FIT3181 Deep Learning

Week 00: Extra Content – Python

Dr. Trung Le
trunglm@monash.edu

Acknowledge Prof. Dinh Phung for this material.

GROUP OF EIGHT AUSTRALIA

Faculty of Information Technology, Monash University, Australia

Quick tour of python for ML

Python

- A modern, general-purpose, object-oriented and high-level programming language
  - can be used in interactive and script mode
    - Interactive: read and execute line by line (e.g., use in python notebook).
    - Script: execute the entire file (save the .py file and run from the terminal).
- Why python?
  - Clean and simple
    - Easy-to-read and intuitive code
    - Easy-to-learn minimalistic

Hello World!

- Code for the usual “Hello, World!” program (only for Python 2.x)

helloworld.py

```
print “Hello, World”
```

- This also works for both versions (2.x and 3.x)

helloworld2.py

```
print(“Hello, World”)
```

Executing a Python file

- Using Python interpreter

```
$ python helloword.py
```

```
Hello World
```

- Using IPython

```
$ ipython helloword.py
```

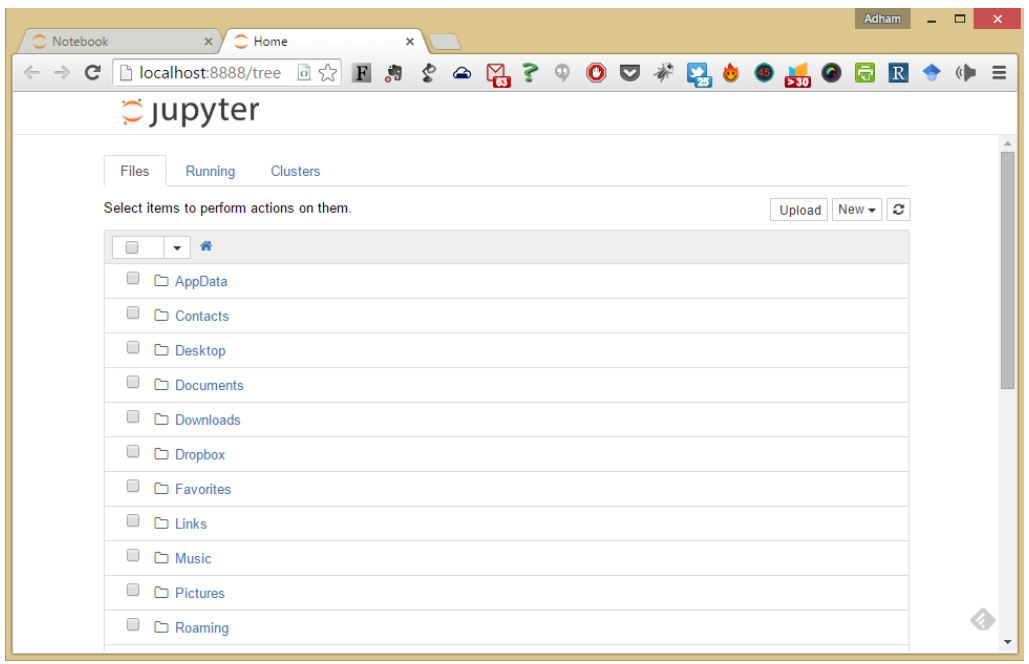
```
Hello World
```

Jupyter (Ipython) Notebook

- Start Jupyter Notebook Server

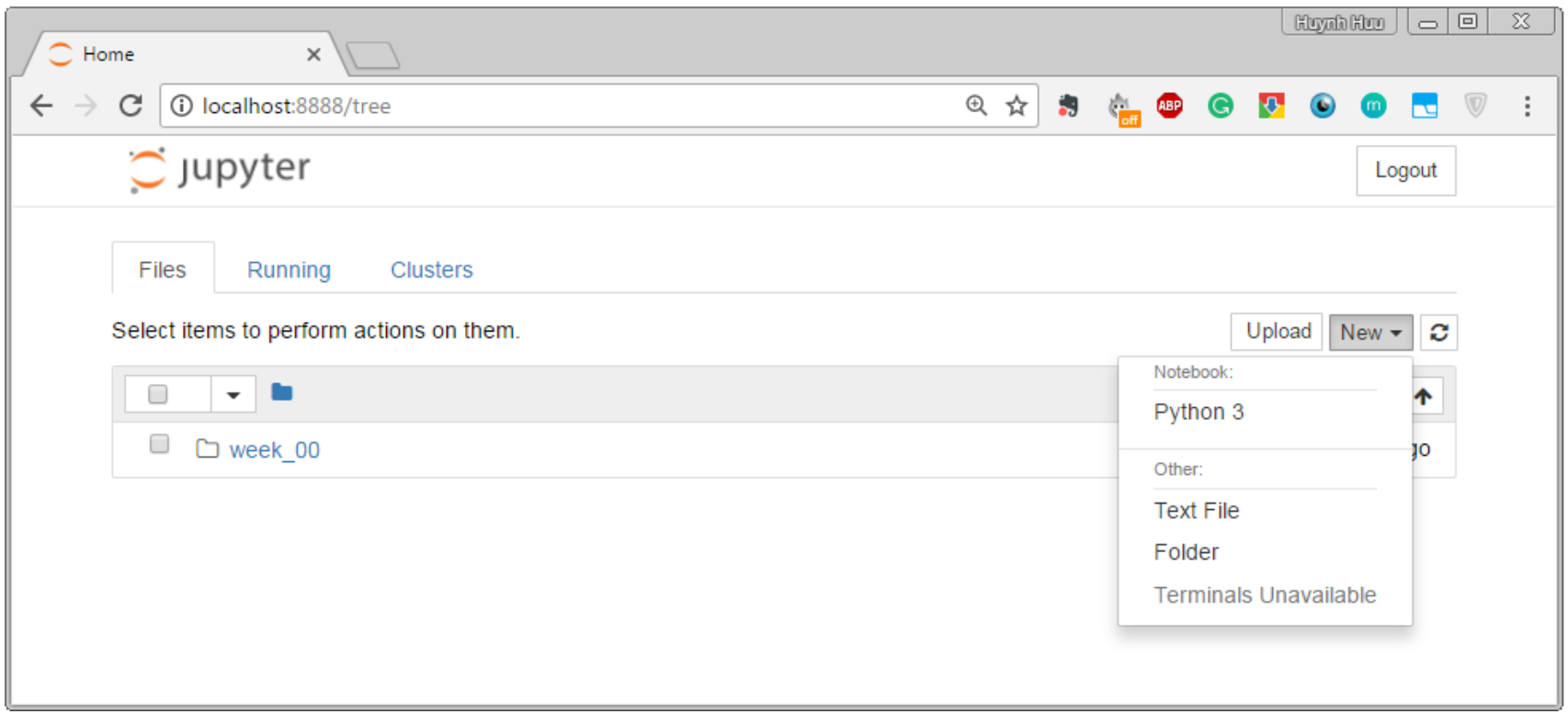
```
$ jupyter notebook [DIR]
```

```
...
The Ipython Notebook is running at: http://localhost:8888/
...
```



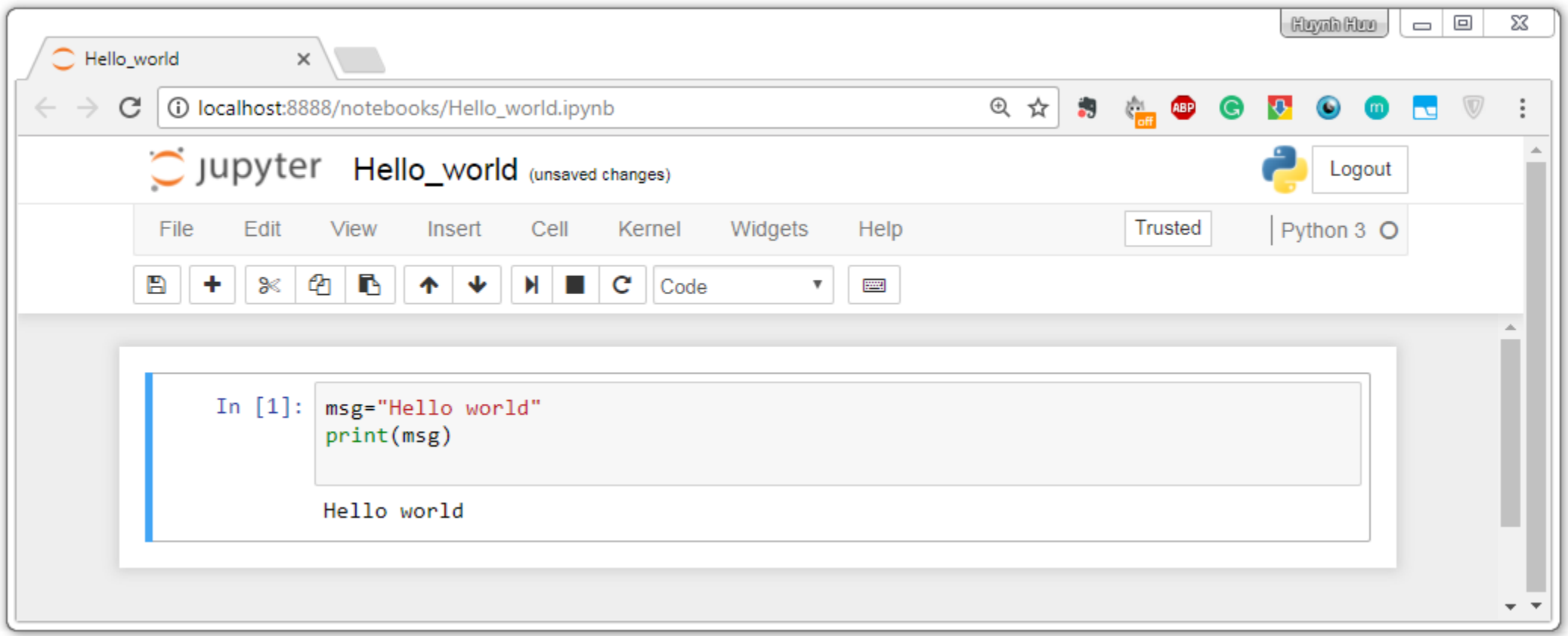
Jupyter (Ipython) Notebook

- To create a new notebook, click on New -> Python 3



IPython Notebook

- To run a code cell in a notebook, press **Ctrl+Enter**.
- Or, **Shift-Enter** (which automatically moves to the next shell)



Comments

- Single-line Commenting

```
# This is a one-line comment

Print(“Hello, World”)
```

```
Hello World
```

- Multi-line Commenting

```
'''
This is a
Multi-line comment
'''
Print(“Hello, World”)
```

```
Hello World
```

Fundamental components

- Data types
  - String, Number, Boolean, etc.
- Operations
  - Arithmetic operators, Relational operators, Logical Operators
- Input and Output
- Functions

Variables

- Variable names can contain a-z, A-Z, 0-9, _
  - convention: variables start with a lower-case letter and classes with upper-case.
- A variable is created when a value is assigned to it

```
x = 10
y = “I am a string”

print("x = ", x)       # use a comma to print multiple statements
print("y = {}".format(y))     # or use string formatter
```

```
x = 10
y = I am a string
```

Variables Types

- Not explicitly specified, variables have types
  - The type is derived from the value that was assigned to it

```
y = “I am a string”

type(y)
```

```
str
```

```
x = 1

type(x)
```

```
int
```

Variables Types

- Fundamental types
  - Integer
  - Float
  - Boolean
  - Complex
- Compound types
  - List
  - String
  - Tuple
  - Dictionary

Variables Types

- Integer

```
x = 1

type(x)
```

```
int
```

- Float

```
x = 1.0

type(x)
```

```
float
```

Variables Types

- Boolean

```
b1 = True
b2 = False

type(b1)
```

```
bool
```

- Complex

```
x = 1.0 - 1.0j

type(x)
```

```
complex
```

Variables Types

- **List**, [item_0, item_1, …, item_n]
  - items need not to be of the same type

```
l = [10, 2, 30, 4, 50, 6, 70, 8]
print(type(l))
print(l)
print(len(l))
```

```
<type 'list'>
[10, 2, 30, 4, 50, 6, 70, 8]
8
```

```
l1 = [1, 'a', 1.0, 1-1j]
l2 = [1, 2, ['a', 1.0], 3]

print(l1, l2)
```

```
[1, 'a', 1.0, 1-1j] [1, 2, ['a', 1.0], 3]
```

Variables Types

- Indexing
  - index starts at 0

```
l = [10, 2, 30, 4, 50, 6, 70, 8]

print(l[0]) # The first element
print(l[3]) # The fourth element
print(l[-1]) # The last element
print(l[-2]) # The second last element
```

```
10
4
8
70
```

Variables Types

- **String**
  - similar to a list of characters

```
s = “Hello World!”

print(type(s))
print(len(s))
```

```
<type ‘str’>
12
```

```
s = “Hello World!”

print(s[3])
print(s[0])
```

```
l
H
```

Variables Types

- Slicing to extract a part of a list/string
  - [start:stop]
  - [start:]

```
l = [1, 2, 3, 4, 5, 6, 7, 8]
s = “This is a string”

print(l[2:5])
print(s[0:6])
print(s[0:-1])
```

```
[3, 4, 5]
This i
This is a strin
```

Variables Types

- Slicing forward

```
s = “This is a string”

print(s[0:10:2]) # 2: step
print(s[:10:2])
```

```
Ti sa
Ti sa
```

- Slicing backward

```
print(s[10:2:-1])
print(s[::-1])
```

```
s a si s
gnirts a si sihT
```

Variables Types

- **Tuples**
  - immutable lists

```
states = ('VIC', 'NSW', 'QLD', 'WA', 'ACT', 'NT', 'TAS', 'SA')

print(type(states))
print(states[1])
```

```
<type 'tuple'>
NSW
```

Variables Types

- **Dictionary**
  - list of key-value pairs

```
params = {“param1”: 1.0,
          “param2”: “a string”,
          “param3”: [1, 2, 3]}

print(type(params))
print(params)
```

```
<type ‘dict’>
{‘param1’: 1.0, ‘param3’: [1, 2, 3], ‘param2’: ‘a string’}
```

Operators

- Operators
  - Arithmetic: + - * / // % **
  - Boolean: and or not
  - Comparison: > < >= <= == is identical

```
l1 = [1, 2]
l2 = [1, 2]
print(l1 is l2, l1 == l2)

l1 = l2 = [1, 2]
print(l1 is l2, l1 == l2)
```

```
False, True
True, True
```

Control Flow

- if, elif, else
  - note the indentation

```
statement1 = False
statement2 = False

if statement1:
    print("statement1 is True")

elif statement1:
    print("statement2 is True")

else:
    print("statement1 and statement2 are False")
```

```
Statement1 and statement2 are False
```

Loops

- **for** loops
  - works on any iterator

```
l1 = [1, 2, 3, 4, 5]

for item in l1:
    print(item)
```

```
1 2 3 4 5
```

```
course = [“an”, ”introduction”, “to”, “data”, “science”]

for word in course:
    print word
```

```
an introduction to data science
```

Loops

- enumerate

```
course = [“an”, ”introduction”, “to”, “data”, “science”]

for i, word in enumerate(course):
    print(i, word)
```

```
0 an
1 introduction
2 to
3 data
4 science
```

- list comprehension

```
l1 = [x**2 for x in range(2, 6)]

print(l1)
```

```
[4, 9, 16, 25]
```

Loops

- **while** loop
  - Pay attention to the indentation, **print(**“done”**)** is outside the loop

```
i = 0

while i < 5:
    print(i)
    i += 1

print “done”
```

```
0 1 2 3 4 done
```

Functions

- keyword **def:**
  - Pay attention to the indentation

```
def powers(x):
    """
    Returns a few powers of x.
    """
    return x**2, x**3, x**4
```

```
print(powers(3))

x1, x2, x3 = powers(3)
print(x2)
```

```
(9, 27, 81)
27
```

Anonymous Functions

- keyword **lambda**

```
def f(x):
    return x**2

g = lambda x: x**2
```

```
Print(f(8), g(8))
```

```
64 64
```

```
def n_increment(n):
    return lambda x: x+n

f5 = n_increment(5)
f9 = n_increment(9)

print(f5(2), f9(2))
```

```
7 11
```

NumPy

- Fundamental package for scientific computing
  - powerful N-dimensional arrays
  - broadcasting functions
  - integrating with C/C++/Fortran
  - linear algebra, Fourier transform, random numbers, …

Importing NumPy

```
import numpy
```

- Convention

```
import numpy as np
```

Arrays
From a list

- Use ‘np.array()’ function

```
x = [1, 7, 3, 0, -4]
y = np.array(x)
y
```

```
array([1, 7, 3, 0, -4])
```

Arrays

From a range

```
print(np.array(range(5)))
print(np.arange(2, 3, 0.2))
print(np.linspace(2, 3, 5))
print(np.logspace(2, 3, 5))
```

```
[0 1 2 3 4]
[ 2.  2.2 2.4 2.6 2.8]
[ 2.  2.25 2.5 2.75 3. ]
[ 100. 177.827941 316.22776602 562.34132519 1000. ]
```

- help!

```
logspace?
```

Arrays

Pre-filled

```
print(np.zeros(5))
print(np.ones(5, dtype=int))
```

```
[0. 0. 0. 0. 0.]
[1 1 1 1 1]
```

- Also check out `zeros_like()` and `ones_like()`

Arrays
Attributes

- Many attributes and methods are associated with arrays, such as:

```
y = np.array([3, 0, -4, 6, 12, 2])
print(y.ndim)
print(y.shape)
print(y.dtype)
print(y.max())
print(y.argmax())
print(y.mean())
```

```
1
(6, )
int32
12
4
3.1666
```

Arrays
Multi-dimensional

- 2D or higher

```
x = [[1, 2, 10, 20], [3, 4, 30, 40]]
y = np.array(x)

print(y)
print(y.ndim, y.shape)
```

```
[[1 2 10 20]
 [3 4 30 40]]

2, (2, 4)
```

Arrays
Multi-dimensional

```
x = np.zeros((3, 5), dtype='int')

print(x)
print(x.ndim, x.shape)
```

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

2, (3, 5)
```

```
np.diag([1, 2, 3])
```

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

Array Manipulating

Indexing

```
x = np.array([2, 8, -2, 4, 3])
print(x[3])
```

```
4
```

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

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

```
3
[4 9 7 3 1]
[4 3]
```

Array Manipulating

Indexing with other arrays

```
x = np.array([2, 8, -2, 4, 3, 6, 9])

idx1 = [1, 3, 4]          # list
idx2 = np.array(idx1)     # array

print(x[idx1], x[idx2])
```

```
[8 4 3], [8 4 3]
```

```
x[idx2] = 0

x
```

```
[2 0 -2 0 0 6 9]
```

Array Manipulating

Index masks

```
x = np.array([2, 8, -2, 4, 3, 9, 0])
mask = np.array([False, True, True, False, False, True, False])

x[mask]
```

```
array([8, -2, 9])
```

```
x[idx2] = 0

x
```

```
[2 0 -2 0 0 6 9]
```

Array Manipulating

Index masks + comparison operators

```
x = np.array([2, 8, -2, 4, 3, 9, 0])
mask = (x>=2) * (x<9)

x[mask]
```

```
array([2, 8, 4, 3])
```

Array Manipulating

Slicing

```
x = np.array([2, 8, -2, 4, 3, 9, 0])

print(x[3:7])
print(x[3:7:2])
```

```
[4 3 9 0]
[4 9]
```

```
x = np.array([[7, 6, 8, 6, 4, 3],
              [4, 7, 0, 5, 9, 5],
              [7, 3, 6, 3, 5, 1]])

print(x[1, 1:4])
print(x[:2, 1::2])
```

```
[7 0 5]

[[6 6 3]
 [4 5 5]]
```

Array Manipulating

Iteration

- Since most of NumPy functions support vectors, in many cases iteration over items is not necessary.

```
a = np.arange(0, 50, 7)
for item in a:
    print (item)
```

```
0 7 14 21 28 35 42 49
```

```
# iterate using indices
a = np.arange(0, 50, 7)
for i in range(a.shape[0]):
    print (a[i])
```

```
0 7 14 21 28 35 42 49
```

Array Manipulating

copy()

- Python uses references not values

```
x = [1, 2, 3]
y = x

y[0] = 0     # now we alter an element of y
print (x, y)   # note that x has changed as well
```

```
[0, 2, 3] [0, 2, 3]
```

```
x = np.array([1, 2, 3])
y = x

y[0] = 0     # now we alter an element of y
print (x, y)   # note that x has changed as well
```

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

Array Manipulating

copy()

- Use **copy()** to obtain a copy not view

```
x = np.array([1, 2, 3])
y = x.copy()      # or np.copy(x)

y[0] = 0     # now we alter an element of y
print (x, y)   # note that x has not changed as well
```

```
[1 2 3] [0 2 3]
```

Array Manipulating

reshape()

```
x = np.arange(6)
y = x.reshape((2, 3))       # or np.reshape(x, (2, 3))

print (x)
print (y)
```

```
[0 1 2 3 4 5]

[[0 1 2]
 [3 4 5]]
```

Array Manipulating

astype()

- Type casting

```
x1 = np.arange(5)
x2 = x1.astype(float)

print (type(x1), x1)
print (type(x2), x2)
```

```
<type 'numpy.ndarray'> [0 1 2 3 4]
<type 'numpy.ndarray'> [0. 1. 2. 3. 4.]
```

Array Manipulating

transpose

- Keyword: T

```
x1 = np.random.randint(5, size=(2, 4))
x2 = x1.T

print (x1)
print (x2)
```

```
[[1 0 2 4]
 [4 2 3 2]]

[[1 4]
 [0 2]
 [2 3]
 [4 2]]
```

Array Operation

Arithmetic

- Operators are element-wise
  - + - * /

```
x1 = np.array([[2, 3, 5, 7],
               [2, 4, 6, 8]], dtype=float)
x2 = np.array([[6, 5, 4, 3],
               [9, 7, 5, 3]], dtype=float)

print (x1 + x2)
print (x1 / x2)
```

```
[[ 8.  8.  9. 10.]
 [11. 11. 11. 11.]]

[[0.33 0.6  1.25 2.33]
 [0.22 0.57 1.2  2.6]]
```

Array Operation

Arithmetic

```
x1 = np.array([[2, 3, 5, 7],
               [2, 4, 6, 8]], dtype=float)

print (3 + x1)
```

```
[[8. 6. 8. 10.]
 [5. 7. 9. 11.]]
```

```
x1 = np.array([[2, 3, 5, 7],
               [2, 4, 6, 8]], dtype=float)

print (3 / x1)
```

```
[[1.5  1.    0.6  0.428]
 [1.5  0.75  0.5  0.375]]
```

Array Operation

Boolean

```
x1 = np.array([2, 3, 5, 7])
x2 = np.array([2, 4, 6, 8])

print (x1<x2)
```

```
[False True True False]
```

```
y = x1<x2

print (y.all())    # all items are True
print (y.any())  # at least one item is True
```

```
False
True
```

Random Numbers

- Draw from standard normal

```
print (np.random.rand())

print (np.random.random([2, 3]))
```

```
0.75

[[0.42 0.68, 0.79],
 [0.09 0.55, 0.47]]
```

```
print (np.random.rand(2, 3))
```

```
[[0.26 0.45 0.83]
 [0.53 0.17 0.89]]
```

Random Numbers

- Random seed

```
for i in range(5):
    print (np.random.rand())
```

```
0.42 0.68, 0.79 0.09 0.55
```

```
for i in range(5):
    np.random.seed(123)
    print (np.random.rand())
```

```
0.26 0.26 0.26 0.26 0.26
```

Vectorizing functions

```
def step_func(x):
    if x>=0:
        return 1
    else:
        return 0
```

```
step_func(np.array([2, 7, 0, -1]))
```

```
ValueError
```

```
step_func_vectorized = np.vectorize(step_func)
step_func_vectorized(([2, 7, 0, -1]))
```

```
array([1, 1, 1, 0])
```

Python Programming

Python Modules and Packages

- Store your useful function definitions in a file.
- Import it as a module in your current program

myfunctions.py

```
def func1(x):
    statement(s)

def func2(x):
    statement(s)

def func3(x):
    statement(s)
```

```
>>> import myfunctions
```

Imports all functions in myfunctions.py to current namespace

```
>>> myfunctions.func1(5)
```

Python Packages

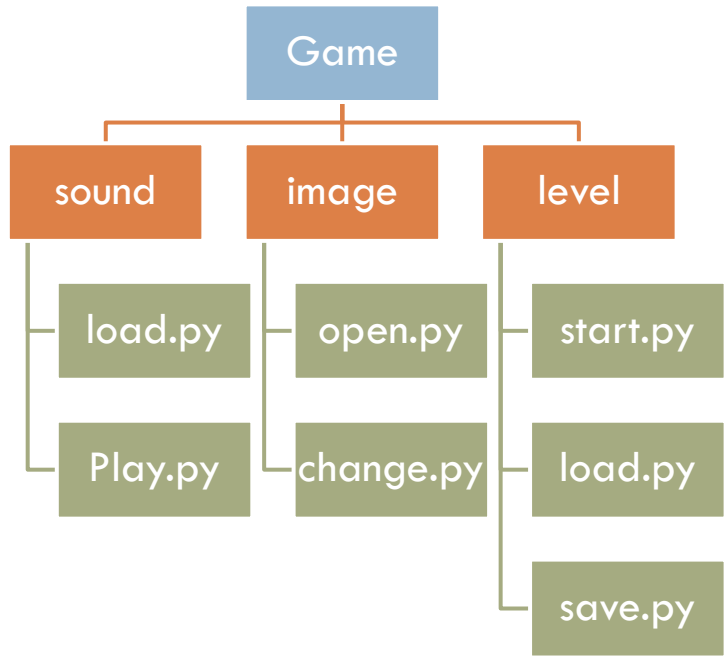
- Python package: directory of modules

Import all modules into our program

```
import game
```

Individual modules can be referenced as:

```
game.sound.load()
game.level.start()
```



Another way to do this:

```
from game import *
```

Python Packages

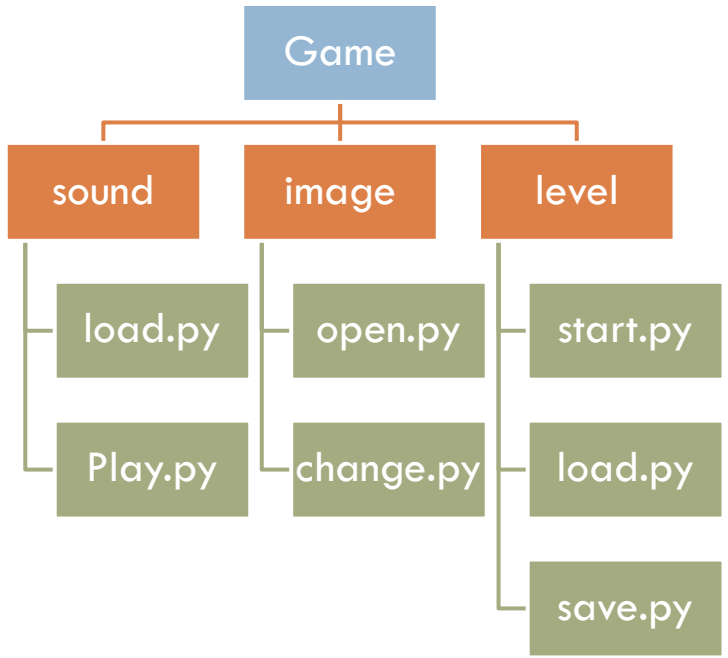
- You can also import specific modules

Import only the modules in level package

```
from game.level import *
```

Import a specific module

```
from game.level import load
```

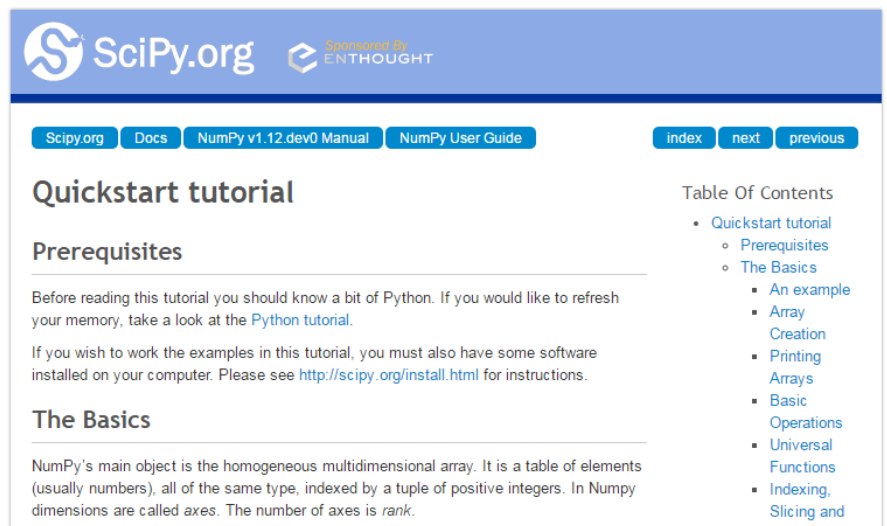


Most heavy lifting is done

- We will use standard modules, packages and libraries
- Contains all popular functions and algorithms

Numpy

- Fundamental package for scientific computing
- Support for large multidimensional arrays and matrices
- Support for high-level mathematical operations on these arrays and matrices
- Similar to lists, but more capable, and fixed size.



https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Numpy

- Import the package numpy:

```
import numpy as np
```

- All functions can now be accessed as `np.<function_name()>`

- for multidimensional numeric data (arrays) and related arithmetic operations.

- Similar to lists, but more capable, and fixed size.

Numpy: Examples

```
import numpy as np

# One dimensional array
a = np.array([0,1,2,3])

# Two dimensional array
b = np.array([[0,1], [2,3]])

#Get the dimensions and shape
a.ndim
a.shape
b.ndim
b.shape
```

```
>>> a =  np.array([1,2,3], float)
>>> b =  np.array([5,2,6], float)
>>> a + b
```

array([6., 4., 9.])

```
>>> a - b
```

array([-4., 0., -3.])

```
>>> a * b
```

array([5., 4., 18.])

```
>>> b/a
```

array([5., 1., 2.])

Numpy: Popular functions

- Random number generation
  - draw random integers, floats from different probability distributions

- Linear Algebra functions
  - dot product, outer product
  - matrix decompositions
  - Matrix inversion
  - Eigenvalues and norms

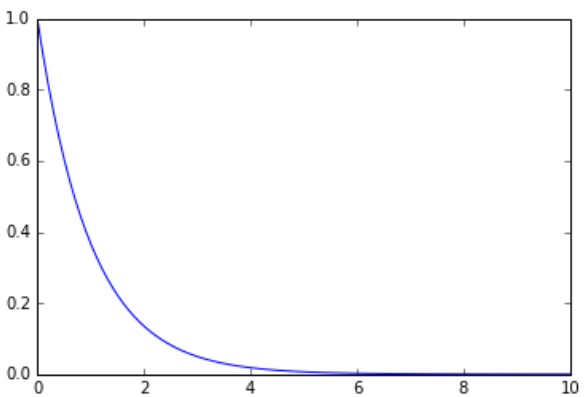
- Statistics
  - Max, min, percentile of values
  - Mean, median, variance of values
  - correlation and histograms

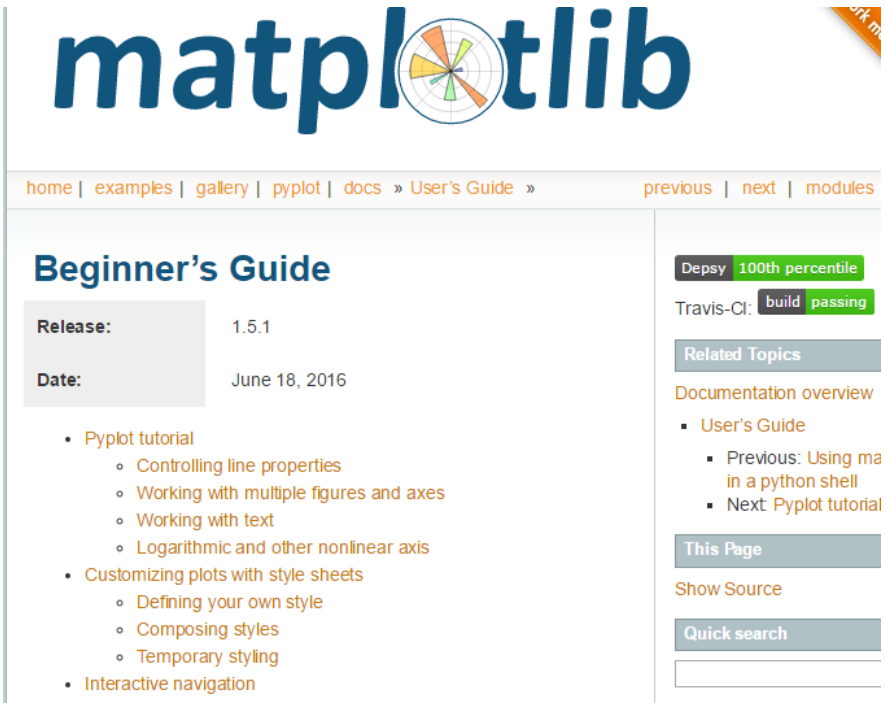
Matplotlib: Plotting in Python

- Open source plotting library for python

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

a = np.linspace(0,10,100)
b = np.exp(-a)
plt.plot(a,b)
plt.show()
```





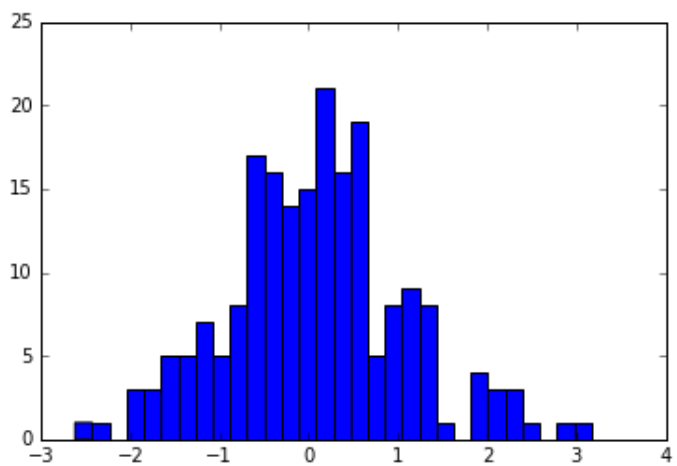
http://matplotlib.org/users/beginner.html

Matplotlib: Example

- Plot a histogram with 30 bins for 200 data points from a normal distribution

```
import matplotlib.pyplot as plt
from numpy.random import normal

x = normal(size=200)
plt.hist(x,bins=30)
plt.show()
```



With matplotlib you can also:

- Label the axis
- Change colors of the plot
- Insert plot legend
- Insert other text

Lot of example code from:
http://matplotlib.org/examples/

Scipy

- Package for scientific computing
- Builds on Numpy objects
- Contains functions for:
  - optimization
  - linear algebra
  - integration and interpolation
  - Signal and Image processing
  - Differential equation solvers

http://docs.scipy.org/doc/scipy-0.17.1/reference/

Important Subpackages

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

Example:

```
from scipy import linalg, optimize
```

Scipy Example

- Lets try to find the minimum of a function

```
import numpy as np
from scipy.optimize import fmin

x = np.arange(-3,3)
a,b,c = 3,2,5

y = lambda x: a*x**2 + b*x + c

solution = fmin(y, 3)
print(solution)
```

```
Optimization terminated successfully.
         Current function value: 4.666667
         Iterations: 19
         Function evaluations: 38
[-0.3333252]
```

- fmin(): uses simplex method
- More complex methods need function derivative as input