Python in 4 sittings: Sitting 3

Introduction to Python

Summary of the third class

26th June, 2021

feel free to contact us;

arabindo@protonmail.com

kaustavbasu97@gmail.com

We can run bash command here, in the Jupyter notebook followed by a !

for example to list the files and dir in the current path one need to run the `!ls`

In [1]:
```
!ls
```

```
 03_Numpy_Notebook.ipynb          problem_set1.ipynb
 files                            problem_set2.ipynb
'Introduction to Pandas.ipynb'    session1_fresh.ipynb
 locker.py                        session2_fresh.ipynb
 pass.py                          session3_fresh.ipynb
 photo_2021-06-13_11-59-05.jpg    session-sheet.ipynb
 private.ipynb                    test.py
```

To access files from from python, one can place the file in current directrory. The problem arises when the file is in another folder in that directory. For that one can use `os` library

In [2]:
```
import os
current = os.getcwd()
print(current)
```

```
/home/arabindo/Desktop/Pi4Py
```

In [3]:
```
# To join the paths
# 'demo.txt' is contained in 'files' folder.
# we want the path of that file.
# the essence of this method, it is independent of any OS.
# The path addressing schemes are in general os dependent
path = os.path.join(current, 'files', 'demo.txt')
print(path)
```

```
/home/arabindo/Desktop/Pi4Py/files/demo.txt
```

In the session2 notebook, we counted words from some

text.(in[5] of session2_fresh.ipynb file). demo.txt file contain that same text. Now we will read that file with python and count the words again

In [4]:

```
import pprint
countW = dict()
# with command acts like a loop
with open(path) as f:
    # we have read the file in 'f' variable
    line = f.readline()
    words = line.split()
    for word in words:
        countW.setdefault(word, 0)
        countW[word] += 1

pprint.pprint(countW)
```

```
{'Jupyter': 1,
 'JupyterLab': 3,
 'a': 2,
 'add': 1,
 'and': 5,
 'arrange': 1,
 'code,': 1,
 'components': 1,
 'computing,': 1,
 'configure': 1,
 'data': 1,
 'data.': 1,
 'development': 1,
 'environment': 1,
 'existing': 1,
 'extensible': 1,
 'flexible:': 1,
 'for': 1,
 'in': 1,
 'integrate': 1,
 'interactive': 1,
 'interface': 1,
 'is': 3,
 'learning.': 1,
 'machine': 1,
 'modular:': 1,
 'new': 1,
 'notebooks,': 1,
 'of': 1,
 'ones.': 1,
 'plugins': 1,
 'range': 1,
 'science,': 1,
 'scientific': 1,
 'support': 1,
 'that': 1,
 'the': 1,
 'to': 1,
 'user': 1,
 'web-based': 1,
 'wide': 1,
 'with': 1,
 'workflows': 1,
 'write': 1}
```

we've used f.readline() method. Try to run f.readlines() indepedently. Observe the output.

Let's make a simple password locker.

The original idea is taken from

Automate the boring stuff with Python - Al Sweigart

In [5]:
```
passw = {'email':',sgsdjk',
            'facebook':'hgjk',
            'google':'dghkjgh'}
ask = input('enter the account name').lower()
if ask in passw:
    print(passw[ask])
```

```
enter the account nameFaceBOOK
hgjk
```

Checkout the locker.py file for command line argument approach

Introduction to NumPy

In [6]:
```
import numpy as np
```

In [7]:
```
arr = np.array([2,3,5])
type(arr)
```

Out[7]:
```
numpy.ndarray
```

In [8]:
```
arr1 = np.array(([1,2,3,4],[2,3,5,8],[8,9,1,0]), dtype='float64')
# dtypes argument is optional
# observe the uses of parenthesis carefully
print(arr1)
```

```
[[1. 2. 3. 4.]
 [2. 3. 5. 8.]
 [8. 9. 1. 0.]]
```

In [9]:
```
# function of a matrix is easy with NumPy
exp = np.cos(arr1)
print(exp)
```

```
[[ 0.54030231 -0.41614684 -0.9899925  -0.65364362]
 [-0.41614684 -0.9899925   0.28366219 -0.14550003]
 [-0.14550003 -0.91113026  0.54030231  1.        ]]
```

In [10]:
```
# Let's have an ordinary list same as of arr1
ol = [[1,2,3,4],[2,3,5,8],[8,9,1,0]]
print(ol)
```

```
[[1, 2, 3, 4], [2, 3, 5, 8], [8, 9, 1, 0]]
```

In [11]:
```
# method of accessing the elements are different

# for the ordinary python list
print(ol[0][2])

#for a numpy array
print(arr1[0,2])
```

```
3
3.0
```

In [12]:
```
# slicie a numpy array
# and store it in another variable

arr2 = arr1[0:2,1:3]
print(arr2)
```

```
[[2. 3.]
 [3. 5.]]
```

In [13]:
```
# now change the arr2[0,0] element

arr2[0,0] = 25
print(arr2)
```

```
[[25.  3.]
 [ 3.  5.]]
```

In [14]:
```
# As expected, but it has also changed the original array.
# So be careful with such operation
print(arr1)
```

```
[[ 1. 25.  3.  4.]
 [ 2.  3.  5.  8.]
 [ 8.  9.  1.  0.]]
```

In [15]:
```
s1 = np.sum(arr1, axis=1) # row wise operation
s2 = np.sum(arr1, axis=0) # column wise operation
```

In [16]:
```
print(f's1={s1}, s2={s2}') # formatted output
```

```
s1=[33. 18. 18.], s2=[11. 37.  9. 12.]
```

In [17]:
```
# We can have a matrix with
# random element
# of required shape

a = np.random.randn(3,6)
print(a)
```

```
[[ 0.27778798  0.49710936  1.00295126 -0.07110269 -0.6036254  -1.36212597]
 [-0.10335781 -0.45032648  1.1410223  -0.83908587  0.90473737 -0.42764684]
 [ 1.0987811  -0.42841012  1.42451354  0.81495492 -0.88917787  0.59471125]]
```

In [18]:
```
# we can do set operations with 1d array as well

arr1 = np.array([2,3,5])
arr2 = np.array([2,6,8])
print(np.union1d(arr1, arr2))
print(np.intersect1d(arr1, arr2))
print(np.setdiff1d(arr1, arr2))
```

```
[2 3 5 6 8]
[2]
[3 5]
```

In [19]:
```
# we can create a list of required ranged
# for equally spaced data points

# there will be 100 equal space in between 0 and 100
x = np.linspace(0,10,100)
print(x)
```

```
[ 0.         0.1010101  0.2020202  0.3030303  0.4040404  0.50505051
  0.60606061 0.70707071 0.80808081 0.90909091 1.01010101 1.11111111
  1.21212121 1.31313131 1.41414141 1.51515152 1.61616162 1.71717172
  1.81818182 1.91919192 2.02020202 2.12121212 2.22222222 2.32323232
  2.42424242 2.52525253 2.62626263 2.72727273 2.82828283 2.92929293
  3.03030303 3.13131313 3.23232323 3.33333333 3.43434343 3.53535354
  3.63636364 3.73737374 3.83838384 3.93939394 4.04040404 4.14141414
  4.24242424 4.34343434 4.44444444 4.54545455 4.64646465 4.74747475
  4.84848485 4.94949495 5.05050505 5.15151515 5.25252525 5.35353535
  5.45454545 5.55555556 5.65656566 5.75757576 5.85858586 5.95959596
  6.06060606 6.16161616 6.26262626 6.36363636 6.46464646 6.56565657
  6.66666667 6.76767677 6.86868687 6.96969697 7.07070707 7.17171717
  7.27272727 7.37373737 7.47474747 7.57575758 7.67676768 7.77777778
  7.87878788 7.97979798 8.08080808 8.18181818 8.28282828 8.38383838
  8.48484848 8.58585859 8.68686869 8.78787879 8.88888889 8.98989899
  9.09090909 9.19191919 9.29292929 9.39393939 9.49494949 9.5959596
  9.6969697  9.7979798  9.8989899 10.        ]
```

In [20]:
```
# We can do filtering with NumPy

# Let's have sine of x
y = np.sin(x)
# Now we are interested in those datapoints in y
# where the value of y is greater than 0.5
# for that we will create a boolean filter
filter = y>0.5
print(filter)
```

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

In [21]:
```
# Now if we can 'mask' the filter over y

z = y[filter]
print(z)
```

```
[0.56963411 0.64960951 0.72296256 0.78894546 0.84688556 0.8961922
 0.93636273 0.96698762 0.98775469 0.99845223 0.99897117 0.98930624
 0.96955595 0.93992165 0.90070545 0.85230712 0.79522006 0.73002623
 0.65739025 0.57805259 0.55261747 0.63384295 0.7086068  0.77614685
 0.83577457 0.8868821  0.92894843 0.96154471 0.98433866 0.99709789
 0.99969234 0.99209556 0.97438499 0.94674118 0.90944594 0.86287948
 0.8075165  0.74392141 0.6727425  0.59470541 0.51060568]
```

We have shown Pandas just to read csv and excel files

I'm showing here for csv files. The method is similar for xls files.

You must be careful with xlsx files(earlier excel file extension)

In [22]:
```
#### I've a file 'test.csv' in the 'files' folder

cwd = os.getcwd()
path = os.path.join(cwd, 'files', 'test.csv')
path
```

Out[22]:
```
'/home/arabindo/Desktop/Pi4Py/files/test.csv'
```

In [23]:
```
import pandas as pd
var = pd.read_csv(path)
```

In [24]:
```
# to know the coumn name
print(var.columns)
```

```
Index(['red', 'green', 'norm'], dtype='object')
```

In [25]:
```
# I can store different columns
# in some list

# Accessing data frames are similar to
# the dictionary
x = var['red']
y = var['green']
```

In [26]:
```
# what is the type of var?
# it is data frame
print(type(var))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [27]:
```
# We can create a dataframe from a dictionary
data = {'1':pd.Series([1,2,3], index=['apple','oranges','breads']),
       '2': pd.Series([5,8,6], index=['one','two', 'three'])}

df = pd.DataFrame(data)
# notice, D and F is Capital
df
```

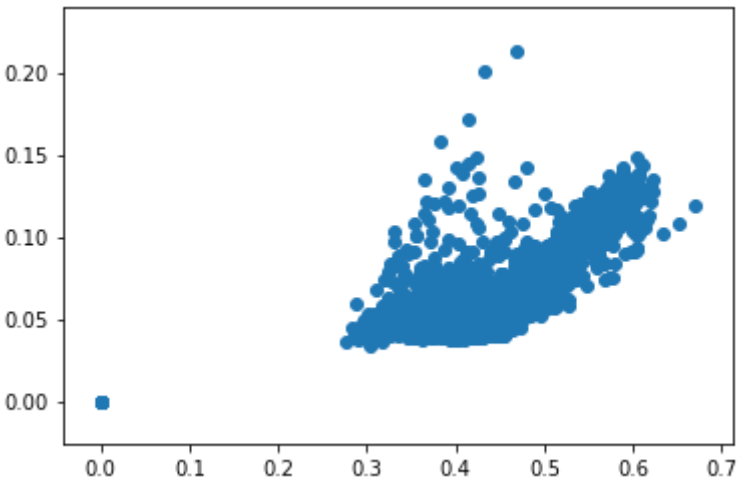
Out[27]:

	1	2
apple	1.0	NaN
breads	3.0	NaN
one	NaN	5.0
oranges	2.0	NaN
three	NaN	6.0
two	NaN	8.0

Before we close, we will do some matplotlib stuff

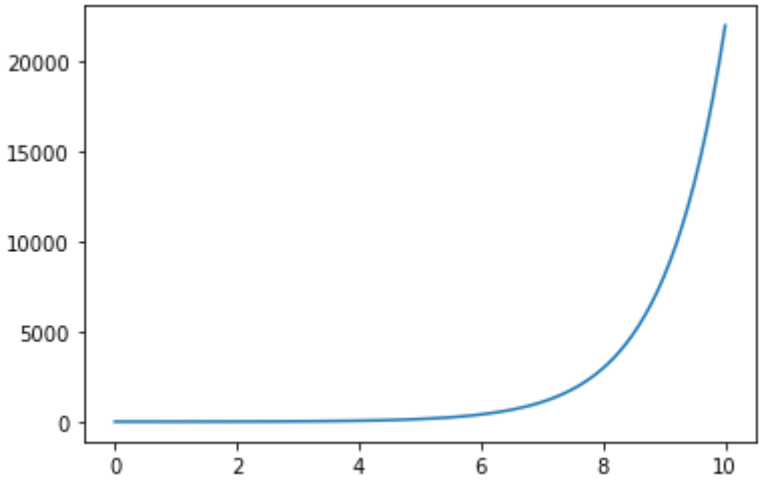
In [28]:
```
import matplotlib.pyplot as plt
%matplotlib inline
# last line for notebook only
```

In [29]:
```
# In the line In[25], we loaded some data
# in the variable x and y.
# Let's plot them as a scatter points

plt.scatter(x,y)
plt.show()
```



In [30]:

```
# We can plot functions or
# data listwith smooth curves

# Let's plot exponential of x
x = np.linspace(0,10,100)
y = np.exp(x)
plt.plot(x,y)
plt.show()
```



We can set label of the plots,

ranges for x-axis and y-axis,

can set title,

and many more.

Just google it.

Or visit Official documentation

https://matplotlib.org/stable/contents.html

Not need to read everything.

Just learn what is needed!

Otherwise it is just huge!

The only way to learn a library efficiently, is to read official documentation

So I'm leaving those as a reading project for you!

In [31]:
```
# Let's try to simulate a superposition
# The code is originally from
# a book
# Unfornutaely, I donot know the name.
# I've only PDF of chapter 3
# Named, Graphics and Visulaisation.
# Search it on Google please! :")
```

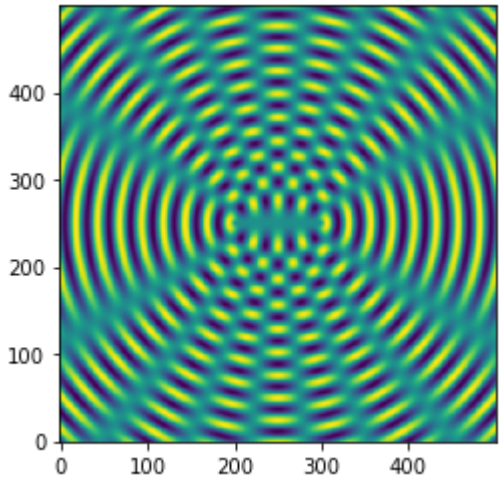
In [32]:
```
# what we will need?

lam = 5.0 #wavelength
k = 2*np.pi/lam #wave number
x0 = 1 #amplitude
sep = 20.0 #separation between two center
side = 100 # image size
pts = 500 # number of data points
spacing = side/pts #spacing between data points
x1 = side/2 + sep/2 #defining center of the waves
y1 = side/2
x2 = side/2 - sep/2
y2 = side/2
data = np.zeros([pts, pts], dtype='float64') # initialize the storage
```

In [33]:
```
for i in range(pts):
    # span either x or y
    y = spacing*i
    for j in range(pts):
        # span another - y or x
        x = spacing*j
        r1 = np.sqrt((x-x1)**2+(y-y1)**2)
        r2 = np.sqrt((x-x2)**2 + (y-y2)**2)
        data[i,j] = x0*np.sin(k*r1) + x0*np.sin(k*r2)
# Now what's in data?
# Amplitudes of the wave at i,j points
# you can check that too by printing it
print(data)
```

```
[[-1.62502554 -1.51544577 -1.36219785 ... -1.17147895 -1.36219785
  -1.51544577]
 [-1.54933007 -1.3923215  -1.19639492 ... -0.96894244 -1.19639492
  -1.3923215 ]
 [-1.42165878 -1.22099523 -0.98749995 ... -0.72957494 -0.98749995
  -1.22099523]
 ...
 [-1.24541089 -1.00635806 -0.74173828 ... -0.46074226 -0.74173828
  -1.00635806]
 [-1.42165878 -1.22099523 -0.98749995 ... -0.72957494 -0.98749995
  -1.22099523]
 [-1.54933007 -1.3923215  -1.19639492 ... -0.96894244 -1.19639492
  -1.3923215 ]]
```

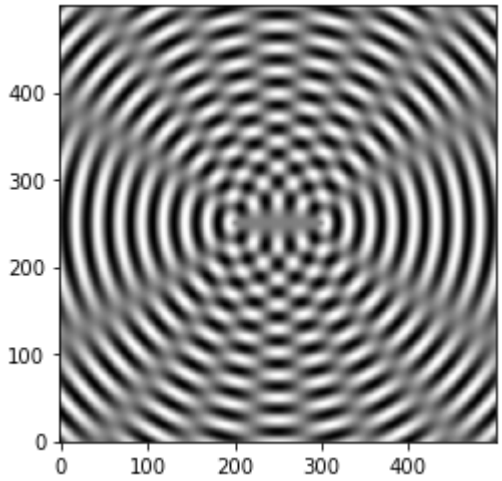
In [34]:

```
# plt.plot is of no use here
# we will use imshow (image show)
plt.imshow(data, origin='below')
# Try without origin argument, spot the difference
# I've explained in the session though.
plt.show()
```



In [35]:

```
# Colours signifie the amplitude value
# you can plot the gray profile
plt.imshow(data, origin='below')
plt.gray()
plt.show()
```



Here you can also have some label for the colours and amplitude relation. Again, you know :p

Google it!

That's it for today! Proble set will be live soon!