Lecture 3: Getting started with Python

- Jupyter notebook
- Numpy Arrays and slicing
- Intro to Matplotlib

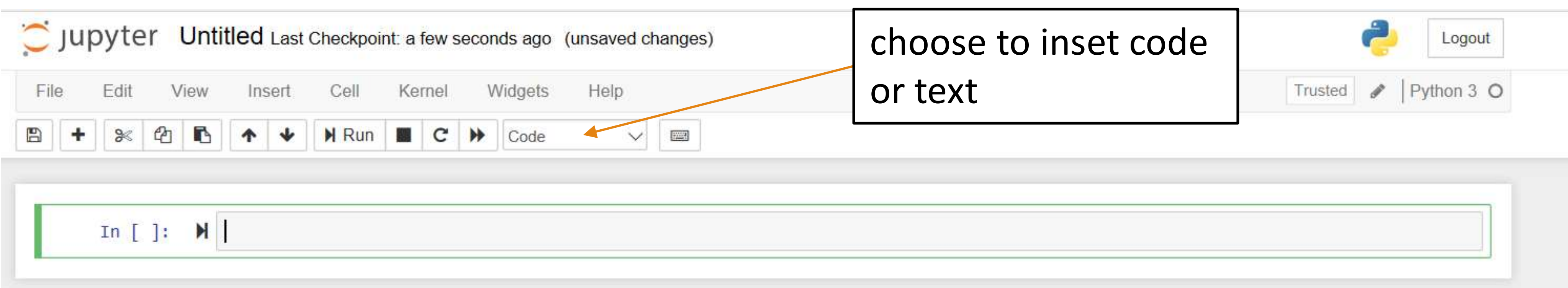
Jupyter notebook

An improvement on interactive shell (ipython) environment

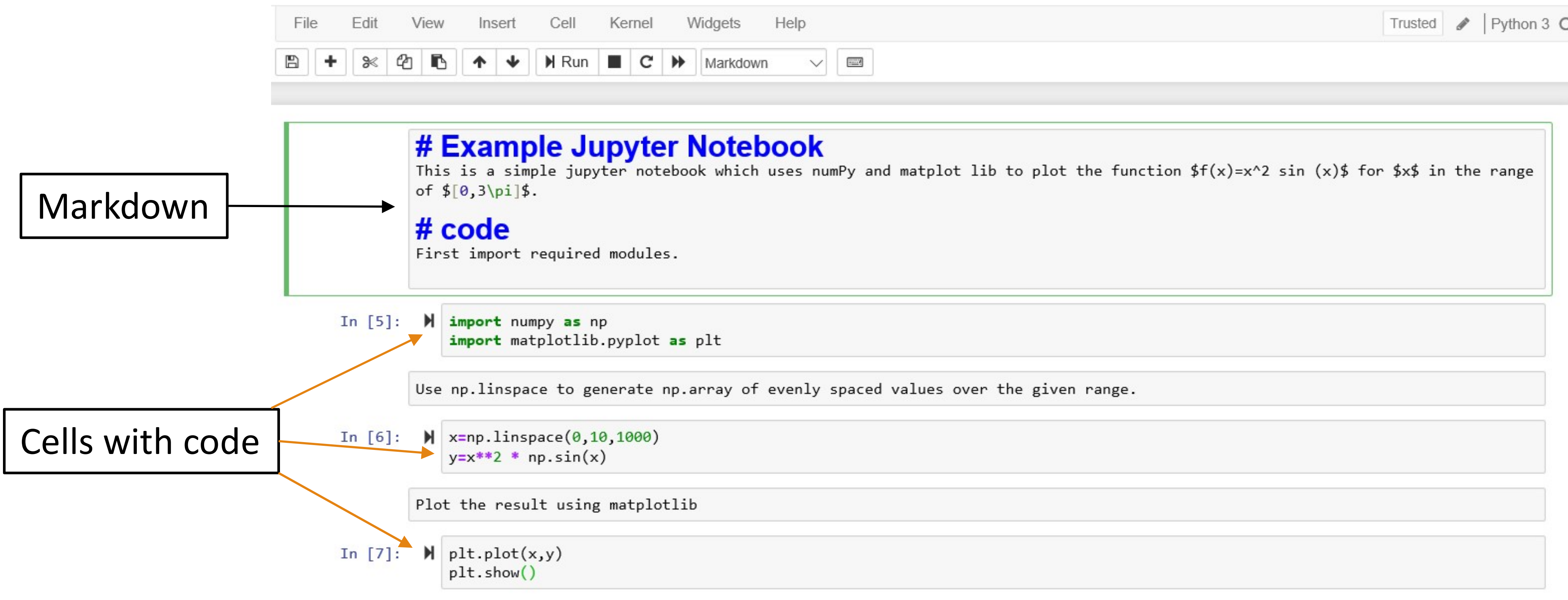
Allow you to create documents which contain live code, equations, and visualizations.

- Markdown allows you to include latex equations, html, etc.

Has support for many different languages (name is from Julia, Python, R)



Example Notebook



After Markdown is executed it is formatted correctly.

Individual cells can be executed with Shift-Enter.

Command mode (Esc) vs. Edit mode

- In command mode, pressing chars issues commands

  a (insert cell above)

  b (inset cell below)

  m (change cell to markdown)

  y (change cell to code)

  … many more in help

Example Jupyter Notebook

This is a simple jupyter notebook which uses numPy and matplotlib to plot the function $f(x) = x^2 sin(x)$ for x in the range of $[0, 3\pi]$.

code

First import required modules.

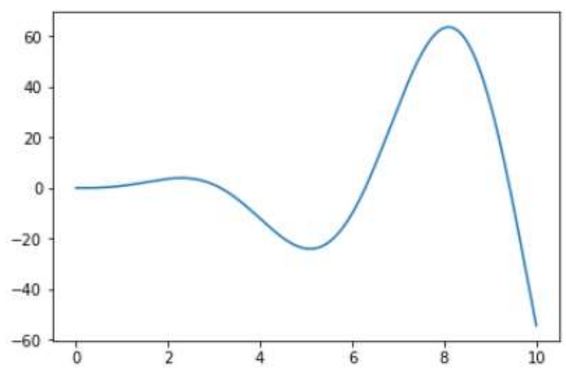
```
import numpy as np
import matplotlib.pyplot as plt
```

Use np.linspace to generate ndarray of evenly spaced values over the given range.

```
x=np.linspace(0,10,1000)
y=x**2 * np.sin(x)
```

Plot the result using matplotlib

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



keyboard shortcuts (in the help)

Command Mode (press `Esc` to enable)

Edit Shortcuts

- `F`: find and replace
- `Ctrl-Shift-F`: open the command palette
- `Ctrl-Shift-P`: open the command palette
- `Enter`: enter edit mode
- `P`: open the command palette
- `Shift-Enter`: run cell, select below
- `Ctrl-Enter`: run selected cells
- `Alt-Enter`: run cell and insert below
- `Y`: change cell to code
- `M`: change cell to markdown
- `R`: change cell to raw
- `1`: change cell to heading 1
- `2`: change cell to heading 2
- `3`: change cell to heading 3
- `4`: change cell to heading 4
- `5`: change cell to heading 5
- `6`: change cell to heading 6
- `K`: select cell above
- `Up`: select cell above
- `Down`: select cell below
- `J`: select cell below
- `Shift-K`: extend selected cells above
- `Shift-Up`: extend selected cells above
- `Shift-Down`: extend selected cells below
- `Shift-J`: extend selected cells below
- `A`: insert cell above
- `B`: insert cell below
- `X`: cut selected cells
- `C`: copy selected cells
- `Shift-V`: paste cells above
- `V`: paste cells below
- `Z`: undo cell deletion
- `D`, `D`: delete selected cells
- `Shift-M`: merge selected cells, or current cell with cell below if only one cell is selected
- `Ctrl-S`: Save and Checkpoint
- `S`: Save and Checkpoint
- `L`: toggle line numbers
- `O`: toggle output of selected cells
- `Shift-O`: toggle output scrolling of selected cells
- `H`: show keyboard shortcuts
- `I`, `I`: interrupt the kernel
- `0`, `0`: restart the kernel (with dialog)
- `Esc`: close the pager
- `Q`: close the pager
- `Shift-L`: toggles line numbers in all cells, and persist the setting
- `Shift-Space`: scroll notebook up
- `Space`: scroll notebook down

Edit Mode (press `Enter` to enable)

- `Tab`: code completion or indent
- `Shift-Tab`: tooltip
- `Ctrl-]`: indent
- `Ctrl-[`: dedent
- `Ctrl-A`: select all
- `Ctrl-Z`: undo
- `Ctrl-/`: comment
- `Ctrl-D`: delete whole line
- `Ctrl-U`: undo selection
- `Insert`: toggle overwrite flag
- `Ctrl-Home`: go to cell start
- `Ctrl-Up`: go to cell start
- `Ctrl-End`: go to cell end
- `Ctrl-Down`: go to cell end
- `Ctrl-Left`: go one word left
- `Ctrl-Right`: go one word right
- `Ctrl-Backspace`: delete word before
- `Ctrl-Delete`: delete word after
- `Ctrl-Y`: redo
- `Alt-U`: redo selection
- `Ctrl-M`: enter command mode
- `Ctrl-Shift-F`: open the command palette
- `Ctrl-Shift-P`: open the command palette
- `Esc`: enter command mode
- `Shift-Enter`: run cell, select below
- `Ctrl-Enter`: run selected cells
- `Alt-Enter`: run cell and insert below
- `Ctrl-Shift-Minus`: split cell at cursor
- `Ctrl-S`: Save and Checkpoint
- `Down`: move cursor down
- `Up`: move cursor up

Including Latex in the markdown

```
Examples of $\LaTeX$  markup and how to generate some  simple equations.

The double dollar signs produce $$\sum_{x=1}^5 y^z$$ a centered equation with its own line. If you use single dollar sign, the equation $\int_a^b f(x)$ becomes an inline equation.

more examples:

$$e=mc^2$$

Another $\pi=\frac{c}{d}$ inline equation.

$$\frac{d}{dx}e^x=e^x$$

$$\frac{d}{dx}\int_{0}^\infty f(s)ds=f(x)$$

$$f(x)=\sum_i=0^\infty \frac{f^{(i)}(0)}{i!}x^i$$

$$x=\sqrt{\frac{x_i}{z}y}$$
```

Examples of $\LaTeX$ markup and how to generate some simple equations.

The double dollar signs produce

$$\sum_{x=1}^5 y^z$$

a centered equation with its own line. If you use single dollar sign, the equation $\int_a^b f(x)$ becomes an inline equation.

more examples:

$$e=mc^2$$

Another $\pi=\frac{c}{d}$ inline equation.

$$\frac{d}{dx}e^x=e^x$$

$$\frac{d}{dx}\int_{0}^\infty f(s)ds=f(x)$$

$$f(x)=\sum_i=0^\infty \frac{f^{(i)}(0)}{i!}x^i$$

$$x=\sqrt{\frac{x_i}{z}y}$$

Commonly used tex codes

^ superscript
_ subscript
{} used to group
special chars or symbols preceded with \

greek letters:
\alpha, \beta, \gamma, etc.
\frac{}{} for fraction
\sqrt{} for square root
\sum{} for summation
\int{} for integral

Warm up

Write a quick program in jupyter notebook to perform the following summation with a for loop,

$$\sum_{i=0}^{10^7} \frac{i^2}{(1+i^3)}$$

Note: to get the timings you can import time

begintime=time.time()

…your code

finaltime=time.time()

print(“Total time “,finaltime-initialtime)

Important libraries

NumPy:

- Creation and manipulation of numeric arrays/matrices. Lots of useful functions: mathematical functions exp/sin/etc, operations on array (sums, products), global properties (max/min, statistics), linear algebra (overlaps with SciPy but is lighter), … lots more

Matplotlib:

- 2D/3D plotting, interactive plots,

SciPy:

- special functions, optimization, interpolation, Fourier transform, linear algebra (matrix inversion, eigenvalues, etc), integration (ODEs)

can be installed into existing python installation with pip. All standard for computationally focused python distributions like anaconda.

NumPy arrays

A Python list is not the same as an array.

Numpy arrays consist of *fixed length* data which is all the *same type*, allowing for efficient calculations on the entire array.

Creating NumPy arrays: np.array, np.zeros, np.ones, np.arange, np.linspace, np.logspace

```
[6]  a=np.array([1.1,2.4,4.5])
     a
```

```
array([1.1, 2.4, 4.5])
```

```
a=np.zeros(5)
a
```

```
array([0., 0., 0., 0., 0.])
```

```
import time
bt=time.time()
x=[]
y=[]
for i in range(1000000):
  x.append(i/100)
  y.append(np.sin(x[i]))
et=time.time()
et1=et-bt
print(et1)
```

```
1.2792015075683594
```

```
import numpy as np
import time
bt=time.time()
x=np.linspace(0,1000,1000000)
y=np.sin(x)
et2 =time.time()-bt
print(et2)
```

```
0.03769087791442871
```

Making arrays of evenly spaced numbers

arrange

Defaults to integer datatype if no increment (or int increment) is used

beginning of range

end of range

np.arange(0,1,.2) → np.array([0,.2,.4,.6,.8])

increment by

(note: array does not include end value)

linspace

np.linspace(0,1,11) → np.array([0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1.0])

first value

final value

# of points in array

Basic Numpy functionality

Trigonometric functions

`sin` (x, /[, out, where, casting, order, ...])	Trigonometric sine, element-wise.
`cos` (x, /[, out, where, casting, order, ...])	Cosine element-wise.
`tan` (x, /[, out, where, casting, order, ...])	Compute tangent element-wise.
`arcsin` (x, /[, out, where, casting, order, ...])	Inverse sine, element-wise.
`arccos` (x, /[, out, where, casting, order, ...])	Trigonometric inverse cosine, element-wise.
`arctan` (x, /[, out, where, casting, order, ...])	Trigonometric inverse tangent, element-wise.
`hypot` (x1, x2, /[, out, where, casting, ...])	Given the “legs” of a right triangle, return its hypotenuse.
`arctan2` (x1, x2, /[, out, where, casting, ...])	Element-wise arc tangent of `x1/x2` choosing the quadrant correctly.
`degrees` (x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.
`radians` (x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`unwrap` (p[, discont, axis])	Unwrap by changing deltas between values to 2*pi complement.
`deg2rad` (x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`rad2deg` (x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.

Hyperbolic functions

`sinh` (x, /[, out, where, casting, order, ...])	Hyperbolic sine, element-wise.
`cosh` (x, /[, out, where, casting, order, ...])	Hyperbolic cosine, element-wise.
`tanh` (x, /[, out, where, casting, order, ...])	Compute hyperbolic tangent element-wise.
`arcsinh` (x, /[, out, where, casting, order, ...])	Inverse hyperbolic sine element-wise.
`arccosh` (x, /[, out, where, casting, order, ...])	Inverse hyperbolic cosine, element-wise.
`arctanh` (x, /[, out, where, casting, order, ...])	Inverse hyperbolic tangent element-wise.

Rounding

`around` (a[, decimals, out])	Evenly round to the given number of decimals.
`round_` (a[, decimals, out])	Round an array to the given number of decimals.
`rint` (x, /[, out, where, casting, order, ...])	Round elements of the array to the nearest integer.
`fix` (x[, out])	Round to nearest integer towards zero.
`floor` (x, /[, out, where, casting, order, ...])	Return the floor of the input, element-wise.
`ceil` (x, /[, out, where, casting, order, ...])	Return the ceiling of the input, element-wise.
`trunc` (x, /[, out, where, casting, order, ...])	Return the truncated value of the input, element-wise.

Sums, products, differences

`prod` (a[, axis, dtype, out, keepdims])	Return the product of array elements over a given axis.
`sum` (a[, axis, dtype, out, keepdims])	Sum of array elements over a given axis.
`nanprod` (a[, axis, dtype, out, keepdims])	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
`nansum` (a[, axis, dtype, out, keepdims])	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
`cumprod` (a[, axis, dtype, out])	Return the cumulative product of elements along a given axis.
`cumsum` (a[, axis, dtype, out])	Return the cumulative sum of the elements along a given axis.
`nancumprod` (a[, axis, dtype, out])	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
`nancumsum` (a[, axis, dtype, out])	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
`diff` (a[, n, axis])	Calculate the n-th discrete difference along given axis.
`ediff1d` (ary[, to_end, to_begin])	The differences between consecutive elements of an array.
`gradient` (f, *varargs, **kwargs)	Return the gradient of an N-dimensional array.
`cross` (a, b[, axisa, axisb, axisc, axis])	Return the cross product of two (arrays of) vectors.
`trapz` (y[, x, dx, axis])	Integrate along the given axis using the composite trapezoidal rule.

lots more, see https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html

vectorization

simple user defined functions are “vectorized” by default.

```
import numpy as np
def myfunc(x):
  return x**3+2.3*np.sin(x)/np.sqrt(x)

x=np.linspace(1,2,10)
myfunc(x)
```

```
array([2.93538327, 3.32720803, 3.78110152, 4.30633307, 4.91217737,
       5.60794875, 6.40302091, 7.30683678, 8.32891173, 9.47883187])
```

can pass array to function and it will efficiently calculate the function for each value in the array.

Functions which contain loops or boolean logic are not vectorized by default, but can be through np.vectorize.

```
@np.vectorize
def myabs(x):
  if(x<0):
    return -x
  else:
    return x
myabs(a)
```

or alternately,

```
vfunc=np.vectorize(myfunc)
vfunc(a)
```

NumPy arrays objects

Data is not copied in assignment unless a ndarray is returned from a method.

```
a=np.array([1.1,2.2,3.3])
b=a # a and b point toward same object
a[0]=9.9 # attribute of object changes, both a and b point to it
print(a)
print(b)
```

```
[9.9 2.2 3.3]
[9.9 2.2 3.3]
```

```
a=np.array([1.1,2.2,3.3])
b=np.array(a)  # new object made for b, same values as a
a[0]=9.9
print(a)
print(b)
```

```
[9.9 2.2 3.3]
[1.1 2.2 3.3]
```

useful methods:
sort
size
std
sum
mean
argmax
argmin
resize
cumsum
reshape

attributes:
shape
size
T (transpose)
real
imag

slicing

```
a=np.arange(0,64)
```

```
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63])
```

initial index

count-by index

a[: :]

final Index (not inclusive)

code	returns
a[0:10:2]	array([0, 2, 4, 6, 8])
a[7:23:3]	array([7, 10, 13, 16, 19, 22])
a[3:13]	array([3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
a[59:]	array([59, 60, 61, 62, 63])
a[:6]	array([0, 1, 2, 3, 4, 5])

If only **one colon** , it is assumed **count-by index equals 1**

If **no index after colon**, it slices to final element in the array

If **no index before colon**, it is assumed to be 0, the first index in the array.

note, returned array has elements up to final index-1. a[3:4] = a[3]

Higher dimensional arrays

```
a=np.arange(0,64)
```

```
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63])
```

```
a.shape=(8,8)
```

```
a
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47],
       [48, 49, 50, 51, 52, 53, 54, 55],
       [56, 57, 58, 59, 60, 61, 62, 63]])
```

The **shape** of any array can be changed as long as new shape has same number of elements as old.

Alternately you can use the a**.reshape(n,m)** method to return a reshaped ndarray.

```
a.shape=(4,4,4)
```

```
a
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]],

       [[16, 17, 18, 19],
        [20, 21, 22, 23],
        [24, 25, 26, 27],
        [28, 29, 30, 31]],

       [[32, 33, 34, 35],
        [36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59],
        [60, 61, 62, 63]]])
```

2D arrays

default indexing is a[row, column], (known as “row major”).

e.g.,
a[1, 2] is 10
a[4,7] is 39

slicing is same as 1D, but with two sets of indices

a[: : , : :]

```
a
```

col 0 col 1   col 2 …

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47],
       [48, 49, 50, 51, 52, 53, 54, 55],
       [56, 57, 58, 59, 60, 61, 62, 63]])
```

row 0
row 1
row 2
.
.
.

How to display values in red box?

2D arrays

default indexing is a[row, column], (known as "row major").

e.g.,
a[1, 2] is 10
a[4,7] is 39

slicing is same as 1D, but with two sets of indices
a[: : , : :]

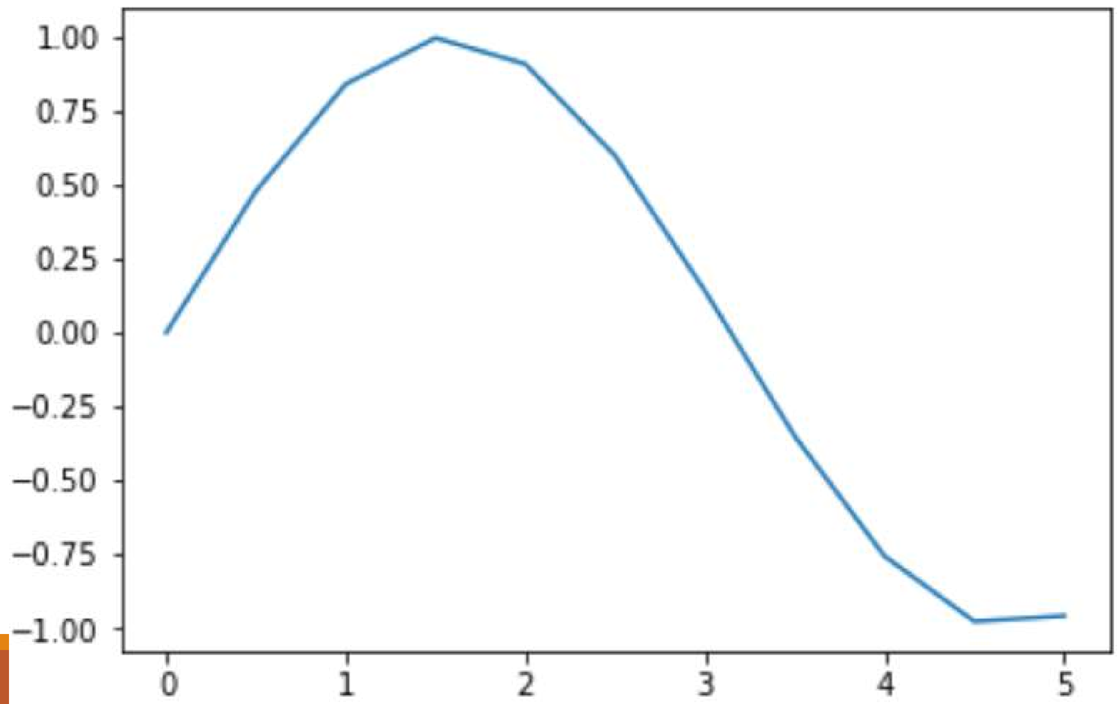
```
a
                 col 0  col 1   col 2  ...

array([[ 0,  1,  2,  3,  4,  5,  6,  7],        row 0
       [ 8,  9, 10, 11, 12, 13, 14, 15],        row 1
       [16, 17, 18, 19, 20, 21, 22, 23],        row 2
       [24, 25, 26, 27, 28, 29, 30, 31],        .
       [32, 33, 34, 35, 36, 37, 38, 39],        .
       [40, 41, 42, 43, 44, 45, 46, 47],        .
       [48, 49, 50, 51, 52, 53, 54, 55],
       [56, 57, 58, 59, 60, 61, 62, 63]])
```

red box corresponds to a[2:7,3:5]

Simple 2D plot with matplotlib

```python
import numpy as np
import matplotlib.pyplot as plt
N=11
xmin=0
xmax=5
x=np.linspace(xmin,xmax,N) # returns np.array with N members,
                           # equally spaced from xmin to xmax.
y=np.sin(x) # y is np.array of length N, with y[i]=sin(x[i])
plt.plot(x,y)
#plt.show() # might be needed to show plot
```



x and y labels

In [189]:

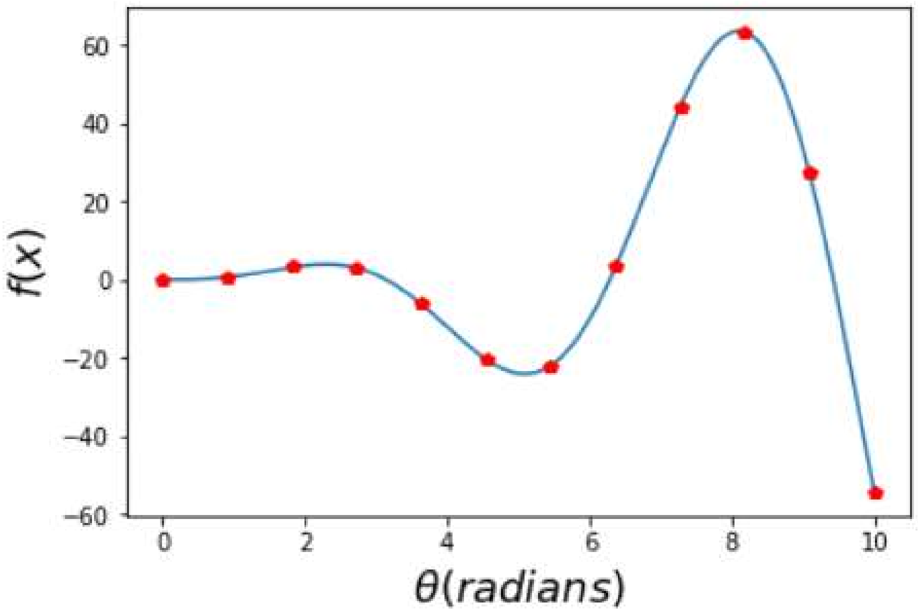
```
x=np.linspace(0,10,1000)
y=x**2 * np.sin(x)
plt.plot(x,y)

x2=np.linspace(0,10,12)
y2=x2**2 * np.sin(x2)
plt.plot(x2,y2,'rp')

plt.xlabel("$\\theta (radians)$", fontsize=18)  #\t is interpreted as tab, needs to be escaped
plt.ylabel("$f(x)$",fontsize=18)

plt.show()
```

Tex code can be used within the axes labels.



Activity 3

The following activity should be done in ***Jupyter notebook***. When finished, export (or print) the notebook to a pdf to upload to LMS.

1) Plot of the function $y = \sin^2(x)/\sqrt{x}$ over the x-range of π to 2π. Your x-axis should be labeled as θ and your y-axis should be labeled as $f(\theta)$

2) Use LaTeX markup to display the integral

$$\int_{\pi}^{2\pi} \frac{sin^2(x)}{x^{1/2}}\,dx$$

3) Numerically perform the integration in (2) using np.trapz or your own code to add up the areas of small rectangles under the curve.