# Lecture_2

January 14, 2023

### 0.1 Exercise 1

[2]:

```
i = 0
while 2**i < 10000:
    print(2**i)
    i = i + 1
```

```
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
```

## 1 LEC 2: NumPy (Numerical Python)

- Most common package for scientific computing with Python
- Its fundamental object is `np.array`, an multidimensional array of numbers
- Provides linear algebra, Fourier transform, random number capabilities
- Building block for other packages (e.g. SciPy, scikit-learn)
- Open source, huge dev community!

[3]:

```
# Quick note on importing
import math
math.sin(5)
```

[3]:

```
-0.9589242746631385
```

[2]:

```
import math as m
m.sin(5)
```

```
[2]: -0.9589242746631385
```

```
[2]: import numpy as np
```

### 1.1 Array

Main object type is `np.array`

Many ways to create it,

One way is to convert a python list

```
[3]: python_list = [ 1,2,3 ]
     print(python_list)
     np.array(python_list)
```

```
[1, 2, 3]
```

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

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

```
[4]: array([1, 2, 3, 5, 6])
```

```
[5]: arr = np.random.random(15)
     arr
```

```
[5]: array([0.73087874, 0.76142338, 0.20573295, 0.99084899, 0.37736725,
            0.16554228, 0.85789842, 0.7875271 , 0.25418106, 0.1612428 ,
            0.71284205, 0.78641188, 0.81007235, 0.31059417, 0.99748959])
```

Many times a list comprehension is used to create a list and then converted to a array

```
[8]: arr = np.array([ 2**i for i in [2,3,9] ])
     arr
```

```
[8]: array([  4,   8, 512])
```

```
[42]: arr = np.array([ 2**i for i in range(10) if i != 4 and i%2 == 0 ])
      arr
```

```
[42]: array([  1,   4,  64, 256])
```

```
[12]: a = np.array([1, 2, 3, 4, 4, 5], str) # the int/float/str in last specifies␣
       ↪that everything in integer/float/string inside the array
      b = np.array([1, 4.0,"sita", 3, 4, 4, 5])
      print(a)
      print(b)
```

```
['1' '2' '3' '4' '4' '5']
['1' '4.0' 'sita' '3' '4' '4' '5']
```

[8]:

```
print(np.linspace(0,100, 4))# NumPy linspace function always returns evenly spaced numbers based on a given interval
print(np.logspace(0,1,5)) # same as linspace but gives logrithmic values instead # default base = 10
print(np.logspace(0,1,5, base=2))
```

```
[  0.          33.33333333  66.66666667 100.        ]
[ 1.          1.77827941  3.16227766  5.62341325 10.        ]
[1.         1.18920712 1.41421356 1.68179283 2.        ]
```

[26]:

```
print(np.arange(1,100)) # arange similar to range function but it generates an array instead of list
print(np.arange(1,100,2))
print(np.arange(100,1,-1))
```

```
[ 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 64 65 66 67 68 69 70 71 72
 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
 97 98 99]
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
 97 99]
[100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83
  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65
  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47
  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29
  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11
  10   9   8   7   6   5   4   3   2]
```

2D Array

[20]:

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

[20]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

[17]:

```
M.ndim
```

[17]:

```
2
```

[18]:

```
M.size
```

[18]:

```
6
```

[19]:

```
M.shape
```

[19]:

```
(2, 3)
```

3D Array

Collection of two $2 \times 2$ arrays

[22]:

```
D = np.array( [ [ [5,6], [7,8] ], [ [2,3], [9,0] ]  ] )
D
```

[22]:

```
array([[[5, 6],
        [7, 8]],

       [[2, 3],
        [9, 0]]])
```

[24]:

```
D.ndim
```

[24]:

```
3
```

[25]:

```
D.shape
```

[25]:

```
(2, 2, 2)
```

### 1.1.1 Exercise 1

Create a numpy array that contain intergers i such that 0<i<100 and $2^i$ has the last digit 6

[41]:

```
a = np.array([ 2**i for i in range(100)])
```

[35]:

```

```

[35]:

```
array([16, 256, 4096, 65536, 1048576, 16777216, 268435456, 4294967296,
       68719476736, 1099511627776, 17592186044416, 281474976710656,
       4503599627370496, 72057594037927936, 1152921504606846976,
       18446744073709551616, 295147905179352825856,
       4722366482869645213696, 75557863725914323419136,
       1208925819614629174706176, 19342813113834066795298816,
       309485009821345068724781056, 4951760157141521099596496896,
       79228162514264337593543950336], dtype=object)
```

### 1.1.2 Exercise 2

Create a 2D numpy array $A$ such that $A_{ij} = i \times j$

[5]:

```
r = int(input("Enter no. of rows: "))
c = int(input("Enter no. of columns: "))
```

```
np.array( [ [ i*j for j in range(1,c+1)] for i in range(1,r+1) ] )
```

```
Enter no. of rows: 3
Enter no. of columns: 3
```

```
[5]: array([[1, 2, 3],
            [2, 4, 6],
            [3, 6, 9]])
```

### 1.2 Array Operations

Can be done with arrays of same dimensions only.

```
[15]: a1 = np.array([7,8,9,1])
      a2 = np.array([3,2,1,8])
```

```
[14]: len(a2)
```

```
[14]: 4
```

```
[16]: sum(a1)
```

```
[16]: 25
```

```
[17]: print(a1+a2, a1-a2, a1*a2, a1/a2, a1%a2, a1//a2, 2*a1, 1/a2)
```

```
[10 10 10  9] [ 4  6  8 -7] [21 16  9  8] [2.33333333 4.         9.
0.125     ] [1 0 0 1] [2 4 9 0] [14 16 18  2] [0.33333333 0.5        1.
0.125     ]
```

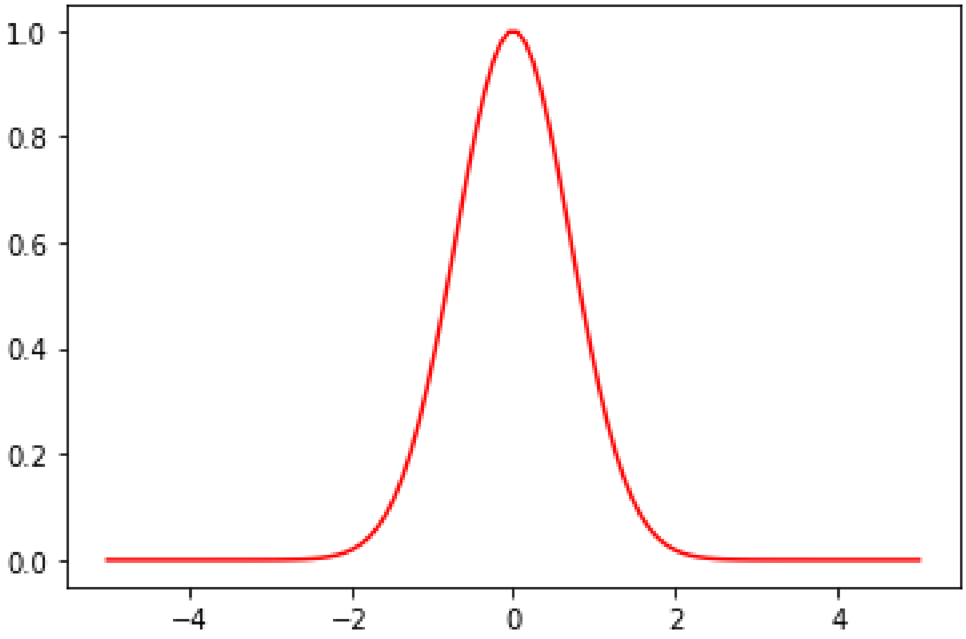
#### 1.2.1 Vectorization

```
[25]: x1 = np.linspace(-5,5,100)
      x2 = np.linspace(0,1,100)

      print(np.exp(3))

      y1 = np.exp(-x1**2)

      y2 = np.sin(10*x2)
```

```
20.085536923187668
```

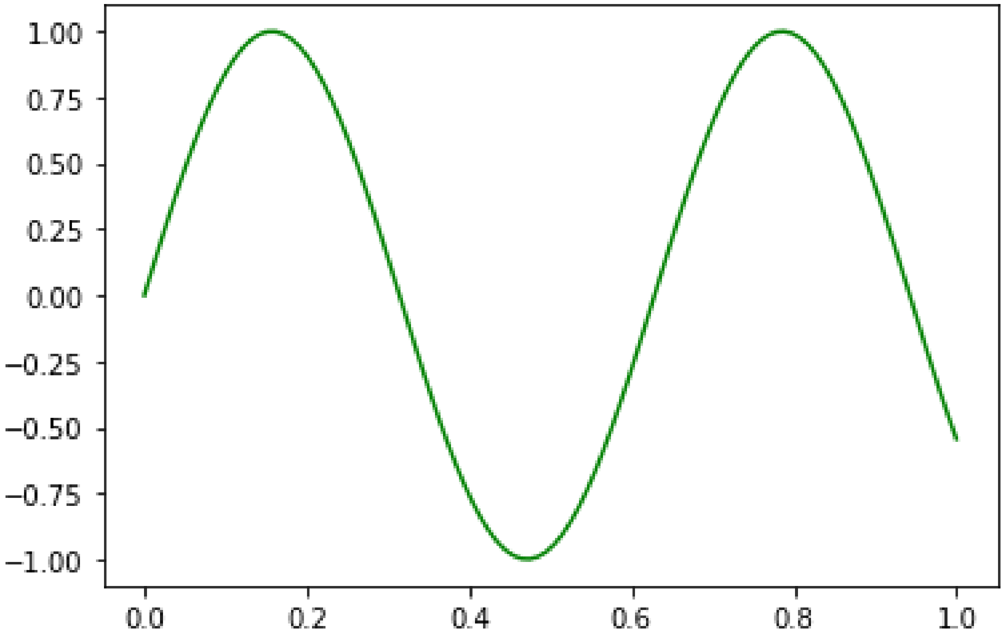
```
[27]: import matplotlib.pyplot as p
      p.plot(x1,y1, color = 'r')
```

```
[27]: [<matplotlib.lines.Line2D at 0x21e9228f280>]
```



```
[26]: plt.plot(x2,y2, color = 'g')
```

```
[26]: [<matplotlib.lines.Line2D at 0x21e92227250>]
```



**Lambda Function** A lambda function is a small anonymous function. It can take any number of arguments, but can have only one expression.

Syntax: lambda arguments : expression

[38]:
```
x = lambda a : a + 10
x(9)
```

```
[38]: 19
```

[40]:
```
y = lambda a,b,c : a*b*c
y(1,2,3)
```

```
[40]: 6
```

Exercise 1 using lambda function

[53]:
```
a = np.array([2**i for i in range(100)], str)
b = np.vectorize(lambda s: s[-1])(a) == '6'
a[b]
```

```
[53]: array(['16', '256', '4096', '65536', '1048576', '16777216', '268435456',
              '4294967296', '68719476736', '1099511627776', '17592186044416',
              '281474976710656', '4503599627370496', '72057594037927936',
              '1152921504606846976', '18446744073709551616',
              '295147905179352825856', '4722366482869645213696',
              '75557863725914323419136', '1208925819614629174706176',
              '19342813113834066795298816', '309485009821345068724781056',
              '4951760157141521099596496896', '79228162514264337593543950336'],
             dtype='<U30')
```

[57]:
```
a = np.array(['ram', 'syam', 'sita', 'gita', 'sagar', 'sima'])
b = np.vectorize(lambda s: s[0])(a) == 's'
a[b]
```

```
[57]: array(['syam', 'sita', 'sagar', 'sima'], dtype='<U5')
```