Assignment 1 - Lab Exercises

1. Short Python Practice

Task: Create a class named Dog

- Each instance of the class should have a variable, name (should be set on creating one)
- The class has two main methods, action and age
- Action is an instance method taking in an *optional* parameter, quietly. Calling action prints "WOOF WOOF WOOF " if nothing is passed in, "woof " if quietly is passed in as True. (Use only one hardcoded string in the method)
- Age is a class method that takes in a list of ages in human years and returns it in dog years (Use list comprehension)

In [0]:
```
class Dog:

    def __init__(self, name):
        # complete the class
        self.name = name

    def action(self, quietly=False):
        bark = "woof "
        if quietly:
            print(bark.strip())
        else:
            print((3*bark.upper()).rstrip())

    def age(humans):
        return [i*7 for i in humans]
```

In [0]:
```
human_ages = [3, 4, 7, 4, 10, 6]
```

1. Create an instance of this class, and print its name
2. Call action with both possible options for the parameter
3. Call age on the provided array above

In [0]:
```
# complete the task above
fido = Dog("Fido")
print(fido.name)
fido.action()
fido.action(quietly=True)
Dog.age(human_ages)
```

```
Fido
WOOF WOOF WOOF
woof
```

Out[0]:
```
[21, 28, 49, 28, 70, 42]
```

2. Numpy Practice

Numpy is a commonly used library in Python for handling data, especially helpful for handling arrays/multi-dimensional arrays. It's particularly important for helping bridge the ineffiencies of Python as a high level language with the improved performance of handling data structures in low level languages like C.

Part 1: General Numpy Array Exercises

1. Create a matrix, with shape 10 x 7, of ones
2. Create 10 matrices, with shape 100 x 70, of random integers from -10 to 10
3. Print the number of elements equal between a pair of matrices, for each possible pair in the 10 created above except for with itself (Don't compare the 1st with the 1st)

In [0]:
```
import numpy as np

# example
die_roll = np.random.randint(1, 6)
print("Rolled a " + str(die_roll))
```

```
Rolled a 3
```

In [0]:
```
#complete part 1
uno = np.ones((10,7))
print(uno)
```

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

In [0]:
```
mats = []
for i in range(10):
    mats.append(np.random.randint(-10,10,size=(100,70)))
```

In [0]:
```
for i in range(10):
    for j in range(i+1, 10):
        common = np.sum(mats[i]==mats[j])
        print("Matrix "+ str(i)+" and matrix "+str(j)+" have "+ str(common)
        +" elements in common")
```

```
Matrix 0 and matrix 1 have 365 elements in common
Matrix 0 and matrix 2 have 357 elements in common
Matrix 0 and matrix 3 have 374 elements in common
Matrix 0 and matrix 4 have 358 elements in common
Matrix 0 and matrix 5 have 325 elements in common
Matrix 0 and matrix 6 have 311 elements in common
Matrix 0 and matrix 7 have 362 elements in common
Matrix 0 and matrix 8 have 350 elements in common
Matrix 0 and matrix 9 have 326 elements in common
Matrix 1 and matrix 2 have 333 elements in common
Matrix 1 and matrix 3 have 373 elements in common
Matrix 1 and matrix 4 have 335 elements in common
Matrix 1 and matrix 5 have 322 elements in common
Matrix 1 and matrix 6 have 359 elements in common
Matrix 1 and matrix 7 have 342 elements in common
Matrix 1 and matrix 8 have 344 elements in common
Matrix 1 and matrix 9 have 355 elements in common
Matrix 2 and matrix 3 have 346 elements in common
Matrix 2 and matrix 4 have 369 elements in common
Matrix 2 and matrix 5 have 357 elements in common
Matrix 2 and matrix 6 have 369 elements in common
Matrix 2 and matrix 7 have 362 elements in common
Matrix 2 and matrix 8 have 373 elements in common
Matrix 2 and matrix 9 have 342 elements in common
Matrix 3 and matrix 4 have 345 elements in common
Matrix 3 and matrix 5 have 315 elements in common
Matrix 3 and matrix 6 have 378 elements in common
Matrix 3 and matrix 7 have 345 elements in common
Matrix 3 and matrix 8 have 367 elements in common
Matrix 3 and matrix 9 have 331 elements in common
Matrix 4 and matrix 5 have 308 elements in common
Matrix 4 and matrix 6 have 384 elements in common
Matrix 4 and matrix 7 have 357 elements in common
Matrix 4 and matrix 8 have 369 elements in common
Matrix 4 and matrix 9 have 324 elements in common
Matrix 5 and matrix 6 have 339 elements in common
Matrix 5 and matrix 7 have 334 elements in common
Matrix 5 and matrix 8 have 346 elements in common
Matrix 5 and matrix 9 have 366 elements in common
Matrix 6 and matrix 7 have 362 elements in common
Matrix 6 and matrix 8 have 369 elements in common
Matrix 6 and matrix 9 have 348 elements in common
Matrix 7 and matrix 8 have 340 elements in common
Matrix 7 and matrix 9 have 371 elements in common
Matrix 8 and matrix 9 have 308 elements in common
```

Part 2: Splicing and some Numpy Methods

1. Find the pseudoinverse of one of the above matrices (or create a new one with shape 100 x 70)
2. Turns out the last 40 rows and last 10 columns of data were useless. Set a new variable to the matrix used above, without the last 40 rows or 10 columns.
3. Find the inverse for this square matrix.

Optional: You can use the same command for 1 and 3 (briefly explain why if you do)

In [0]:
```
# complete part 2
mat = mats[0]
pinv = np.linalg.pinv(mat)
mat_square = mat[:-40, :-10]
inv = np.linalg.pinv(mat_square)
```

In [0]:
```
print(mat.shape)
print(pinv.shape)
print(mat_square.shape)
print(inv.shape)
```

```
(100, 70)
(70, 100)
(60, 60)
(60, 60)
```

Part 3: Matrix Methods

1. Generate 2 more matrices, *a* and *b* with shape 2 x 3 and 4 x 3.
2. Create 2 matrices, *a_on_b* and *b_on_a* - for the first, stack a on top of b, and the opposite for the second. The join them to create *abba*, with *b_on_a* to the right of *a_on_b*
3. Print the right eigenvalues and eigenvectors for *abba*

In [0]:
```
# complete part 3
a = np.random.randint(-10, 10, size=(2,3))
b = np.random.randint(-10, 10, size=(4,3))
a_on_b = np.vstack((a,b))
b_on_a = np.vstack((b,a))
abba = np.hstack((a_on_b, b_on_a))
print(a)
print(b)
print(abba)
```

```
[[ 2 -5 -8]
 [ 5  2  8]]
[[  6  -3  -1]
 [  6  -7  -4]
 [  7   8   6]
 [  6  -1 -10]]
[[  2  -5  -8   6  -3  -1]
 [  5   2   8   6  -7  -4]
 [  6  -3  -1   7   8   6]
 [  6  -7  -4   6  -1 -10]
 [  7   8   6   2  -5  -8]
 [  6  -1 -10   5   2   8]]
```

In [0]:
```
print(np.linalg.eig(abba))
```

```
(array([ 6.22019279+14.90272269j,  6.22019279-14.90272269j,
       -3.48443388 +9.49878874j, -3.48443388 -9.49878874j,
       -2.01124785 +0.j        ,  8.53973004 +0.j        ]), array([[ 0.4397589
-0.05839195j,  0.4397589 +0.05839195j,
         0.01880205+0.07369621j,  0.01880205-0.07369621j,
        -0.5855294 +0.j        ,  0.21492057+0.j        ],
       [ 0.00403524-0.32072171j,  0.00403524+0.32072171j,
         0.61316062+0.j        ,  0.61316062-0.j        ,
         0.45052556+0.j        , -0.27216933+0.j        ],
       [ 0.02333355-0.47111122j,  0.02333355+0.47111122j,
        -0.34449578+0.22272504j, -0.34449578-0.22272504j,
        -0.19001174+0.j        , -0.21087539+0.j        ],
       [ 0.50277566+0.j        ,  0.50277566-0.j        ,
         0.0073965 -0.00460599j,  0.0073965 +0.00460599j,
         0.59650993+0.j        , -0.4525855 +0.j        ],
       [-0.05409778-0.19424935j, -0.05409778+0.19424935j,
         0.28109597-0.52675325j,  0.28109597+0.52675325j,
         0.21056003+0.j        , -0.5489487 +0.j        ],
       [ 0.24603627-0.35193317j,  0.24603627+0.35193317j,
        -0.30560246-0.00359126j, -0.30560246+0.00359126j,
        -0.13385795+0.j        ,  0.5736604 +0.j        ]]))
```

3. Matplotlib Practice

Matplotlib is a commonly used visualization library.

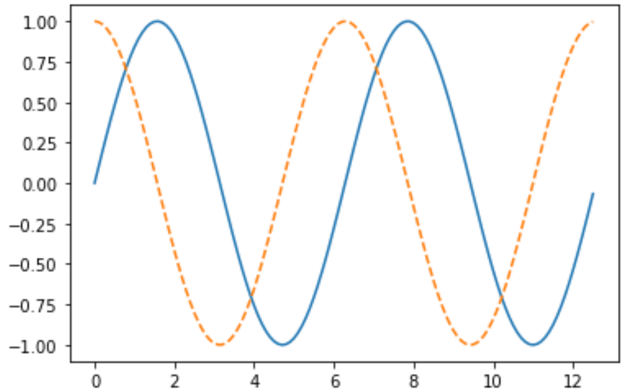
Part 1: Generate plots

1. Generate sine (y_sin) and cosine (y_cos) data for x from 0 to 4 * pi (Use np.arange with step size 0.1)
2. Create a plot with both on the same chart. The sine wave should be solid, cosine dashed
3. Create a second plot with 2 subplots, with the first plotting the sine wave and second plotting the cosine wave

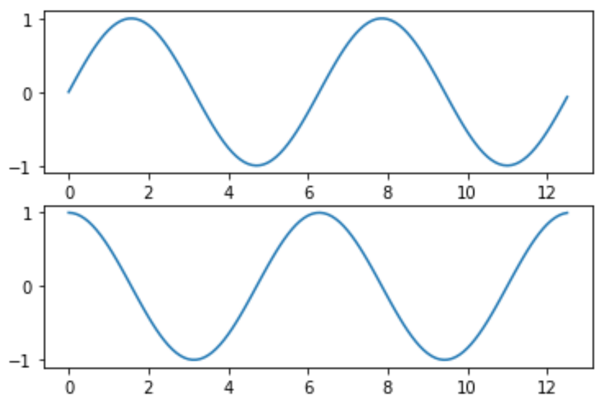
In [0]:
```
import matplotlib.pyplot as plt
```

In [0]:
```
# complete 1
x = np.arange(0, 4*np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
```

In [0]:
```
# complete 2
overlayed = plt.figure()
plt.plot(x, y_sin, "-")
plt.plot(x, y_cos, "--")
plt.show()
```



In [0]:
```
# complete 3
subplotted = plt.figure()
plt.subplot(2,1,1)
plt.plot(x, y_sin)
plt.subplot(2,1,2)
plt.plot(x, y_cos)
plt.show()
```



Part 2: Save Output

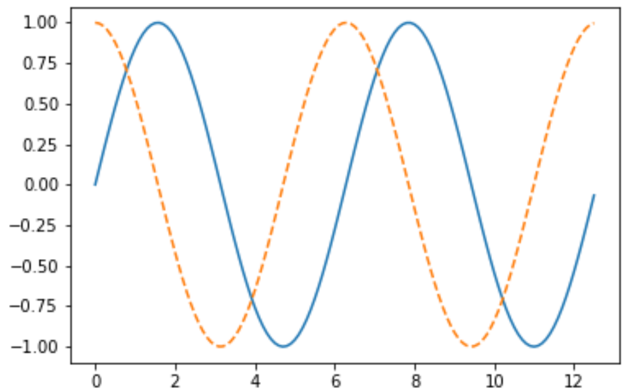
1. For the above plots, save each as a image file.
2. Open both files in this notebook

In [0]:
```
# complete 1
overlayed.savefig("overlayed.png")
subplotted.savefig("subplotted.png")
```

In [0]:
```
from IPython.display import Image
```

In [0]:
```
# complete 2
Image("overlayed.png")
```

Out[0]:



In [0]:
```
Image("subplotted.png")
```

Out[0]: