Python Tutorial

Basic Python and Linear Algebra Applications

Installing Python

https://wiki.python.org/moin/BeginnersGuide/Download

The Python Interpreter

Which editors are good to use?

- Sublime (https://www.sublimetext.com/)
- Notepad++ (https://notepad-plus-plus.org/)

Running Python files

Some tutorial

Etc etc, main content. Perhaps:

1. Basics (if, elif, variables, importance of indentation, etc)
2. Importing libraries, examples of good libraries
3. Arrays
4. Matrices
   [http://www.bogotobogo.com/python/python_numpy_matrix_tutorial.php, http://cs231n.github.io/python-numpy-tutorial/]

REPL

- Read Evaluate Print Loop (AKA an interpreter)

```
ryin@Raymonds-MBP:~$ python3
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('sup everyone')
sup everyone
>>> 5 + 6
11
>>> 
```

- Get information with `dir()`, `help()`, `type()`
- A great place to try things out if you are unsure!

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Identifiers, names, variables

- All 3 mean the same thing
- Variable naming convention
  - Functions and variables: lower_with_underscore
    - `my_num = 5`
  - Constants: UPPER_WITH_UNDERSCORE
    - `SECONDS_PER_MINUTE = 60`

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Binding

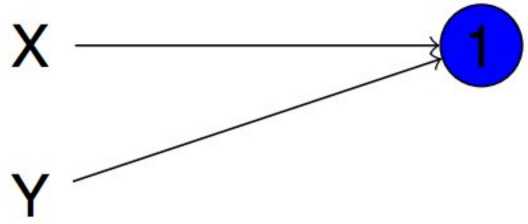
x = 1

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Binding

x = 1

y = x



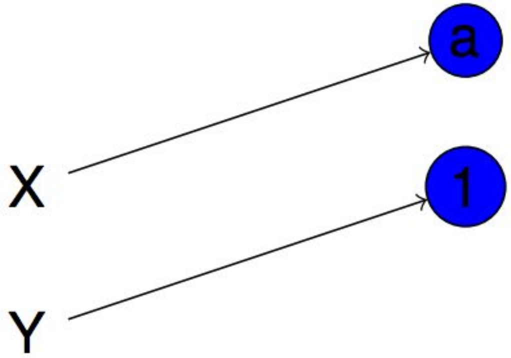
source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Binding

x = 1

y = x

x = ‘a’



source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Objects

Python treats all data as objects

Identity

Memory address: Does not change

Type

Does not change

Value

Mutable: value can be changed (e.g. `[1,2]`) - Has both deep and shallow copy methods

Immutable: value cannot be changed after creation (e.g. `(1,2)`) - Only has shallow copy

Equality

Use `is` for referential equality (do x and y point to the same object?)

Use == for structural equality (are x and y equal based on object's __eq__ method?)

'ś'

Types

Every object has a type

- Inspect types with `type(object)`
- `isinstance(object,type)` checks type hierarchy
- Types can be compared for equality, but you usually want `isinstance`

Some types:

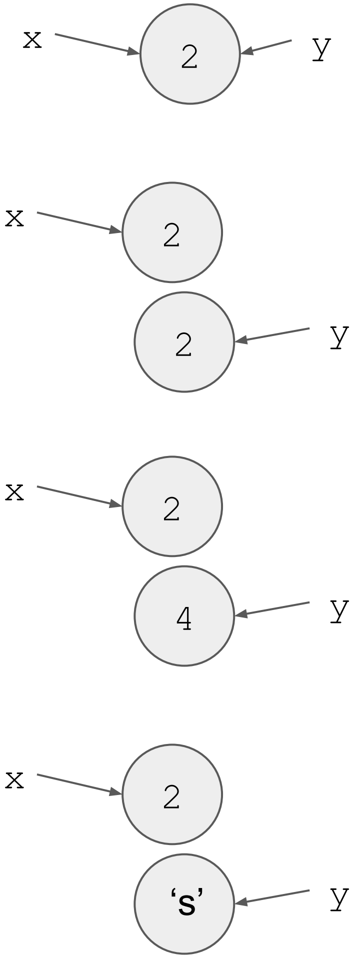
int, float str, tuple, list, dict range, bool, None, function

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Booleans

- Boolean values are `True` and `False`
- Boolean statements are combined with `and`, `or`
- If statement example:

```
if x is y:
   print(“x and y are the same object”)
elif x == y:
   print(“x and y are equivalent objects”)
elif type(x) == int and type(y) == int:
   print(“x and y are ints”)
else:
   print(“x and y are different”)
```



Booleans

Any object can be tested for truth value for use in conditionals, or as operands of the mentioned Boolean operations.

Considered “falsy”:

- `None`
- `0`
- `0.0`
- Any empty string/sequence/collection (`[]`, `()`, etc.)

Operations

```
x = 3
print type(x) # Prints "<type 'int'>"
print x       # Prints "3"
print x + 1   # Addition; prints "4"
print x - 1   # Subtraction; prints "2"
print x * 2   # Multiplication; prints "6"
print x ** 2  # Exponentiation; prints "9"
x += 1
print x  # Prints "4"
x *= 2
print x  # Prints "8"
y = 2.5
print type(y) # Prints "<type 'float'>"
print y, y + 1, y * 2, y ** 2 # Prints "2.5 3.5 5.0 6.25"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Range

Immutable sequence of numbers

`range(stop), range(start,stop), range(start,stop,step)`

- start defaults to 0 step defaults to 1
- All numbers in [start,stop), where we increment start by step
- Negative steps are valid

Memory efficient: Calculates values as you iterate over them

Functions

Functions are first class

They're objects, too!

- Can pass them as arguments
- Can assign them to variables

Define functions with a def

return keyword to return a value

If a function reaches the end of the block without returning, it will return None (null)

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Importing modules

Allow use of other python files and libraries with imports:

```
import math
```

- Named imports: `import math as m`
- Specific imports: `from math import pow`
- Import all: `from math import *`

source: https://www.cis.upenn.edu/~cis192/files/lec/lec1.pdf

Lists

- `list()` and `[]` are both new empty lists
- Comma separated [1, 2, 3] and nested [[1, 2], [3, 4]]
- Construct from iterable: `list(range(3))`
- Concatenating two lists with + creates a new list.
- Lists are mutable
- Implemented as a resizable array in CPython

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

Indexing & slicing

- Index with square brackets
- Negative indexing gets elements from the end of list
  - `lst[-1]` is the last element
  - `lst[-2]` is the second to last element
- Can index multiple times with `lst_of_lst[][]`

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

The right way to iterate

- Iterate with `for x in lst:`
  - Then use `x` in the loop
- <u>Never</u> do `for i in range(len(lst)):`
- Index and value with `for i,x in enumerate(lst):`
  - Useful if you sometimes want `lst[2*i]` or `other_list[i]`

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

Multiplication and copies

- Multiplying a list adds it to itself.
  - The component lists are not copies, they're the same object
- Shallow copy a list with `lst[:]`
  - If x is a list:
  - `y = x # This is a deep copy: x and y are the same object. Changing y will change x, changing x will change y.`
  - `y = x[:] # This is a shallow copy: x and y contain the same values but changing y will NOT change x.`
- Use the `copy` module for deep copy
  - `copy.deepcopy(lst)`

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

Tuples

- Immutable lists.
- Standard notation is (a, b, c, d)
  - The parentheses aren't necessary though.
- Support unpacking:
  - `x, y, z = t`, where t is a 3 element tuple
  - This is most often seen in functions returning multiple values as tuples.
- Write (x,) for a single element tuple.

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

Dictionaries

- A dictionary is a hash map
    - It hashes the keys to lookup values
    - Keys must be immutable so that the hash doesn't change
- `dict()` and `{}` are empty
- `dict([(k1, v1), (k2, v2)]) or {k1:v1, k2:v2}`
- `dict(zip(key_lst, val_lst))`
- `d[k]` accesses the value mapped to `k`
- `d[k] = v` updates the value mapped to `k`

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

Methods

- `len()`, `in`, and `del` work like lists
- `d.keys()` and `d.values()` return views of the keys and values.
  - Views support iteration, `len()`, and `in`
  - **Views change when the dictionary changes**
- `d.items()` is a view of `(k,v)` pairs
- `d.get(k,x)` looks up the value of `k`.
  - Returns `x` if `k` not in `d`
- `d.pop(k,x)` returns and remove value at `k`.
  - Returns `x` as default

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf

List comprehensions

- `[expr for v in iter]`
- `[expr for v1,v2 in iter]`
- `[expr for v in iter if cond]`
- **`res = [v1 * v2 for v1, v2 in lst if v1 > v2]`**
- Translation:
  - ```
    res = []
    for v1, v2 in lst:
        if v1 > v2:
            res.append(v1 * v2)
    ```

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf
    ```

List comprehensions

- **`[x for x in lst1 if x > 2 for y in lst2 for z in lst3 if x + y + z < 8]`**
- Translation:
  - ```
    res = []
    for x in lst1:
        if x > 2:
            for y in lst2:
                for z in lst3:
                    if x + y + z > 8:
                        res.append(x)
    ```

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf
    ```

Dict comprehensions

- Like lists but swap [] for {}
- Starts with: `d = dict()`
- Appends with: `d[k] = v`
- **`{k: v for k,v in lst}`**
- Translation:
  - ```
    d = dict()
    for k, v in lst:
        d[k] = v
    ```

source: https://www.cis.upenn.edu/~cis192/spring2015/files/lec/lec2.pdf
    ```

Classes

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print 'HELLO, %s!' % self.name.upper()
        else:
            print 'Hello, %s' % self.name

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

source: cs231n.github.io/python-numpy-tutorial/

numpy

```
import numpy as np
```

- Matrix and vector operations!

```
a = np.array([1, 2, 3])       # Create a rank 1 array
print type(a)                      # Prints "<type 'numpy.ndarray'>"
print a.shape                      # Prints "(3,)" (Note: this array cannot be transposed)

print a[0], a[1], a[2]        # Prints "1 2 3"
a[0] = 5                      # Change an element of the array
a_new = np.reshape(a,[1,-1])
print a_new                   # Prints "[[5, 2, 3]]"
print a_new.shape             # Prints "(1,3)" (Note: reshaping allows for transposes)

b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print b.shape                     # Prints "(2, 3)"
print b[0, 0], b[0, 1], b[1, 0]   # Prints "1 2 4"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Functions to create arrays

```
import numpy as np

a = np.zeros((2,2))  # Create an array of all zeros
print a              # Prints "[[ 0.  0.]
                     #          [ 0.  0.]]"

b = np.ones((1,2))   # Create an array of all ones
print b              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Create a constant array
print c               # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"

d = np.eye(2)        # Create a 2x2 identity matrix
print d              # Prints "[[ 1.  0.]
                     #          [ 0.  1.]]"

e = np.random.random((2,2)) # Create an array filled with random values
print e                     # Might print "[[ 0.91940167  0.08143941]
                            #               [ 0.68744134  0.87236687]]"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Indexing and slicing

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print a[0, 1]   # Prints "2"
b[0, 0] = 77    # b[0, 0] is the same piece of data as a[0, 1]
print a[0, 1]   # Prints "77"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Indexing and slicing

```
a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.

print bool_idx      # Prints "[[False False]
                    #          [ True  True]
                    #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print a[bool_idx]  # Prints "[3 4 5 6]"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Mutating elements

```
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print a  # prints "array([[ 1,  2,  3],
         #                [ 4,  5,  6],
         #                [ 7,  8,  9],
         #                [10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print a[np.arange(4), b]  # Prints "[ 1  6  7 11]"

# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print a  # prints "array([[11,  2,  3],
         #                [ 4,  5, 16],
         #                [17,  8,  9],
         #                [10, 21, 12]])
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Array Math - Elementwise Operations

```
# Elementwise addition:
print x + y
print np.add(x, y)


# Elementwise subtraction
print x - y
print np.subtract(x, y)


# Elementwise multiplication:
print x * y
print np.multiply(x, y)
```

```
# Elementwise division:
print x / y
print np.divide(x, y)


# Elementwise square root
print np.sqrt(x)

# Elementwise power
print x**2
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Array Math - Matrix Operations

```
import numpy as np
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print v.dot(w)
print np.dot(v, w)

# Matrix / vector product; both produce the
# rank 1 array [29 67]
print x.dot(v)
print np.dot(x, v)
```

```
# Matrix / matrix product; both produce the
# rank 2 array
# [[19 22]
#  [43 50]]
print x.dot(y)
print np.dot(x, y)
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Other useful operations

```
# searching
x = np.array([[1,2],[3,4]])
print 4 in x       #True
print 8 in x       #False

print np.sum(x)  # Compute sum of all elements; prints "10"
print np.sum(x, axis=0)  # Compute sum of each column; prints "[4 6]"
print np.sum(x, axis=1)  # Compute sum of each row; prints "[3 7]"

# Transpose
print x.T #x transpose
# CAREFUL with 1D arrays! .T does nothing
v = np.array([1,2,3])
print v    # Prints "[1 2 3]"
print v.T  # Prints "[1 2 3]"
v.shape()  # Prints "(3,)"
```

```
# This can be fixed (more on slide 30):
v1 = np.array([[1,2,3]])
print v1     # Prints "[[1 2 3]]"
print v1.T   # Prints          [[1]
                                [2]
                                [3]]
v1.shape()   # Prints "(1,3)"
```

source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Other useful operations

```
# creating repeating arrays
v = np.array([5, 1, 9])
vv = np.tile(v, (4, 1)) # creates a new array:
```

$$\begin{bmatrix} 5 & 1 & 9 \\ 5 & 1 & 9 \\ 5 & 1 & 9 \\ 5 & 1 & 9 \end{bmatrix}$$

```
# Here the arrays are not the same shape but one of the dimensions
# matches. Python guesses that you want to add a copy of v
# to each element of x. This called broadcasting
x = np.array([[1,2,3], [4,5,6]])
print x + v

# Other functions support broadcasting universal functions
http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs
```

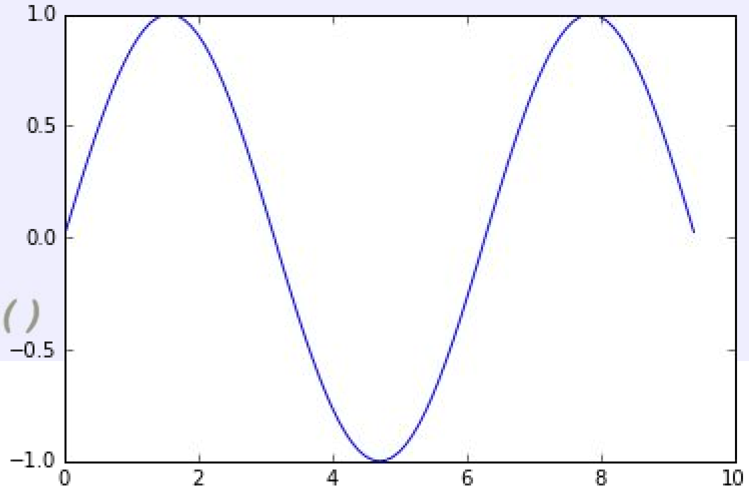
source: cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Plotting

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

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show()
```

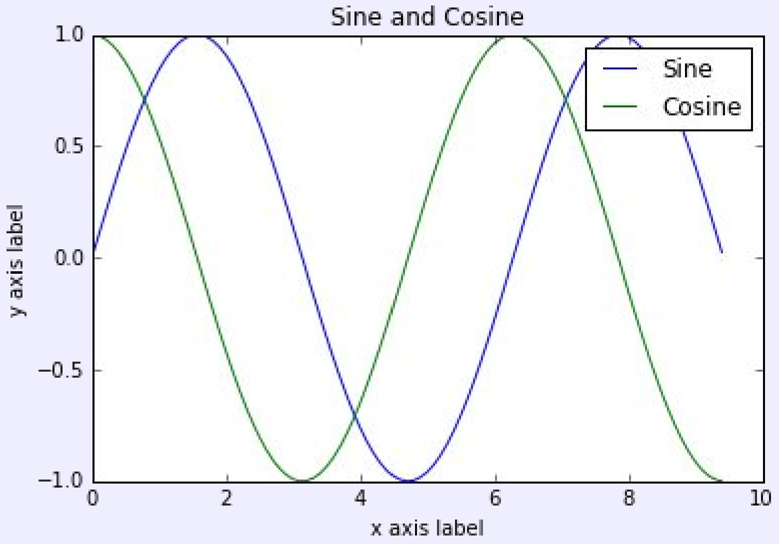


source: cs231n.github.io/python-numpy-tutorial/#matplotlib

Plotting

```python
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



source: cs231n.github.io/python-numpy-tutorial/#matplotlib

Common Errors

- ~~np.zeros(3,4)~~ np.zeros((3,4))
- First element in array a[0] not ~~a[1]~~
- http://imgur.com/WRuJV6r (A flow chart to find the source of common errors.)
- A = B the “deep copy” vs. A = np.copy(B) “the shallow copy”
- Sneaky integer math: 1/2 = 0 where 1.0/2 = 0.5 (depends on Python version)

Resources

- Numpy manual: http://docs.scipy.org/doc/numpy/
- Numpy list of matrix functions: http://docs.scipy.org/doc/numpy/reference/routines.array-manipulation.html
- More numpy tutorials: http://cs231n.github.io/python-numpy-tutorial/
- Plotting: http://cs231n.github.io/python-numpy-tutorial/#matplotlib
- Code Academy interactive course: https://www.codecademy.com/learn/python
- An Introduction to Numpy and Scipy: http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf