In [ ]:
```
#lambda expression and functions
```

In [1]:
```
"""
call a regular function.
we define it and then call it
"""


def magic(x):
    return x*90


print( magic(100) )
```

```
9000
```

In [2]:
```
#a function that can accept a list and get applied on all elements in the list


def square(l1):
    for element in l1:
        print(element ** 2)


square([1,2,3,4,5])
```

```
1
4
9
16
25
```

In [ ]:
```
def cube(l1):
    for element in l1:
        print(element ** 3)
```

In [ ]:
```
#lambda  <input parameters>   : <output expression>(something that returns a value)
```

In [3]:
```
#recommended approach?----------------->functional programming


def applyLogic(  f1, l1  ):
    for element in l1: #repeat this operation for all numbers in l1
        print( f1(element)  ) #print result of f1 applied on the given element

applyLogic( lambda x : x**2, [1,2,3,4,5]     )
applyLogic( lambda x: x**3, [1,2,3,4,5] )
```

```
1
4
9
16
25
1
8
27
64
125
```

In [4]:
```
#applying a certain operation on all items in a collection


# map, filter and reduce



#apply a logic to a collection


print(  list( map(  lambda x : x**2  ,   [1,2,3,4,5]))  ) #a list of answers
```

```
[1, 4, 9, 16, 25]
```

In [5]:
```
print( list( map( lambda x : x*0.9 , [10,20,30,40,50] ) ) )#list of answers
```

```
[9.0, 18.0, 27.0, 36.0, 45.0]
```

In [ ]:
```
#take 5 numbers from a user as input ON A SINGLE LINE separated by space

#write a function that can convert the numbers given as input into int type and store them
#in a list
```

In [ ]:
```
"""
focus on WHAT TO DO rather than HOW TO DO IT
"""
```

In [7]:
```
print(list( map( lambda x : int(x) , input("Enter 5 numbers separated by space ").split(" ")) ) )
```

```
Enter 5 numbers separated by space 12 14 -98 -76 21
[12, 14, -98, -76, 21]
```

In [8]:
```
import pandas as pd

df = pd.read_csv("/home/harshit/DataSets/YESBANK.NS.csv")
df
```

Out[8]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-12-11	313.500000	315.799988	310.600006	311.600006	300.880615	4416465.0
1	2017-12-12	312.000000	312.000000	305.899994	306.799988	296.245758	5457103.0
2	2017-12-13	306.350006	307.350006	301.049988	301.899994	291.514282	6911856.0
3	2017-12-14	303.899994	304.649994	301.750000	303.899994	293.445526	4904177.0
4	2017-12-15	307.000000	317.450012	307.000000	315.899994	305.032715	20571225.0
...	...	...	...	...	...	...	...
733	2020-12-02	15.700000	15.900000	14.850000	15.450000	15.450000	311349886.0
734	2020-12-03	15.650000	15.800000	15.250000	15.450000	15.450000	152445535.0
735	2020-12-04	15.600000	15.600000	15.050000	15.350000	15.350000	149691622.0
736	2020-12-07	15.650000	15.850000	15.500000	15.750000	15.750000	193242183.0
737	2020-12-08	16.000000	17.299999	16.000000	17.299999	17.299999	562741066.0

738 rows × 7 columns

In [9]:
```
#apply(this works like the map function from basic python)

#apply a function on every value in a column in pandas

df['Open'].apply(  lambda x : x*0.5  )
```

Out[9]:
```
0      156.750000
1      156.000000
2      153.175003
3      151.949997
4      153.500000
          ...
733      7.850000
734      7.825000
735      7.800000
736      7.825000
737      8.000000
Name: Open, Length: 738, dtype: float64
```

In [10]:
```
#

df1=pd.read_csv("/home/harshit/DataSets/Titanic.csv")
df1
```

Out[10]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [24]:
```
df1.head(2)
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

In [28]:
```
#50 percent discount for female passengers and 20 percent for male passengers


#a single row data frame
def calculateDiscount(df):
    if df['Gender'] == 'male':
        return 0.2 * df['Fare']

    else:
        return 0.5 * df['Fare']

#take gender and fare from EACH ROW ONE BY ONE, apply calculateDiscount on that row
df1[ ['Gender','Fare'] ].apply(  calculateDiscount, axis=1 )
```

Out[28]:
```
0       1.45000
1      35.64165
2       3.96250
3      26.55000
4       1.61000
         ...
886     2.60000
887    15.00000
888    11.72500
889     6.00000
890     1.55000
Length: 891, dtype: float64
```

In [33]:
```
df1.drop('Discount',axis=1,inplace=True)
```

In [34]:
```
#one line if -else expression is:     <if clause part > <if condition> else <else clause part>


"""
take an input parameter df. output is 0.2 times the fare column if gender is male else it is 0.5 times fare
"""
f1 = lambda df   : 0.2 * df['Fare'] if df['Gender'] == 'male' else 0.5 * df['Fare']


#store the result of apply function call into answer.
answer= df1[['Gender','Fare']].apply( f1 , axis=1 )


#insert at position 10, column name Discount, answer series
df1.insert(10,'Discount',answer)
```

In [35]:
```
df1
```

Out[35]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Discount	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	1.45000	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	35.64165	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	3.96250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	26.55000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	1.61000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev.	male	27.0	0	0	211536	13.0000	2.60000	NaN	S

				Juozas									
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	15.00000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	11.72500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	6.00000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	1.55000	NaN	Q

891 rows × 13 columns

In [ ]:
```
"""
mapping is a strategy of taking a function and applyting it on all values in a collection

E.g : - using map on a list of values,    using a apply with a function name on a data frame

"""
```

In [ ]:
```
"""
filtering strategy----> we remove items from a collection based on a condition
"""
```

In [36]:
```
#[ 1, 2, 3, 4, 5]


#only allow even numbers in this list. remove everything else

###IMPORTANT: lambda output should be a boolean!!! for filter to work

#take an input x. check if input is an even number
print( list( filter(  lambda x : x %2==0, [1,2,3,4,5]  ) ) ) #values satifying the given criteria
```

```
[2, 4]
```

In [37]:
```
#filter and get numbers divisible by 7
print( list( filter(  lambda x : x %7==0, [1,42,35,49,56]  ) ) ) #values satifying the given criteria
```

```
[42, 35, 49, 56]
```

In [38]:
```
vowels=['a',"e","i","o","u"]


print( list( filter(  lambda letter : letter in vowels, "harshit"  ) ) ) #values satifying the given criteria
```

```
['a', 'i']
```

In [42]:
```
df1
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Discount	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	1.45000	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	35.64165	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	3.96250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	26.55000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	1.61000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	2.60000	NaN	S
				Graham, Miss.									

887	888	1	1	Margaret Edith	female	19.0	0	0	112053	30.0000	15.00000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	11.72500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	6.00000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	1.55000	NaN	Q

891 rows × 13 columns

In [49]:
```
#lambda expression as a filter condition in pandas data frame

#extract rows where discount is more than 30 pounds ?

df1.apply(  lambda df1 : df1 ['Discount'] > 30 , axis= 1  )
```

Out[49]:
```
0      False
1       True
2      False
3      False
4      False
       ...
886    False
887    False
888    False
889    False
890    False
Length: 891, dtype: bool
```

In [56]:
```
condition = (df1['Discount'] > 30 )

df1[condition]
```

Out[56]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Discount	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	35.64165	C85	C
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	52.60000	C23 C25 C27	S
31	32	1	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	NaN	1	0	PC 17569	146.5208	73.26040	B78	C
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	38.36460	D33	C
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000	40.00000	B28	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
835	836	1	1	Compton, Miss. Sara Rebecca	female	39.0	1	1	PC 17756	83.1583	41.57915	E49	C
849	850	1	1	Goldenberg, Mrs. Samuel L (Edwiga Grabowska)	female	NaN	1	0	17453	89.1042	44.55210	C92	C
856	857	1	1	Wick, Mrs. George Dennick (Mary Hitchcock)	female	45.0	1	1	36928	164.8667	82.43335	NaN	S
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	34.77500	NaN	S
				Potter, Mrs. Thomas Jr									

879	880	1	1	(Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	41.57915	C50	C

82 rows × 13 columns

In [54]:
```
df1[df1.apply(lambda df : df['Discount'] > 30 , axis = 1)] #avinash's solution
```

Out[54]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Discount	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	35.64165	C85	C
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	52.60000	C23 C25 C27	S
31	32	1	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	NaN	1	0	PC 17569	146.5208	73.26040	B78	C
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	38.36460	D33	C
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000	40.00000	B28	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
835	836	1	1	Compton, Miss. Sara Rebecca	female	39.0	1	1	PC 17756	83.1583	41.57915	E49	C
849	850	1	1	Goldenberg, Mrs. Samuel L (Edwiga Grabowska)	female	NaN	1	0	17453	89.1042	44.55210	C92	C
856	857	1	1	Wick, Mrs. George Dennick (Mary Hitchcock)	female	45.0	1	1	36928	164.8667	82.43335	NaN	S
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	34.77500	NaN	S
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	41.57915	C50	C

82 rows × 13 columns

In [ ]:
```
#reduce-----> apply an operation on first item in a collection, calculate result.
            In a loop
            -----> take the result of previous step, apply opertion. get new result
```

In [58]:
```
"""
2**1 = 2

2 ** 2 = 4

4 ** 3 = 64

64 ** 4 = some number


some number ** 5


"""
```

Out[58]:
```
'\n2**1 = 2\n\n2 ** 2 = 4\n\n4 ** 3 = 64\n\n64 ** 4 = some number\n\n\nsome number ** 5\n\n\n'
```

In [59]:
```
powers=[1,2,3,4,5]
```

```
ans = 2
for num in powers:
    ans **= num
    print(ans)
```

```
2
4
64
16777216
132922799578491587290380706028034457 6
```

In [65]:
```
from functools import reduce

#reduce always returns a single answer
print( reduce(  lambda x, y : x**y, [2,1,2,3,4,5] ) )
```

```
132922799578491587290380706028034457 6
```

In [66]:
```
#reduction based multiplication


print( reduce(  lambda x, y : x*y, [1,2,3,4,5] ) )
```

```
120
```

In [ ]:
```
#pandas--->
```

In [70]:
```
#shifting and lagging
df.drop('EXAMPLE',inplace=True,axis=1)
df['Previous'] = df['Close'].shift(1)
```

In [73]:
```
df['Difference'] = df['Close'] - df['Previous']
df[['Close','Previous','Difference']]
```

Out[73]:

	Close	Previous	Difference
0	311.600006	NaN	NaN
1	306.799988	311.600006	-4.800018
2	301.899994	306.799988	-4.899994
3	303.899994	301.899994	2.000000
4	315.899994	303.899994	12.000000
...	...	...	...
733	15.450000	15.400000	0.050000
734	15.450000	15.450000	0.000000
735	15.350000	15.450000	-0.100000
736	15.750000	15.350000	0.400000
737	17.299999	15.750000	1.549999

738 rows × 3 columns

In [81]:
```
#compare on a montly basis. show me trend(difference) when compared with previous month

df=pd.read_csv("/home/harshit/DataSets/YESBANK.NS.csv",parse_dates=['Date'],index_col='Date')
df
```

Out[81]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-12-11	313.500000	315.799988	310.600006	311.600006	300.880615	4416465.0
2017-12-12	312.000000	312.000000	305.899994	306.799988	296.245758	5457103.0
2017-12-13	306.350006	307.350006	301.049988	301.899994	291.514282	6911856.0
2017-12-14	303.899994	304.649994	301.750000	303.899994	293.445526	4904177.0
2017-12-15	307.000000	317.450012	307.000000	315.899994	305.032715	20571225.0
...	...	...	...	...	...	...
2020-12-02	15.700000	15.900000	14.850000	15.450000	15.450000	311349886.0
2020-12-03	15.650000	15.800000	15.250000	15.450000	15.450000	152445535.0
2020-12-04	15.600000	15.600000	15.050000	15.350000	15.350000	149691622.0
2020-12-07	15.650000	15.850000	15.500000	15.750000	15.750000	193242183.0
2020-12-08	16.000000	17.299999	16.000000	17.299999	17.299999	562741066.0

738 rows × 6 columns

In [77]:
```
#save your resampled data into another variable
temp=df.resample('M').mean()

temp['Previous'] = temp['Close'].shift(1) #use shift to create a previous close column

#use current day minus previous day close to get difference
temp['Difference'] = temp['Close'] - temp['Previous']


temp[['Close','Previous','Difference']]
```

Out[77]:

Date	Close	Previous	Difference
2017-12-31	311.078570	NaN	NaN
2018-01-31	340.895455	311.078570	29.816885
2018-02-28	327.386839	340.895455	-13.508616
2018-03-31	307.473683	327.386839	-19.913157
2018-04-30	319.123807	307.473683	11.650124
2018-05-31	343.768182	319.123807	24.644375
2018-06-30	335.669050	343.768182	-8.099132
2018-07-31	369.111360	335.669050	33.442310
2018-08-31	376.383336	369.111360	7.271976
2018-09-30	288.986109	376.383336	-87.397227
2018-10-31	214.519047	288.986109	-74.467062
2018-11-30	200.140000	214.519047	-14.379046
2018-12-31	178.002500	200.140000	-22.137500
2019-01-31	195.954348	178.002500	17.951848
2019-02-28	202.921053	195.954348	6.966706
2019-03-31	246.150002	202.921053	43.228949
2019-04-30	254.781578	246.150002	8.631575
2019-05-31	152.640909	254.781578	-102.140668
2019-06-30	122.994737	152.640909	-29.646173
2019-07-31	93.452174	122.994737	-29.542562
2019-08-31	73.200000	93.452174	-20.252174
2019-09-30	59.157895	73.200000	-14.042105
2019-10-31	47.128948	59.157895	-12.028947
2019-11-30	67.110000	47.128948	19.981052
2019-12-31	51.452381	67.110000	-15.657619
2020-01-31	42.250000	51.452381	-9.202381
2020-02-29	36.447368	42.250000	-5.802632
2020-03-31	33.628571	36.447368	-2.818797
2020-04-30	26.355555	33.628571	-7.273016
2020-05-31	27.410526	26.355555	1.054971
2020-06-30	28.093182	27.410526	0.682656
2020-07-31	19.869565	28.093182	-8.223617
2020-08-31	14.614286	19.869565	-5.255279
2020-09-30	13.906818	14.614286	-0.707468
2020-10-31	12.930952	13.906818	-0.975866
2020-11-30	13.744737	12.930952	0.813784
2020-12-31	15.783333	13.744737	2.038596

In [106…
```
#100 shares purchased on 29th december 2017 after the closing of the market(order)

#if I sell my shares on 31st December 2018, what is my exact(net) profit/ loss?
#(final day price - first day price)
df.loc['2018-12-31']['Adj Close'] - df.loc['2017-12-29']['Adj Close']
```

Out[106… -127.36854599999998

In [82]:
```
df
```

Out[82]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-12-11	313.500000	315.799988	310.600006	311.600006	300.880615	4416465.0
2017-12-12	312.000000	312.000000	305.899994	306.799988	296.245758	5457103.0
2017-12-13	306.350006	307.350006	301.049988	301.899994	291.514282	6911856.0
2017-12-14	303.899994	304.649994	301.750000	303.899994	293.445526	4904177.0
2017-12-15	307.000000	317.450012	307.000000	315.899994	305.032715	20571225.0
...	...	...	...	...	...	...
2020-12-02	15.700000	15.900000	14.850000	15.450000	15.450000	311349886.0
2020-12-03	15.650000	15.800000	15.250000	15.450000	15.450000	152445535.0
2020-12-04	15.600000	15.600000	15.050000	15.350000	15.350000	149691622.0
2020-12-07	15.650000	15.850000	15.500000	15.750000	15.750000	193242183.0
2020-12-08	16.000000	17.299999	16.000000	17.299999	17.299999	562741066.0

738 rows × 6 columns

In [87]:
```
df.loc['2017-12-29' :'2017-12-30'  ]
```

Out[87]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-12-29	313.5	315.799988	312.799988	315.149994	304.308472	4718184.0

In [86]:
```
df.loc['2018-12-31': '2018-01']
```

Out[86]:
```
Open          1.832000e+02
High          1.838500e+02
Low           1.810000e+02
Close         1.818000e+02
Adj Close     1.769399e+02
Volume        1.897086e+07
Name: 2018-12-31 00:00:00, dtype: float64
```

In [101…
```
df
```

Out[101…

Date	Open	High	Low	Close	Adj Close	Volume
2017-12-11	313.500000	315.799988	310.600006	311.600006	300.880615	4416465.0
2017-12-12	312.000000	312.000000	305.899994	306.799988	296.245758	5457103.0
2017-12-13	306.350006	307.350006	301.049988	301.899994	291.514282	6911856.0
2017-12-14	303.899994	304.649994	301.750000	303.899994	293.445526	4904177.0
2017-12-15	307.000000	317.450012	307.000000	315.899994	305.032715	20571225.0
...	...	...	...	...	...	...
2020-12-02	15.700000	15.900000	14.850000	15.450000	15.450000	311349886.0
2020-12-03	15.650000	15.800000	15.250000	15.450000	15.450000	152445535.0
2020-12-04	15.600000	15.600000	15.050000	15.350000	15.350000	149691622.0
2020-12-07	15.650000	15.850000	15.500000	15.750000	15.750000	193242183.0
2020-12-08	16.000000	17.299999	16.000000	17.299999	17.299999	562741066.0

738 rows × 6 columns

In [109…
```
data=df.loc['2018'].resample('Q').mean()
```

In [113…
```
#change in percentage of current reading compared with previous reading!
data[['Adj Close']].pct_change()
```

Out[113…

Date	Adj Close
2018-03-31	NaN
2018-06-30	0.023992
2018-09-30	0.050555
2018-12-31	-0.431470

In [ ]:
```
#net change UP TO second quarter?
0.23396
```

In [115…
```
#net change UP TO third quarter?
0.023992 + 0.050555
```

Out[115… 0.074547

In [ ]:
```
#net change UP TO fourth quarter?
```

In [114…
```
data[['Adj Close']].pct_change().sum() #total change in 4 quarters
```

Out[114…
```
Adj Close   -0.356923
dtype: float64
```

In [117…
```
#cumulative change up to a certain reading
data[['Adj Close']].pct_change().cumsum()
```

Out[117…

Date	Adj Close
2018-03-31	NaN
2018-06-30	0.023992
2018-09-30	0.074547
2018-12-31	-0.356923

In [ ]:
```
#cumulative change every month in 2018

#calculate the cumulative closing price for all weeks in January 2018 and january 2019

"""
####resampling weekly,
#####loc for finding 2018 and 2019 data
cumulative sum
####concat
"""

"""
what operations i
"""
```

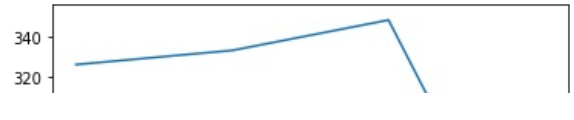
In [129…
```
pd.concat([df.loc['2018-01'], df.loc['2019-01']]).resample('W')[['Adj Close']].mean().cumsum().dropna()
```

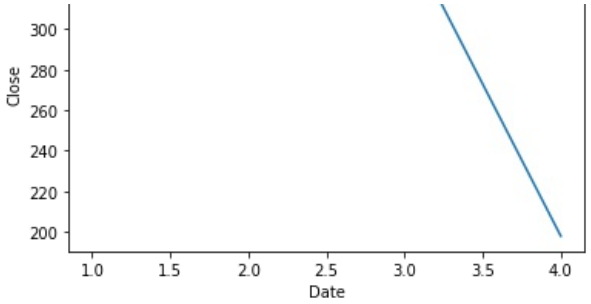
Out[129…

Date	Adj Close
2018-01-07	307.070129
2018-01-14	635.142010
2018-01-21	964.179504
2018-01-28	1312.108940
2018-02-04	1655.137952
2019-01-06	1835.837113
2019-01-13	2018.246794
2019-01-20	2214.223820
2019-01-27	2412.069517
2019-02-03	2607.696165

In [133…
```
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1,1, figsize=(15,5))

data1=df.loc['2018']
data2=df.loc['2019']

ans=data.groupby( data.index.quarter  )[['Close']].mean()
sns.lineplot( x=ans.index, y='Close' ,data=ans)
```

Out[133… <AxesSubplot:xlabel='Date', ylabel='Close'>





In [ ]:

In [120…
```
df
```

Out[120…

Date	Open	High	Low	Close	Adj Close	Volume
2017-12-11	313.500000	315.799988	310.600006	311.600006	300.880615	4416465.0
2017-12-12	312.000000	312.000000	305.899994	306.799988	296.245758	5457103.0
2017-12-13	306.350006	307.350006	301.049988	301.899994	291.514282	6911856.0
2017-12-14	303.899994	304.649994	301.750000	303.899994	293.445526	4904177.0
2017-12-15	307.000000	317.450012	307.000000	315.899994	305.032715	20571225.0
...	...	...	...	...	...	...
2020-12-02	15.700000	15.900000	14.850000	15.450000	15.450000	311349886.0
2020-12-03	15.650000	15.800000	15.250000	15.450000	15.450000	152445535.0
2020-12-04	15.600000	15.600000	15.050000	15.350000	15.350000	149691622.0
2020-12-07	15.650000	15.850000	15.500000	15.750000	15.750000	193242183.0
2020-12-08	16.000000	17.299999	16.000000	17.299999	17.299999	562741066.0

738 rows × 6 columns

In [121…
```
df.loc['2018-01']
```

Out[121…

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-01	315.500000	317.750000	311.299988	312.600006	301.846252	4019878.0
2018-01-02	313.399994	314.000000	307.149994	311.649994	300.928894	5224976.0
2018-01-03	312.000000	316.500000	311.149994	315.850006	304.984436	5672263.0
2018-01-04	316.000000	318.399994	313.000000	317.100006	306.191437	5667580.0
2018-01-05	317.500000	337.899994	317.450012	332.850006	321.399628	30720675.0
2018-01-08	336.000000	341.299988	331.299988	333.600006	322.123779	12747890.0
2018-01-09	334.899994	342.799988	327.549988	341.350006	329.607208	13282560.0
2018-01-10	341.500000	342.350006	335.450012	339.799988	328.110474	10385044.0
2018-01-11	339.000000	344.250000	335.299988	343.149994	331.345276	8266126.0
2018-01-12	344.100006	344.700012	337.549988	340.899994	329.172668	5688676.0
2018-01-15	341.899994	343.700012	335.100006	336.000000	324.441223	7142164.0
2018-01-16	336.000000	338.750000	328.000000	334.850006	323.330811	7296505.0
2018-01-17	335.100006	343.500000	331.399994	342.399994	330.621063	7985222.0
2018-01-18	350.000000	356.899994	332.350006	341.200012	329.462372	35465087.0
2018-01-19	347.500000	352.250000	339.100006	349.350006	337.332001	21425789.0
2018-01-22	349.950012	358.250000	348.750000	355.350006	343.125580	13456538.0
2018-01-23	359.850006	360.399994	352.299988	359.549988	347.181091	10196645.0
2018-01-24	357.000000	366.299988	356.000000	364.799988	352.250488	11258771.0
2018-01-25	364.500000	364.500000	355.649994	361.600006	349.160583	8963188.0
2018-01-29	361.200012	363.700012	355.549988	358.000000	345.684387	7931235.0
2018-01-30	358.000000	360.799988	351.850006	353.350006	341.194397	7890491.0
2018-01-31	353.000000	356.549988	350.450012	354.399994	342.208252	8527044.0

In [118…
```
df.loc[['2018-01','2019-01']]
```

Out[118…

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-01	315.500000	317.750000	311.299988	312.600006	301.846252	4019878.0
2019-01-01	182.600006	185.899994	181.000000	184.250000	179.324417	24160878.0

In [134…
```
import numpy as np
```

In [137…
```
print( type( df['Close'] ))
```
```
<class 'pandas.core.series.Series'>
```

In [ ]:
```
"""
a data frame is a combination of number of numpy arrays

[ Machine learning training and testing of data! ]
"""
```

In [136…
```
#number crunching


arr1=np.array( [1,2,3,4,5]  )
print( arr1)
print(type(arr1))
```
```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [138…
```
print( arr1.shape) #1 dimension array of 5 values in a single row
```
```
(5,)
```

In [139…
```
ones_arr =  np.ones(shape=(7,)) #1 _D array of 7 '1' values put together in a single row
```

Out[139…
```
array([1., 1., 1., 1., 1., 1., 1.])
```

In [140…
```
ones_arr =  np.ones(shape=(7,4))
print( ones_arr )
```
```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [141…
```
zero_arr =  np.zeros(shape=(7,4))
print( zero_arr )
```
```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [144…
```
ones_arr_int = np.ones(shape=(7,4), dtype=np.int32)
print(ones_arr_int.dtype)
```
```
int32
```

In [142…
```
print( arr1.dtype, ones_arr.dtype, zero_arr.dtype  )
```
```
int64 float64 float64
```

In [148…
```
data=np.array(  [ [1,2,3],
           [4,5,6],
           [7,8,9]
           ], dtype=np.int64 )
```

In [150…
```
data.shape
```

Out[150…
```
(3, 3)
```

In [154…
```
data
```

Out[154…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [152…
```
#how to represent 9 values in rows and columns in different shapes?
```

```
data.reshape( (9,1)  )
```

Out[152…
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8],
       [9]])
```

In [153…
```
data.reshape( (1,9)  )
```

Out[153…
```
array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

In [ ]:
```
#numpy array support arithmetic operations as well
```

In [155…
```
arr1 = np.array( [1,2,3,4,5] )

arr2 = np.array([10,20,30,40,50])

print(arr1 +  arr2 )
```

```
[11 22 33 44 55]
```

In [156…
```
arr1 = np.array( [1,2,3,4,5] )

arr2 = np.array([10,20,30,40,50])

print(arr1 -  arr2 )
```

```
[ -9 -18 -27 -36 -45]
```

In [157…
```
#element wise multiplication

arr1 = np.array( [1,2,3,4,5] )

arr2 = np.array([10,20,30,40,50])

print(arr1 *  arr2 )
```

```
[ 10  40  90 160 250]
```

In [159…
```
#will not work for incompatible shape
arr1 = np.array( [1,2,3,4] )

arr2 = np.array([10,20,30,40,50])

print(arr1 *  arr2 )
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-159-818cc2e40153> in <module>
      3 arr2 = np.array([10,20,30,40,50])
      4
----> 5 print(arr1 *  arr2 )

ValueError: operands could not be broadcast together with shapes (4,) (5,)
```

In [166…
```
arr1 = np.array( [1,2,3,4] )

arr1=arr1.reshape((2,2))

arr2 = np.array([10,20,30,40])
arr2=arr2.reshape((4,1))

print(arr1.reshape((4,1)) + arr2)
print("-------------------------------")
print(arr1 + arr2.reshape(2,2))
```

```
[[11]
 [22]
 [33]
 [44]]
-------------------------------
[[11 22]
 [33 44]]
```

In [167…
```
print(arr1.reshape( arr2.shape ) + arr2)
```

```
[[11]
 [22]
 [33]
 [44]]
```

In [171…
```
#generate equidistant points as an array?
```

```
#start, ending and total number of values
print( np.linspace( 1, 10 , 2 ) )
print( np.linspace( 1, 10 , 5 ) )

print( np.linspace( 1, 10 , 4 ) )
```

```
[ 1. 10.]
[ 1.    3.25  5.5   7.75 10.  ]
[ 1.  4.  7. 10.]
```

In [172…
```
#random values


np.random.randint(1,10, 5) #an array of 5 random values between 1 and 10
```

Out[172…
```
array([6, 9, 3, 1, 6])
```

In [174…
```
#7 *7 matrix of random number between 500 and 8971

np.random.randint(500,8971, 49 ).reshape( (7,7)  )
```

Out[174…
```
array([[4120, 2060, 4474, 3405, 1493, 7950, 2584],
       [8616, 8168, 4048, 3984, 3296, 8755, 5518],
       [8406, 2875, 4170, 3980, 6834, 8011, 6749],
       [4590, 1296, 1799, 3512, 3625, 5069, 4435],
       [ 977, 8397, 7780, 1506, 8757, 2423, 7053],
       [2776, 6212, 6698, 1512, 4829, 7592, 7270],
       [3067, 6864, 7487, 3063, 5402,  679,  729]])
```

In [177…
```
arr1=np.array(  [ [1,2,3],
           [4,5,6],
           [7,8,9]
           ], dtype=np.int64 )


arr2= np.array( [ [1,2,3] ] )

print(arr1.shape, arr2.shape)
print( arr1.size, arr2.size   )
```

```
(3, 3) (1, 3)
9 3
```

In [178…
```
print(arr1 + arr2)
```

```
[[ 2  4  6]
 [ 5  7  9]
 [ 8 10 12]]
```

In [ ]:
```
#if the number of columns are same for 2 arrays BUT THE SHAPE IS NOT, numpy performs broadcasting!


# R programming language -----> recycling

[1,2,3]                [1,2,3]
                       [1,2,3]
[4,5,6]                [1,2,3]
[7,8,9]
```