In [ ]:

```
#Python List Vs Tuple:
#we know list is mutable and tuple is immutable .But List within a tuple can be updated.
a=[1,2,(1,2)] #list
b=(1,2,[1,2]) #tuple
a[2].append(3) #Gives Error:'tuple' object has no attribute 'append'
b[2].append(3)
print(b) # Gives output:(1, 2, [1, 2, 3])
b[2]=100 # Gives an Error:'tuple' object does not support item assignment.

#Conclusion: Dont go always with mutability or immutabilty. Fact is tuples does not support
```

In [ ]:

```
#Python Soft Copy Vs Deep Copy
###<<<Soft Copy>>>
a=[1,2]
b=a
b[0]=-1
print(a) #Gives output:[-1,2]
#==============================
###<<<Deep Copy>>>
import copy
a=[1,2]
b=copy.deepcopy(a)
b[0]=-1
print(a) #Gives output:[1,2]

#Conclusion:Normal assignment operation is soft copy.Soft copy copys the reference/link.Hen
```

In [ ]:

```
#Python Append vs Extend
a=[1,2]
b=[3,4]
#a.append(b)
print(a) #Gives output:[1, 2, [3, 4]]
a.extend(b)
print(a) #Gives output:[1, 2, 3, 4]
```

In [ ]:

```
#Higher order function:A fuction which takes another function as a parameter .map,filter,re
#In python3, rather than returning a list; filter, map return an iterable.([i for i in flis
#map vs reduce: both having two parameters.map function's first parameter returns value but

def f(x): return x % 2 == 0
def m(y): return y * 2
def n(x,y): return x+y

l = [1,2,3,4]

flist = filter(f, l)
print(l)
print(flist)

mlist = map(m, l)
print(l)
print(mlist)

from functools import reduce
nlist=reduce(n,l)
print(l)
print(nlist)
```

In [ ]:

```
#Custom/User defined higher order function
def square(num):
    return num * num

def cube(square,n):
    return square(n)*n

print(cube(square,5)) #Gives output:125
```

In [ ]:

```
#Is Python functional programming?
#yes,Python functional as well as object oriented programming
#As it passes function in a function as parameter and stores lamda function in a variable a
g = lambda x: x*x*x
print(g(7))
```

In [ ]:

```
#Pickling vs unpickling(Serilization vs Deserilization)
#Pickling - is the process whereby a Python object hierarchy is converted into a byte strea
#Unpickling - is the inverse operation, whereby a byte stream is converted back into an obj
#Why pickling:Pickling is just serialization: putting data into a form that can be stored i
```

In [ ]:

```
#Sort vs Sorted function in Python
#sorted() returns a new sorted list, leaving the original list unaffected. list.sort() sort
#Sort is faster than sorted .As it does not copy .Sort does sorting on original object
```

Why Numpy array is faster than List?

#Numpy arrays are stored in a single contiguous (continuous) block of memory.List does not import numpy as np import sys l1 = [i for i in range(1000000)] l2=np.array(l1) print(id(l1[999])==id(l1[9999]))#Gives output:False print(id(l2[999])==id(l2[999999])) #Gives output:True print(sys.getsizeof(l1))#Gives output:8697464 print(sys.getsizeof(l2))#Gives output:4000096

In [ ]:

```
#Get first n letter of a string from column in a pandas df
df['new_col'] = df['First'].astype(str).str[0:n]
```

In [ ]:

```
#How to Get Part of a Column Names in Pandas Data Frame?
df.columns.str.split('_').str[0]
```

In [ ]:

```
#Time complexity of "in",max(),len()
#in,max() :O(n)
#len is an O(1) because in your RAM, lists are stored as tables (series of contiguous addre
```

In [4]:

```
#How to write my own split function?
a="Tapas Pall Tapas TPal TapP al Pala" #string
s="Tapas" #pattern
def fun(a,s):
  st=""
  l=len(s)
  li=[]
  lii=[]
  for i in range(0,len(a)):
    if a[i:i+l]!=s:
        st=st+a[i]
    elif i+l>len(a):
        st=st+a[i]
    else:
        li.append(st)
        i=i+l
        st=""
  li.append(st)
  lii.append(li[0])
  for i in li[1:]:
      lii.append(i[l-1:])
  return lii
print(fun(a,s))
print(a.split(s))
```

```
['', ' Pall ', ' TPal TapP al Pala']
['', ' Pall ', ' TPal TapP al Pala']
```

In [5]:

```
#Sort a dictonary in Python
d={1:100,2:102,3:103}
sorted_d = sorted(d.items(), key=lambda x: x[1]) #x[0] sort by key and x[1] sort by value
print(sorted_d)
```

```
[(1, 100), (2, 102), (3, 103)]
```

In [ ]:

```
#Generator vs Iterator ?
#Yield ?
```

In [2]:

```
#expolde list of a column into rows in a df .It works on or above 0.25 pandas version
import pandas as pd
df = pd.DataFrame([{'var1': 'a,b,c', 'var2': 1},
             {'var1': 'd,e,f', 'var2': 2}])
df.assign(var1=df.var1.str.split(',')).explode('var1')
#==================================
#column splitted into multiple column
import pandas as pd

d1 = {'teams': [['SF', 'NYG'],['SF', 'NYG'],['SF', 'NYG'],
                ['SF', 'NYG'],['SF', 'NYG'],['SF', 'NYG'],['SF', 'NYG']]}
df2 = pd.DataFrame(d1)
print (df2)
df2[['team1','team2']] = pd.DataFrame(df2.teams.values.tolist(), index= df2.index)
print (df2)
```

```
       teams
0  [SF, NYG]
1  [SF, NYG]
2  [SF, NYG]
3  [SF, NYG]
4  [SF, NYG]
5  [SF, NYG]
6  [SF, NYG]
       teams team1 team2
0  [SF, NYG]    SF   NYG
1  [SF, NYG]    SF   NYG
2  [SF, NYG]    SF   NYG
3  [SF, NYG]    SF   NYG
4  [SF, NYG]    SF   NYG
5  [SF, NYG]    SF   NYG
6  [SF, NYG]    SF   NYG
```

In [ ]:

```
#read a text file(two columns only) a put the value into dictanary
res = {}
with open("test.txt", "r") as f:
  # Read file skipping the header
  for line in f.readlines()[1:]:
    name, value = line.strip().split()
    if name not in res:
      res[name] = int(value)
      continue
    res[name] += int(value)
print(res)
```

In [ ]:

```
#Unzip zip files in folders and subfolders with python
import zipfile,fnmatch,os

rootPath = r"C:\Project"
pattern = '*.zip'
for root, dirs, files in os.walk(rootPath):
    for filename in fnmatch.filter(files, pattern):
        print(os.path.join(root, filename))
        zipfile.ZipFile(os.path.join(root, filename)).extractall(os.path.join(root, os.path
```

In [ ]:

```
#why cyclic import does not work in Python?
```

In [8]:

```
#Split using vectorization in Pandas:
import pandas as pd
# create a new data frame
df = pd.DataFrame({'Name': ['Steve Smith', 'Joe Nadal',
                            'Roger Federer'],
                   'Age':[32, 34, 36]})
print(df)
df.Name.str.split(expand=True,)
df[['First','Last']] = df.Name.str.split(" ",expand=True,)
print(df)
#replace using vectorization:
df["Name"] = df.Name.str.replace(" ",",").astype(str)
print(df)
#Slicing using vectorization:
df["Name_slice"]=df["Name"].str[0:4]
print(df)
```

```
   Age           Name
0   32    Steve Smith
1   34      Joe Nadal
2   36  Roger Federer
   Age           Name  First     Last
0   32    Steve Smith  Steve    Smith
1   34      Joe Nadal    Joe    Nadal
2   36  Roger Federer  Roger  Federer
   Age           Name  First     Last
0   32    Steve,Smith  Steve    Smith
1   34      Joe,Nadal    Joe    Nadal
2   36  Roger,Federer  Roger  Federer
   Age           Name  First     Last Name_slice
0   32    Steve,Smith  Steve    Smith       Stev
1   34      Joe,Nadal    Joe    Nadal       Joe,
2   36  Roger,Federer  Roger  Federer       Roge
```

In [ ]:

```
# Aggregation on multiple columns using Group by
animals.groupby("kind").agg(
   ...:     min_height=('height', 'min'),
   ...:     max_height=('height', 'max'),
   ...:     average_weight=('weight', np.mean),
   ...: )
```