Programming in Python

BBM103 Introduction to Programming Lab 1
Week 11

Fall 2016
NumPy (http://www.numpy.org) is the fundamental package for scientific computing with Python.

NumPy provides the most important data types for econometrics, statistics and numerical analysis – arrays and matrices.
Array Creation

```python
import numpy as np
x = np.array([1, 2, 3])
print(x)
y = np.arange(10)  # like Python's range, but returns an array
print(y)
```

Output:
```
[1 2 3]
[0 1 2 3 4 5 6 7 8 9]
```

Basic Operations

```python
a = np.array([1, 2, 3, 6])
b = np.linspace(0, 2, 4)  # create an array with four equally spaced points starting with 0 and ending with 2.
c = a - b
print(c)
print(a**2)
```

Output:
```
[ 1. 1.33333333 1.66666667 4.]
[ 1 4 9 36]
```
Universal functions

```python
84 a = np.linspace(-np.pi, np.pi, 100)
85 b = np.sin(a)
86 c = np.cos(a)
87
88 print(a)
```

Output:

```
-1.9991533 -1.93572881 -1.87226229 -1.80879577 -1.74532925 -1.68186273
-1.61839622 -1.5549297 -1.49146318 -1.42799666 -1.36453014 -1.30106362
-1.23759711 -1.17413059 -1.11066407 -1.04719755 -0.98373103 -0.92026451
-0.856798 -0.79333148 -0.72986496 -0.66639844 -0.60293192 -0.53946541
-0.47599889 -0.41253237 -0.34906585 -0.28559933 -0.22213281 -0.1586663
-0.09519978 -0.03173326 0.03173326 0.09519978 0.1586663 0.22213281
0.28559933 0.34906585 0.41253237 0.47599889 0.53946541 0.60293192
0.66639844 0.72986496 0.79333148 0.856798 0.92026451 0.98373103
1.04719755 1.11066407 1.17413059 1.23799666 1.30106362 1.36453014
1.42799666 1.49146318 1.5549297 1.61839622 1.68186273 1.74532925
1.80879577 1.87226229 1.93572881 1.9991533 2.06266184 2.12612836
2.18959488 2.2530614 2.31652792 2.37999443 2.44346095 2.50692747
2.57039399 2.63386051 2.69732703 2.76079354 2.82426006 2.88772658
2.9511931 3.01465962 3.07812614 3.14159265]
```
Linear Algebra

```
from numpy.random import rand
from numpy.linalg import solve, inv

a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
print(a)
print(a.transpose())

b = np.array([3, 2, 1])
print(b)

print(solve(a, b)) # solve the equation ax = b

c = rand(3, 3) # create a 3x3 random matrix

print(c)

print(np.dot(a, c)) # matrix multiplication
```

**Output:**

```
[[ 1.  2.  3.]
 [ 3.  4.  6.7]
 [ 5.  9.  5.]]

[[ 1.  3.  5.]
 [ 2.  4.  9.]
 [ 3.  6.7  5.]]

[[-2.27683616  0.96045198  0.07909605]
 [ 1.04519774 -0.56497175  0.1299435 ]
 [ 0.39548023  0.05649718 -0.11299435]]

[3 2 1]

[-4.83050847  2.13559322  1.18644068]

[[ 0.83843849  0.99200584  0.3483117 ]
 [ 0.1630258   0.54165777  0.34870755]
 [ 0.02351926  0.67038873  0.28482933]]

[[  1.23504786   4.08648756   1.90021478]
 [  3.32499769   9.63425306   4.34812178]
 [  5.77702094  13.18689276   6.30407306]]
```
Elementary statistics

Example: Mean

def mean_func(lst):
    mn = sum(lst)/len(lst)
    return mn
data_in = [2.75, 2.86, 3.37, 2.76, 2.62, 3.49, 3.05, 3.12]
print ("The mean is %.4f." % mean_func(data_in))

Output:
The mean is 3.0025.

Example: Mean with NumPy

import numpy as np
data_in = [2.75, 2.86, 3.37, 2.76, 2.62, 3.49, 3.05, 3.12]
mean_data = np.mean(data_in)
print ("The mean is ",mean_data)

Output:
The mean is 3.0025
**Example: Median**

```python
def median(lst):
    quotient, remainder = divmod(len(lst), 2)
    if remainder:
        return sorted(lst)[quotient]
    return sum(sorted(lst)[quotient - 1:quotient + 1]) / 2.

data_in = [2.75, 2.86, 3.37, 2.76, 2.62, 3.49, 3.05, 3.12]
print(median(data_in))
```

**Output:**

```
2.955
```

---

**Example: Median with NumPy**

```python
data_in = [2.75, 2.86, 3.37, 2.76, 2.62, 3.49, 3.05, 3.12]
median_data = np.median(data_in)
print("The median is:", median_data)
```

**Output:**

```
The median is: 2.955
```
Example: Mod

```python
from collections import Counter
mode_data = [1, 2, 4, 1, 5, 3, 2, 1, 5]
mode_counts = Counter(mode_data)  # turn the list into a Counter object
print(mode_counts)  # a python dictionary
max_counts = mode_counts.most_common(1)
print(max_counts)
```

Output:
Counter({1: 3, 2: 2, 5: 2, 3: 1, 4: 1})
[(1, 3)]
Example: Correlation with numpy

```
108 a=np.correlate([1, 2, 3], [0, 1, 0.5])
109 print(a)
110 b=np.correlate([1, 2, 3], [0, 1, 0.5], "same")
111 print(b)
112 c=np.correlate([1, 2, 3], [0, 1, 0.5], "full")
113 print(c)
```

Output:

```
[ 3.5]
[ 2.  3.5  3. ]
[ 0.5  2.  3.5  3.  0. ]
```
**Example:** Variation with numpy

```python
117 a = np.array([[1, 2], [3, 4]])
118 print(np.var(a))
119 print(np.var(a, axis=0))
120 print(np.var(a, axis=1))
```

Output:

```
1.25
[ 1.  1.]
[ 0.25  0.25]
```

In single precision, `var()` can be inaccurate:

```python
123 a = np.zeros((2, 512*512), dtype=np.float32)
124 a[0, :] = 1.0
125 a[1, :] = 0.1
126 print(a)
127 print(np.var(a))
```

```
[[ 1.  1.  1. ...,  1.  1.  1. ]
 [ 0.1 0.1 0.1 ...,  0.1 0.1 0.1]]
0.2025
```

Computing the variance in float64 is more accurate:

```python
131 print(str(np.var(a, dtype=np.float64)))
```

```
0.202499999329
```
RECURSION
In order to understand recursion, you must understand recursion.

RECURSION
In order to understand recursion, you must understand recursion.

RECURSION
In order to understand recursion, you must understand recursion.

RECURSION
In order to understand recursion, you must understand recursion.

RECURSION
In order to understand recursion, you must understand recursion.
Example: Fibonacci Numbers

The Fibonacci numbers are the numbers of the following sequence of integer values:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
The Fibonacci numbers are defined by:
\[ F_n = F_{n-1} + F_{n-2} \]
with \( F_0 = 0 \) and \( F_1 = 1 \)

```python
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

number = input("Please enter a number to print fibonacci numbers!")
print(fibonacci(int(number)))
```

Output:
Please enter a number to print fibonacci numbers!4
3
Example: Calculate factorial

We can track how the function works by adding two print() function to the previous function definition:

```python
def factorial(n):
    print("factorial has been called with n = " + str(n))
    if n == 1:
        return 1
    else:
        res = n * factorial(n-1)
        print("intermediate result for " + str(n) + " * factorial(\"n-1, \")\" : ", res)
    return res

number=input("Please enter a number to calculate factorial! ")
print(factorial(int(number)))
```

Output:

Please enter a number to calculate factorial! 5
factorial has been called with n = 5
factorial has been called with n = 4
factorial has been called with n = 3
factorial has been called with n = 2
factorial has been called with n = 1
intermediate result for 2 * factorial(1): 2
intermediate result for 3 * factorial(2): 6
intermediate result for 4 * factorial(3): 24
intermediate result for 5 * factorial(4): 120
120
Example: Calculate factorial

```python
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

number=input("Please enter a number to calculate factorial!")
print(factorial(int(number)))
```

Output:

Please enter a number to calculate factorial! 5
120
Example: Visualizing Recursion

```python
import turtle

def tree(branchLen,t):
    if branchLen > 5:
        t.forward(branchLen)
        t.right(20)
        tree(branchLen-15,t)
        t.left(40)
        tree(branchLen-15,t)
        t.right(20)
        t.backward(branchLen)

def main():
    t = turtle.Turtle()
    myWin = turtle.Screen()
    t.left(90)
    t.up()
    t.backward(100)
    t.down()
    t.color("green")
    tree(75,t)
    myWin.exitonclick()

main()
```

Output:
Example: Computing Exponent

We can compute exponent in fewer steps if we use successive squaring.

```python
def exp(x, n):
    ""
    Computes the result of x raised to the power of n.
    ""
    if n == 0:
        return 1
    else:
        return x * exp(x, n-1)

number1=input("print a number as base")
number2=input("print a number as exponent")
print(exp(int(number1),int(number2)))
```

```
def fast_exp(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return fast_exp(x*x, n/2)
    else:
        return x * fast_exp(x, n-1)

number1=input("print a number as base")
number2=input("print a number as exponent")
print(fast_exp(int(number1),int(number2)))
```
Example: Flatten a List

```python
def flatten_list(a, result=None):
    if result is None:
        result = []
    for x in a:
        if isinstance(x, list):
            flatten_list(x, result)
        else:
            result.append(x)
    return result
listToFlat=[[1, 2, [3, 4]], [5, 6], 7]
print(listToFlat)
flatList=flatten_list(listToFlat)
print(flatList)
```

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