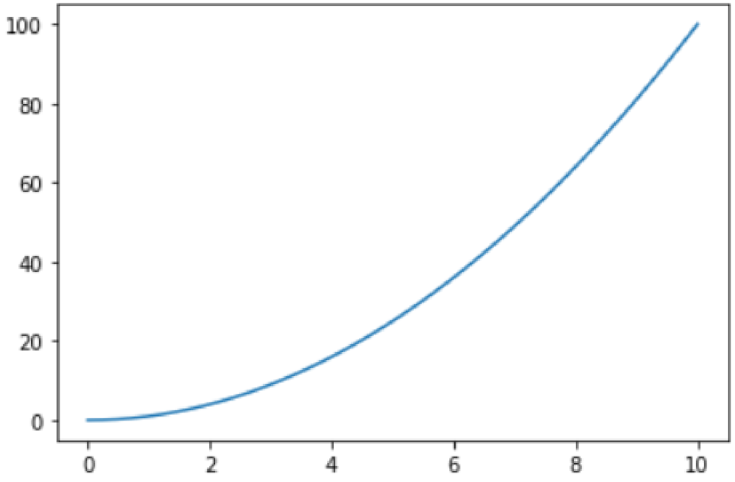


Exercise 2-Answer

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

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:

$$y = x^2$$

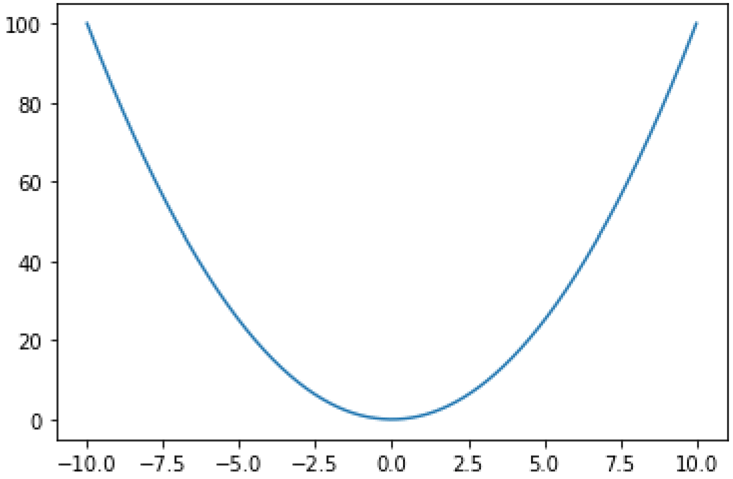
`np.linspace(start, stop, num=50)`

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

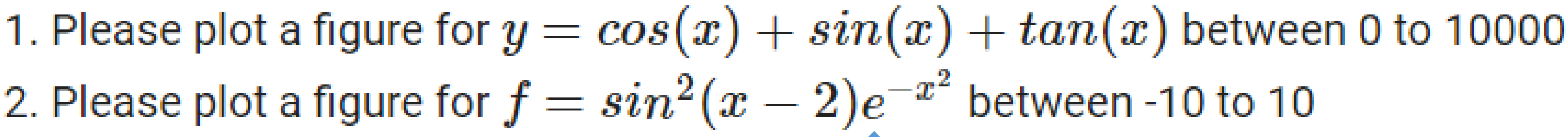
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (-10, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



To plot the curve of the following mathematical function:

$y = x^2$

Exercise-3



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

Exercise-3-Answer

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

[redacted]

Exercise-3-Answer

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

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)
```

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

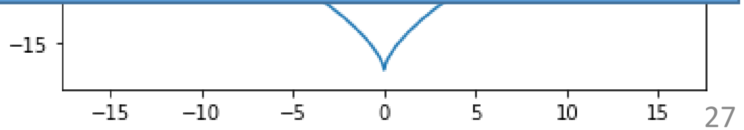
$$y = 13 * cos(\theta) - 5 * cos(2\theta) - 2 * cos(3\theta) - cos(4\theta)$$

Answer

This is a figure under the polar coordinate system

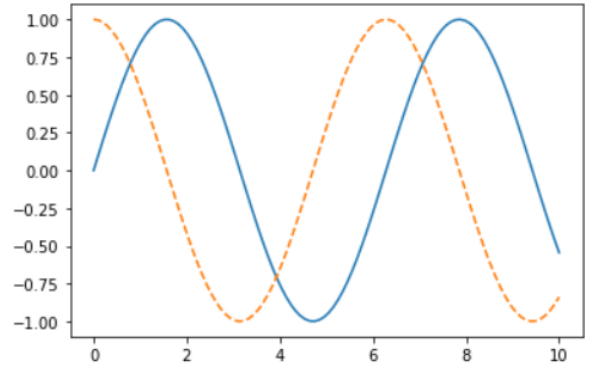
```
import numpy as np
from matplotlib import pyplot as plt
```

[redacted]

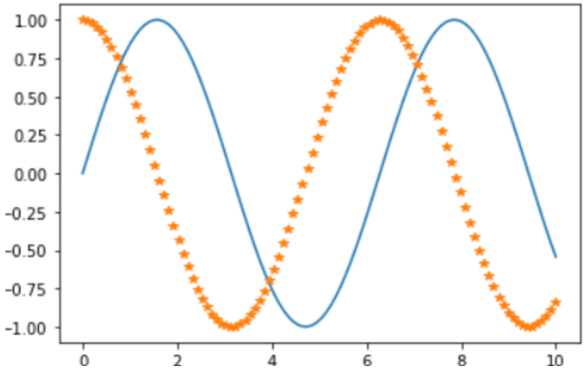


Plot with different styles

```
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), '*');
```

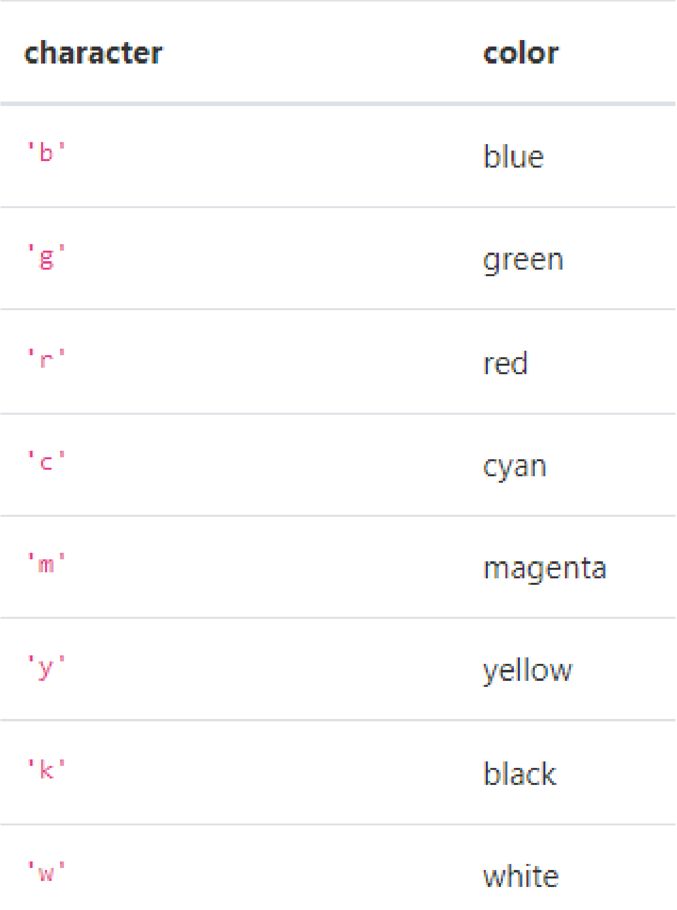


Change style

```
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='$...$')
```

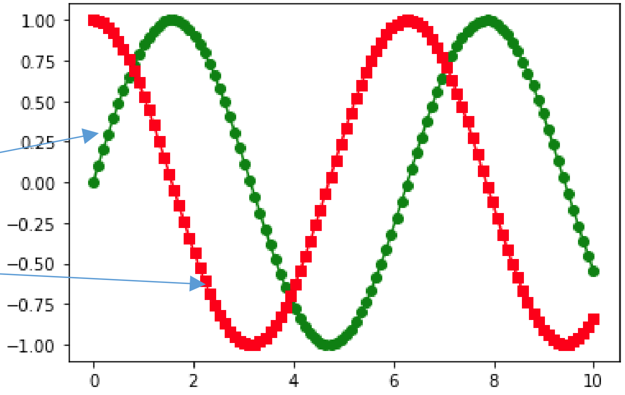
Marker style can be found below:
https://matplotlib.org/stable/api/markers_api.html



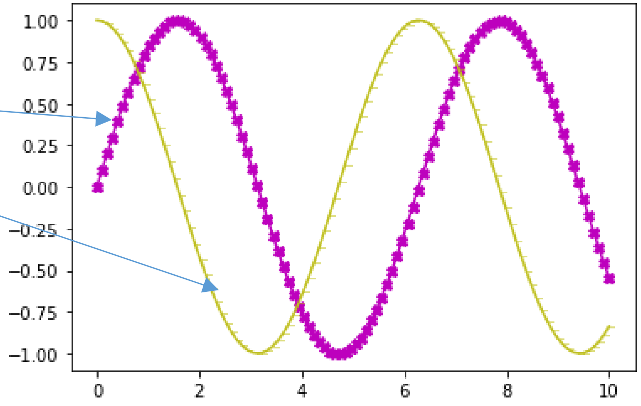
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

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='$...$')
```



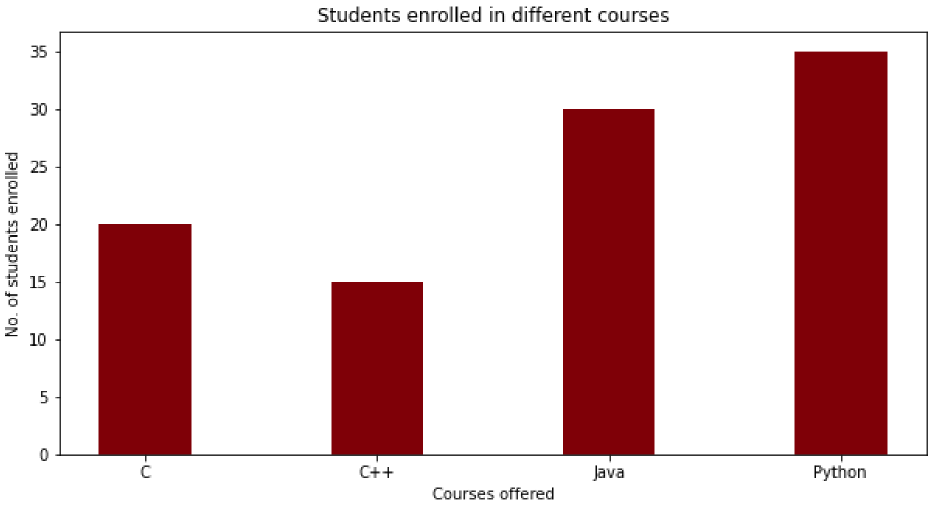
Marker style can be found below:
https://matplotlib.org/stable/api/markers_api.html

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()
```

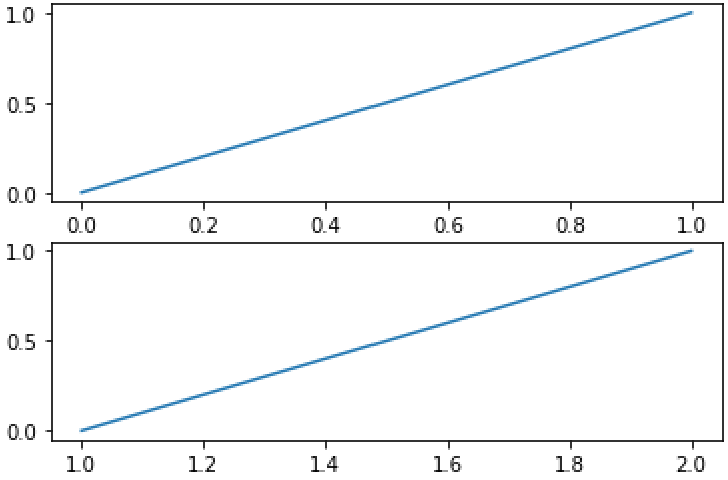


subplots(***nrows=1, ncols=1***)

- 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***

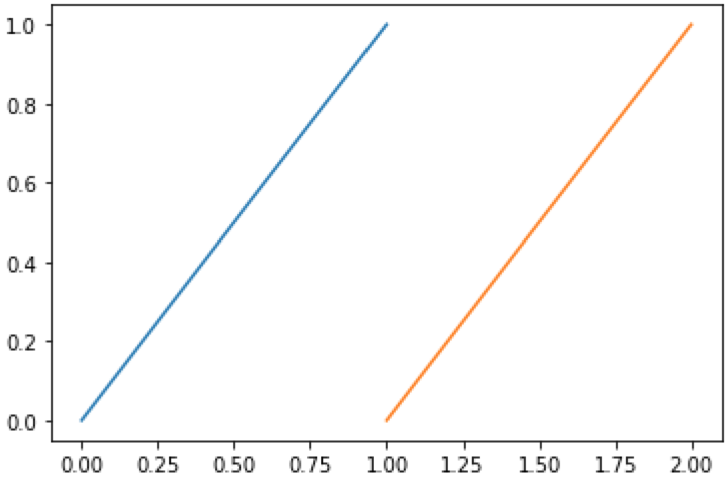
A class

```
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()

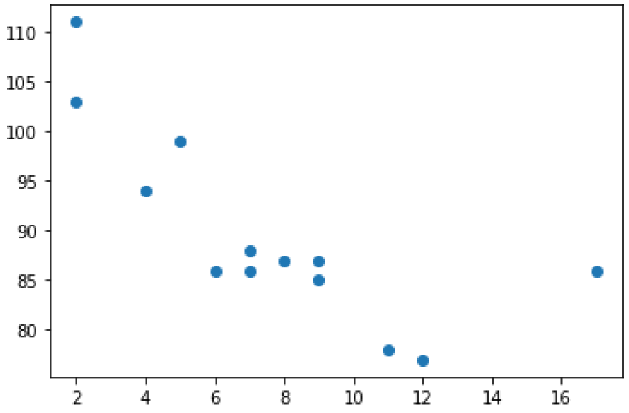
```python
import matplotlib.pyplot as plt

fig = plt.figure()

# Plot first figure
ax = fig.subplots()

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



PyPlot: Scatter

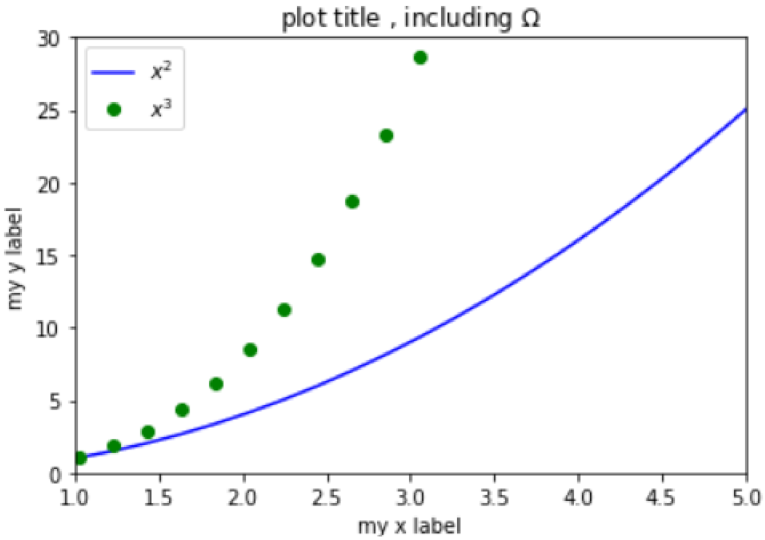
- The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
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()
```



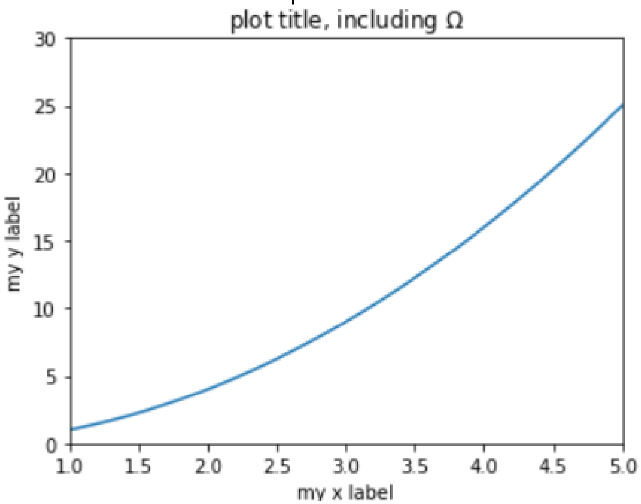
To plot the curve of the following mathematical function:

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

Make your figure more clear

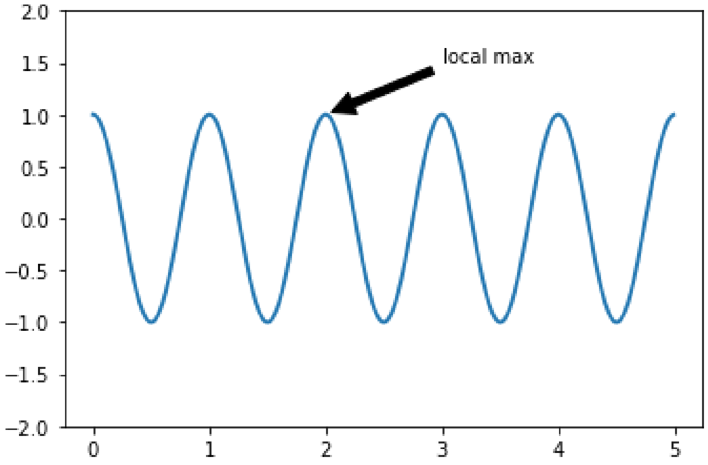
```python
#Adding titles and labels
import numpy as np
import matplotlib.pyplot as plt

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()
```


local max