Python

# Python Lists:

Data = [‘a’,’b’,’c’,’d’,’e’]

Names =[‘raju’,’Praveen’,’pandu’,’Buchi’,’siva’]

nxt = [‘m’,’v’,’b’]

1)Append

Data.Append(‘f’)

o/p : [‘a’,’b’,’c’,’d’,’e’,’f’]

Data.Append([‘g’,’h’])

O/P : Data = ['a', 'b', 'c', 'd', 'e', ['f', 'g']]

Multiples append should be in braces(),[] ..output appears like that only.

2) Extend

Data.extend(nxt)

O/P: ['a', 'b', 'c', 'd', 'e', ’m’,’v’,’b’]

3) remove

Data.remove(‘e’)

O/P : ['a', 'b', 'c', 'd']

4) pop : Index based data removing

Data.pop(1)

O/P : ['a', 'c', 'd',’e’]

5) Index : Retrieve Position of Index (elem,Starting Index)

Data.Index(‘d’) O/P : 3

6) Copy : copy the list Alternative Method : new list =old list (old is assigned to new)

Y= []

Y = data.copy() listname.copy

print (y)

O/P : [‘a’,’b’,’c’,’d’,’e’]

7) Insert:

Data.insert(2,’r’) # insert(Index,elem)

O/P : [‘a’,’b’,’r’,’d’,’e’]

8) Sort :

Data.sort() # Sorts in ascending order

sorted(Data, key=len)

The list is sorted based on the length of its each element, from lowest count to highest.

Reverse =TRUE indicates Desc

O/P : : [‘a’,’b,’d’,’e’,’r’]

9) Count : # Counts letters in word or no of times element repeated

Data.count(‘a’)

Names.count(‘mvb’)

O/P :1

O/P : 1

10) Reverse :

Data.reverse()

O/P [‘e’,’d’,’c’,’b’,’a’]

11) Clear:

Data.clear()

o/p : []

Reference Links : <https://www.programiz.com/python-programming/methods/list/copy>

Data Quest 3

Conversion of tuple to List :

X= (1,’a’,6.5,1)

We can convert into tuple x into list

list(X)

O/P : x= [1,’a’,6.5,1]

Conversion of set to list

s= {1,’a’,6.5,1}

list (s) # Removes duplicate items

O/P : s= [1,’a’,6.5]

|  |  |  |  |
| --- | --- | --- | --- |
| Tuple | List | Dictionary | Set |
| X=(1,’a’,4.5,1) | x=[1,’a’,4.5,1] | X={‘a’:1,’b’:2,’c’:4.5} | X= {1,’a’,6.5,1} |
| List(x)  O/P: [1,’a’,4.5,1] |  | X={**'a'**:1,**'b'**:2,**'c'**:4.5} lst =[] **for** i **in** X:  lst.append(i)  lst.append(X[i]) print(lst)  O/P :[‘a’,1,’b’,2,’c’,4.5] | List(x)  O/P: [1,’a’,6.5]  Removes duplictes |
| Immutable |  | Items,keys,values |  |
|  |  |  |  |

Ref: <https://www.youtube.com/watch?v=R-HLU9Fl5ug>

**Note : String.replace()**

To replace value in a string str =”hello world” str.replace(”hello”,”world”) 🡪world world

Note : when you use new module better to know about directory of module

pantry = ["apple", "orange", "grape", "apple", "orange", "apple", "tomato", "potato", "grape"]

pantry\_counts = {}

for i in pantry:

if i in pantry\_counts:

pantry\_counts[i] = pantry\_counts[i] +1

else :

pantry\_counts[i] = 1

* Import csv

Reader = csv.reader(path)

Header = next(Reader)

* With file opening used to close the file explicitly

Anaconda :

Cells can be operated in two modes : 1) Edit Mode (Green Color) 2)Command Mode(Blue Color)

Presss Esc enter into Command mode

We have To options as code and markdown

code is for normal text and markdown is for Header or Description kind of stuff

Cmds : A for insert cell above , y for code \ m for mark down

Key Board Shortcuts cmd (mode): h key for help of all commands

X for cut(delete) cells

B for insert below cells

A for insert above cells

## PANDAS:

Series : gives the list information as a column in **1-Dimensional**

Import pandas as pd

Lst = [“a”,”b”,”c”]

Pd.Series(Lst)

Attributes : like select only which don’t perform action

Methods : Perform some actions

**Note** : shift +tab to get documentary….of method

Tab for options displaying

**INTERPOLATE:**

Interpolate is a series method to fill nan values

Series.interpolate(method=”Linear”)

df['FlightNumber'] = df['FlightNumber'].interpolate(method="nearest").astype(int)-🡪 Assigns Nan to nearest values

df['FlightNumber'] = df['FlightNumber'].interpolate(method="linear").astype(int)🡪having same difference

# Importing Files to Pandas:

While importing files from local put “r” to remove carriage return value (if not uni code error may come)

import pandas as pd

df=pd.read\_csv(r'C:\\Users\\buchi\\Desktop\\emp.csv') (should be in single Quotation [‘ ‘])

df

You can also use the [alias](https://en.wikipedia.org/wiki/ISO/IEC_8859-1) 'latin1' instead of 'ISO-8859-1'. Encoding parameters

**To use specific columns : ‪**

df=pd.read\_csv(r'C:\\Users\\buchi\\Desktop\\emp.csv',usecols= [“emp\_id”,”ename”],squeeze =True)

**usecols** should be in List type

Squeeze gives us series representation instead of Data Frame representation(Table Repre)

**Built in functions in Python :**

**Len -**🡪 Gives length

Type🡪 Gives data type

Dir(Data set) -🡪Gives Available methods and attributes-🡪 dir(pd)

Sorted🡪Gives sorted input

Dict and list , Max and min;

Emp.shape 🡪 gives no .of rows and columns

Sort Values Method :

emp.sort\_values()----🡪 Gives brand new series with alphabhetal order--------for series don’t confuse

Sort\_values(by = column (it should be in quotes)---by=[col1,col2],ascending=[False,True]

Emp .Sort\_Index() --- sort based on index helps in retrieval fast—Best Practice

Inplace = True means overwrite

Extracting information from index emp[0:10] (instead of emp[0,1] or emp[2])

**IN key word**

“smith” in emp -🡪 default it looks in index label not on values

10 in emp 🡪True (10 in emp.index)

Emp.values---gives all values as list of lists

“smith” in emp.values 🡪 gives correct result

Math Methods on Series:

Count gives no of values in a data set (it excludes null values)

Len gives no of values in a data set (it includes null (NaN) values)

Emp.describe()-🡪 gives all method

Apply method :

Create a function and give that function to act as input to .apply() method --apply on single column

Apply function to all values in a dataset

Map method takes values from 1st source lookup index in second source and gives corresponding values in second source(we can use for dict)

1. Values from first series look into index from second series and gives result



**Data Frame 1:**

Shared Methods between Series and data Frame : head,tail

Shared Attributes : index,values(football.values -🡪 list of list values) ,dtypes-----df.dtypes

Attributes which are exclusively in Data Frame: columns,axes(football.axes-🡪index and column names)

Exclusive Data Frame methods: info()🡪 all available details DF

Sum method in data frames gives vertical count default (axis=0) and we can get horizontal count (axis=1)

Axis =1-🡪index based,axis=0🡪row based(T2B)

Add columns to a data frame : dataframe.insert(loc,columnname,value)🡪(3,”leaue”,”FIFA”)

value\_counts()-🡪gives no of unique values in a series

dropna-🡪removes rows who has NAN values(how= ‘any’(default anywhere) “all” applies when all values are NAN

dropna(subset=[“columns”,”columns”] -🡪 which removes row where it has NAN values in given columns

fillna method to fill NAN values better use this method with individual column

astype()🡪converts float to integer and reduce memory by converting to category

rank

unique () 🡪 to get distinct values and nunique() method for number representation

df.duplicated(“column\_name”,keep=”first)---gives the resultant set of all duplicate values (which repeated more than once) from 2nd element ---it won’t think 1st element as duplicate

~(negation of) df.duplicated(“column\_name”,keep=”first)---gives all unique elements in a data set

ex: mask[~mask.duplicated("Customer ID",keep="first")]

keep = False ---performs operation only on duplicates(gives duplicated resultant set) if any unique rows it won’t give

**Query method**:

max\_item\_price =max(chipo.item\_price)

chipo.query('item\_price>44')["item\_name"]

**Memory Optimization :** Conversion of data types to required

**Filters :** Dataframe[dataframe[“column”] (condition]

sales[sales["Order Priority"] == "Low"] ----🡪Filter based on Order Priority Gives Data Set

we can use two filter conditions (or | and &) when you are giving 3 conditions make sure in braces

i.e (A&B)|c

**Isin**-----🡪 To check the values in columns,

Between method can be used with dates

**DEALING** NULL VALUES IN Pandas:

1. Finding null values in a dataset (anywhere in a column)

Dataframe.isnull().any(axis=1) ----🡪 gives all values in a dataset has any Nan values in it.

football[football.isnull().any(axis=1)]

1. Finding null values in a dataset (all values in a column should be null)

Dataframe.isnull().all(axis=1) ----🡪 gives all values in a dataset has any Nan values in it.

football[football.isnull().all(axis=1)]

1. Count of null values by column wise

Dataframe.isnull().sum()

To count null values first sum gives column wise count and second sum gives total count

football.isnull().sum().sum()

1. Using len and count methods we can get length and count respectively
2. Using dropna() and notnull() method we can ignore all null values

football.dropna(how="any")

football.dropna(how="all")

football.notnull()

1. Check null values by column wise

Football[“Name”].isnull()

7)Filling all null values in a data set using fillna()

Football.fillna(value)

df.fillna(value,method,limit)

df.fillna(dataframe.mean()[col2:col3])

method=pad,ffill-🡪 gives previous value to NAN

bfill,backfill-🡪 gives after value to NAN

Use ~ for not operation

**Note**: If any NAN values present in column int value appear as float . remove NAN to convert to int

**Data Frame III:**

Set\_index()-🡪 set index column

Football.set\_index(“Column name”,inplace = False)

Reset\_index()--🡪 reset index

**Index Based Retrieval :**

Basically we have three 3 methods to retrieve data from data frame based on index position

1)loc

2) iloc

3)ix

**Loc**-🡪 fetches data based on index column it should includes alphabetical in index column

Football.loc[“Neymar”] -------🡪fetches data

Note : “Neymar” in football.index should be true

**Iloc** --🡪 fetches data based on index column ,parameter should be numerical

Football.iloc[1:10]

**Ix:** This method is combination of loc and iloc ,it both arguments alpha and numeric ..vice versa

But based on index column only

football.ix[2,"Nationality"]

football.ix[3,6]

**Update Single value in a data frame :**

We can update value by ix method by assigning value at corresponding position

football.ix[2,["Age","Overall"]] = [26,99]

Update Multiple values :

Ix method takes view as input

Football.ix[view,”column”] =”change name” -------- applies on original series

Football[football[“Name”]] = =“Neymar” --------------- gives brand new series

Rename columns:

Football.rename(columns={column1:”newname”,col2:”newname”},inplace =True)

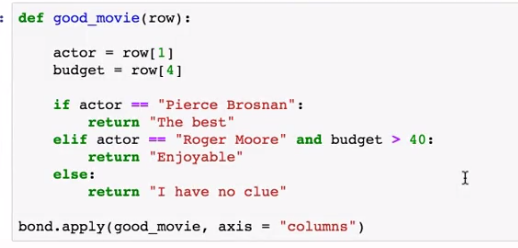
**Drop**:

Football.drop(0,axis=0) and for columns axis =1----defaultly operation applies on rows

**Apply Method on Single Column:**

1. Create a function with return statement
2. Applies on single column in a data frame
3. It should be custom function
4. Def func name(parameters):
5. Logic
6. Return
7. Dataframe[column\_name] .apply(func name)
8. If you want to over write you can by dataframe[column\_name] = Dataframe[column\_name] .apply(func name)

**Apply Method on Rows**: It gives result for Index column



**Working With Text (Strings) Data**:

“Hello world”.upper() 🡪 upper case

“Hello world”.lower()----------🡪lower case

“Hello world”.title()------------🡪First letter Capital 'Hello World'

When we are working on strings in a dataframe always use with .str.

**String Methods :**

1. .str.replace(“old”,”new)
2. .str.split(“ “) or(“,”) -----🡪 it will split based on parameter
3. .str.get(0) ------🡪 element wise access
4. .str.strip(),.str,lstrip(),.str.rstrip()
5. .str.contains(“Name”)-------shows result which contains string
6. .str.startswith(“Name”) -----search for Name starts with
7. .str.endswith(“Name”) ---- search for Name ends with

football["Player\_Name"].str.split(" ").str.get(0).str.strip().value\_counts()

f =football.dropna(axis=0)

f[f["Player\_Name"].str.contains("Messi")]

we can apply string to **indexes** as well as columns

dataframe.index.str.upper()

datframe .columns.str.lower()

.str.split(“ “,expand =True,n=1)

Expand = True gives data frame object

N= no of times wants to split

**GROUP BY:**

Import data set…

Var =dataset.groupby(“Column\_name”) –which we want to perform group by func

Var.size() -------gives grouping information

Var.first()---Fiirst values

Var.last()---Last Values

**Get\_Group:**

Get group to retrieve grouping column values

Var.get\_group(“grouping\_column”)[“Column”]----------There is no dot between grouping column and column

ss.get\_group("Consumer")["Profit"].sum() -----consumer is grouping column value

**Agg**

ss.agg({“Column”:sum,”Column\_name”:mean,”Column”:[size,sum]}) -----mathematical operations in quotes iff

ss.agg([“size”,”sum”,”mean”]) ----which performs operation on every single numerical column

**Merging & Joining & Concatenating:**

Pd.concat([“df1,”df2”])--------------week1,week2 should be in list

To avoid any Index problem ignore.index=True -----when two data frame indexes are mutual

Keys =[“A”&”B]

Df.ix[(“A,20),”Column\_name]---------------20 index-----keys should be in tuple

**INNER JOIN**:--multiple column

df1.merge(df2,how=”join type”,on = “Column\_name”, suffixes=(x,y(replace))

EX: c=week1.merge(week2,how="inner",on="Customer ID")

**OUTER JOIN**:

outer=week1.merge(week2,how="outer",on="Customer ID",indicator=True)

indicator –gives from which dataset resultant came---left,right,both

**LEFT JOIN :**

Similar to above—how=”left”

**ON Parameter with Different column Names(left\_on & right\_on):**

If column names from both dataset are different we need specify by using left\_on and right\_on

Left\_on = column name of left data set

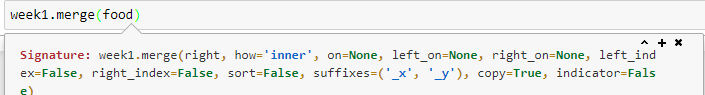
Right\_on= column name of right data set

It gives both columns we can use drop(“column”,axis=1) to remove duplicated

**Merging Based on Index:**

If joining column is in index\_column then use

Left\_index=True



**Join**()—if you want join to data set based on index (shortcut to merge index)

Pd.merge()—directly we can provide dataset

**Date & Time:**

pandas datetime module is inbuilt module

import datetime as dt

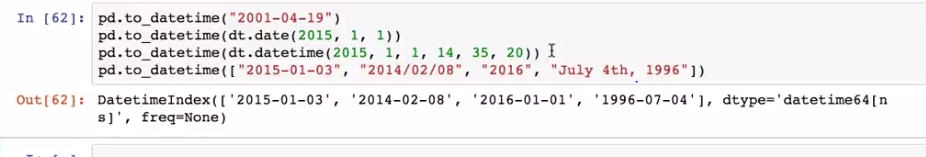
dt.date(2017,04,14)---datetime built in packages input should be sep with ,

str(dt.date(2017,04,14)----gives string representation

from pandas we can perform date operations

import pandas as pd

pd.Timestamp(“2017/04/20”)-🡪 year/month/date



Errors=coerce -----gives Nat

Pd.date\_range(“start date”,”enddate”,freq=)

Freq---D,2D,B(business),W(WeekDay),H(hours),6H (every 6hrs)

M—month,MS –moth start date,A---last date of every year,B for Business Days

**Resample:**  converting or tuning to certain range

Datecolumn.resample(‘M’) ---all are grouped to month

Periods---No of results you want and set freq

Start=”date”,period=20---gives 20 results from start date based on freq

End =”date”,period=20---gives 20 results till end date based on freq

dt. Accesor:

like .str for string .dt for date

series .dt.day or s.dt.month,s.dt.is\_quarter\_day

**DataReader**

from pandas\_datareader import data

data.DataReader(name="MSFT",start="2013-01-01",end="2017-11-21",data\_source="google")

delays = df['RecentDelays'].apply(pd.Series)

delays.columns = ['delay\_{}'.format(n) for n in range(1, len(delays.columns)+1)]

df = df.drop('RecentDelays', axis=1).join(delays)