In [1]:
```
import pandas as pd
```

In [2]:
```
groceries = pd.Series(data = [30, 6, 'Yes', "No"], index = ['eggs', 'apples', 'mill
groceries
```

Out[2]:
```
eggs       30
apples      6
milk      Yes
bread      No
dtype: object
```

In [3]:
```
#size of each dimension in data
groceries.shape
```

Out[3]:
```
(4,)
```

In [4]:
```
#number of dimensions of data
groceries.ndim
```

Out[4]:
```
1
```

In [5]:
```
#total number of values in array
groceries.size
```

Out[5]:
```
4
```

In [6]:
```
#gives the index labels
groceries.index
```

Out[6]:
```
Index(['eggs', 'apples', 'milk', 'bread'], dtype='object')
```

In [7]:
```
#gives the data values
groceries.values
```

Out[7]:
```
array([30, 6, 'Yes', 'No'], dtype=object)
```

In [8]:
```
#check if an index is in a series
'eggs' in groceries
```

Out[8]:
```
True
```

In [9]:
```
###############################################################################
```

In [10]:
```
#accessing and modifying the elements
```

In [11]:
```
#accessing via index labels
groceries['eggs']
```

Out[11]:
```
30
```

In [12]:
```
#getting multiple elements using a list
groceries[['milk', 'eggs']]
```

Out[12]:
```
milk    Yes
eggs     30
dtype: object
```

In [13]:
```
#accessing using numerical indices
```

```
groceries[0]
```

Out[13]:
```
30
```

In [14]:
```
#accessing last element using numerical indicies
groceries[-1]
```

Out[14]:
```
'No'
```

In [15]:
```
#accesing multiple elements using numerical indicies
groceries[[0, 1]]
```

Out[15]:
```
eggs      30
apples     6
dtype: object
```

In [16]:
```
#explicitly state that you are using a label index
groceries.loc[['eggs', 'apples']]
```

Out[16]:
```
eggs      30
apples     6
dtype: object
```

In [17]:
```
#explicitly state that you are using a numerical index
groceries.iloc[[0, -1]]
```

Out[17]:
```
eggs     30
bread    No
dtype: object
```

In [18]:
```
#you can change the elements
#change the number of eggs
groceries['eggs'] = 2
groceries
```

Out[18]:
```
eggs        2
apples      6
milk      Yes
bread      No
dtype: object
```

In [19]:
```
#removing an element from the series (Brackets)
groceries.drop('apples')
#it doesn't change the original series
```

Out[19]:
```
eggs       2
milk     Yes
bread     No
dtype: object
```

In [20]:
```
#to change the original series we set the inplace parameter to true
groceries.drop('apples', inplace=True)
```

In [21]:
```
groceries
```

Out[21]:
```
eggs       2
milk     Yes
bread     No
dtype: object
```

In [22]:
```
################################################################################
```

In [23]:
```
#We can do element wise arithmetic operations on a pandas series
```

In [24]:
```
fruits = pd.Series([10, 3, 6], ['apples', 'oranges', 'bananas'])
fruits
```

Out[24]:
```
apples     10
oranges     3
bananas     6
dtype: int64
```

In [25]:
```
fruits + 2
```

Out[25]:
```
apples     12
oranges     5
bananas     8
dtype: int64
```

In [26]:
```
fruits - 2
```

Out[26]:
```
apples     8
oranges    1
bananas    4
dtype: int64
```

In [27]:
```
fruits * 2
```

Out[27]:
```
apples     20
oranges     6
bananas    12
dtype: int64
```

In [28]:
```
fruits / 2
```

Out[28]:
```
apples     5.0
oranges    1.5
bananas    3.0
dtype: float64
```

In [29]:
```
# we can also apply numpy mathematical functions on the series
```

In [30]:
```
import numpy as np
fruits
```

Out[30]:
```
apples     10
oranges     3
bananas     6
dtype: int64
```

In [31]:
```
np.sqrt(fruits)
```

Out[31]:
```
apples     3.162278
oranges    1.732051
bananas    2.449490
dtype: float64
```

In [32]:
```
np.exp(fruits)
```

Out[32]:
```
apples     22026.465795
oranges       20.085537
bananas      403.428793
dtype: float64
```

In [33]:
```
np.power(fruits, 2)
```

Out[33]:
```
apples     100
oranges      9
bananas     36
dtype: int64
```

In [34]:
```
#we can do arithmetic operation on specific elements like this
```

In [35]:
```
fruits['bananas'] * 2
```

Out[35]:
```
12
```

In [36]:
```
fruits.iloc[0] - 2
```

Out[36]:
```
8
```

In [37]:
```
fruits[['apples', 'oranges']] * 2
```

Out[37]:
```
apples     20
oranges     6
dtype: int64
```

In [38]:
```
fruits.loc[['apples', 'oranges']] // 2
```

Out[38]:
```
apples     5
oranges    1
dtype: int64
```

In [39]:
```
#you can do arithmetic operation on a mixed series provided that the
#arithmetic opeartion is defined for all data types
```

In [40]:
```
groceries
```

Out[40]:
```
eggs       2
milk     Yes
bread     No
dtype: object
```

In [41]:
```
groceries * 2
```

Out[41]:
```
eggs          4
milk     YesYes
bread      NoNo
dtype: object
```

In [42]:
```
################################################################################
```

In [43]:
```
#Data frames is a 2d object
```

In [44]:
```
#creating a dataframe manually
```

In [45]:
```
items = {'Bob': pd.Series([245, 25, 55], index=['bike', 'pants', 'watch']),
         'Alice': pd.Series([40, 110, 500, 45], index=['book', 'glasses', 'bike',

type(items)
```

Out[45]:
```
dict
```

In [46]:
```
shopping_carts = pd.DataFrame(items)
shopping_carts
```

Out[46]:

	Bob	Alice
bike	245.0	500.0
book	NaN	40.0
glasses	NaN	110.0
pants	25.0	45.0
watch	55.0	NaN

In [47]:
```
data = {'Bob': pd.Series([245, 25, 55]),
        'Alice': pd.Series([40, 110, 500, 45])}
df = pd.DataFrame(data)
df
```

Out[47]:

	Bob	Alice
0	245.0	40
1	25.0	110
2	55.0	500
3	NaN	45

In [48]:
```
#getting the index labels
shopping_carts.index
```

Out[48]:
```
Index(['bike', 'book', 'glasses', 'pants', 'watch'], dtype='object')
```

In [49]:
```
#Getting the column labels
```

In [50]:
```
shopping_carts.columns
```

Out[50]:
```
Index(['Bob', 'Alice'], dtype='object')
```

In [51]:
```
#getting the values
```

In [52]:
```
shopping_carts.values
```

Out[52]:
```
array([[245., 500.],
       [ nan,  40.],
       [ nan, 110.],
       [ 25.,  45.],
       [ 55.,  nan]])
```

In [53]:
```
#it also uses the same attributes as a series
```

In [54]:
```
#getting info about shape
shopping_carts.shape
```

Out[54]:
```
(5, 2)
```

In [55]:
```
#getting info about dimension
shopping_carts.ndim
```

Out[55]:
```
2
```

In [56]:
```
#getting info about the number of values
```

```
shopping_carts.size
```

Out[56]:
```
10
```

In [57]:
```
#you can choose which data to put in a data frame
```

In [58]:
```
#dataframe for bob's shopping cart
bob_shopping_cart = pd.DataFrame(items, columns=['Bob'])
bob_shopping_cart
```

Out[58]:

	Bob
bike	245
pants	25
watch	55

In [59]:
```
#dataframe of select items
sel_shopping_cart = pd.DataFrame(items, index=['pants', 'book'])
sel_shopping_cart
```

Out[59]:

	Bob	Alice
pants	25.0	45
book	NaN	40

In [60]:
```
#dataframe of select items in alice's list
alice_shopping_cart = pd.DataFrame(items, index=['glasses', 'bike'], columns=['Ali
alice_shopping_cart
```

Out[60]:

	Alice
glasses	110
bike	500

In [61]:
```
#creating a dataframe from a dict of lists(lists must be of same length)
data = {"integers": [1, 2, 3],
        "Floats": [4.5, 8.2, 9.6]}

df = pd.DataFrame(data, index=['label 1', "label 2", 'label 3'])
df
```

Out[61]:

	integers	Floats
label 1	1	4.5
label 2	2	8.2
label 3	3	9.6

In [62]:
```
#creating a dataframe using a list of dicts
items = [{"bikes" : 20, 'pants': 30, "watches": 35}, {"watches": 10, "glasses" : 5(

store_items = pd.DataFrame(items, index=["store 1", "store 2"])
store_items
```

Out[62]:

	bikes	pants	watches	glasses
store 1	20	30	35	NaN
store 2	15	5	10	50.0

In [63]:
```
##########################################################################
```

In [64]:
```
#accesssing elements in dataframes
```

In [65]:
```
store_items["bikes"]
```

Out[65]:
```
store 1    20
store 2    15
Name: bikes, dtype: int64
```

In [66]:
```
store_items[["pants", 'watches']]
```

Out[66]:

	pants	watches
store 1	30	35
store 2	5	10

In [67]:
```
store_items.loc[['store 1']]
```

Out[67]:

	bikes	pants	watches	glasses
store 1	20	30	35	NaN

In [68]:
```
store_items['bikes']['store 2']
```

Out[68]:
```
15
```

In [69]:
```
#Adding a column to the data frame
store_items['shirts'] = [15, 2]
store_items
```

Out[69]:

	bikes	pants	watches	glasses	shirts
store 1	20	30	35	NaN	15
store 2	15	5	10	50.0	2

In [70]:
```
#adding new columns using arithmetic operations on other columns
store_items['suits'] = store_items['shirts'] + store_items['pants']
store_items
```

Out[70]:

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15	45
store 2	15	5	10	50.0	2	7

In [71]:
```
#to add a new row you have to create a new data frame
#then append it to the original.

new_items = [{"bikes" : 20, 'pants': 30, "watches": 35, 'glasses': 4}]
```

```
new_store = pd.DataFrame(new_items, index=['store 3'])
new_store
```

Out[71]:

	bikes	pants	watches	glasses
store 3	20	30	35	4

In [72]:
```
store_items = store_items.append(new_store)
store_items
```

```
C:\Users\minec\AppData\Local\Temp\ipykernel_9744\2720228100.py:1: FutureWarning: T
he frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
  store_items = store_items.append(new_store)
```

Out[72]:

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15.0	45.0
store 2	15	5	10	50.0	2.0	7.0
store 3	20	30	35	4.0	NaN	NaN

In [73]:
```
#creating a new variable which has a value as another
#appending the change to select rows
store_items['new_watches'] = store_items['watches'][1:]
store_items
```

Out[73]:

	bikes	pants	watches	glasses	shirts	suits	new_watches
store 1	20	30	35	NaN	15.0	45.0	NaN
store 2	15	5	10	50.0	2.0	7.0	10.0
store 3	20	30	35	4.0	NaN	NaN	35.0

In [74]:
```
#you can insert new columns whereever we want
#columns label data
store_items.insert(5, 'shoes', [8, 5, 0])
store_items
```

Out[74]:

	bikes	pants	watches	glasses	shirts	shoes	suits	new_watches
store 1	20	30	35	NaN	15.0	8	45.0	NaN
store 2	15	5	10	50.0	2.0	5	7.0	10.0
store 3	20	30	35	4.0	NaN	0	NaN	35.0

In [75]:
```
#we can delete columns using pop
store_items.pop("new_watches")
store_items
```

Out[75]:

	bikes	pants	watches	glasses	shirts	shoes	suits
store 1	20	30	35	NaN	15.0	8	45.0
store 2	15	5	10	50.0	2.0	5	7.0
store 3	20	30	35	4.0	NaN	0	NaN

In [76]:
```
store_items = store_items.drop(['watches', 'shoes'], axis=1)
```

```
store_items
```

Out[76]:

	bikes	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
store 2	15	5	50.0	2.0	7.0
store 3	20	30	4.0	NaN	NaN

In [77]:
```
store_items = store_items.drop(['store 1', 'store 2'], axis=0)
store_items
```

Out[77]:

	bikes	pants	glasses	shirts	suits
store 3	20	30	4.0	NaN	NaN

In [78]:
```
#we can rename columns or rows
#for columns
store_items = store_items.rename(columns={"bikes": "hats"})
store_items
```

Out[78]:

	hats	pants	glasses	shirts	suits
store 3	20	30	4.0	NaN	NaN

In [79]:
```
#for rows
store_items = store_items.rename(index={"store 3": "last store"})
store_items
```

Out[79]:

	hats	pants	glasses	shirts	suits
last store	20	30	4.0	NaN	NaN

In [80]:
```
#we can make the index the value of the columns
store_items = store_items.set_index("pants")
store_items
```

Out[80]:

pants	hats	glasses	shirts	suits
30	20	4.0	NaN	NaN

In [81]:
```
################################################################################
```

In [82]:
```
## CLEANING DATA ##
```

In [83]:
```
items = [{"bikes" : 20, 'pants': 30, "watches": 35, 'shirts': 15, 'shoes': 8, 'suit
        {'watches':10, 'glasses': 50, "bikes" : 15, 'pants': 5,'shirts': 2, 'shoes
        {"bikes" : 20, 'pants': 30, "watches": 35,  'glasses': 4,'shoes': 10}]

store_items = pd.DataFrame(items, index=['store 1', 'store 2', 'store 3'])
store_items
```

Out[83]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

In [84]:
```
#counting the nan values
```

In [85]:
```
#method 1 is null returns boolean true when nan value exists
x = store_items.isnull().sum().sum()
print(x)
```

```
3
```

In [86]:
```
#you can find the nan values using isnull().any()
store_items.isnull().any()
```

Out[86]:
```
bikes      False
pants      False
watches    False
shirts      True
shoes      False
suits       True
glasses     True
dtype: bool
```

In [87]:
```
#count the valid values
store_items.count()
```

Out[87]:
```
bikes      3
pants      3
watches    3
shirts     2
shoes      3
suits      2
glasses    2
dtype: int64
```

In [88]:
```
#removing nan values
store_items.dropna(axis = 0)#removes index
```

Out[88]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 2	15	5	10	2.0	5	7.0	50.0

In [89]:
```
store_items.dropna(axis=1)#removes colums
#doesn't modify the original , you can do so using inplace = true
```

Out[89]:

	bikes	pants	watches	shoes
store 1	20	30	35	8
store 2	15	5	10	5
store 3	20	30	35	10

In [90]:
```
#Replacing nan values
```

In [91]:
```
#replacing with zeros
store_items.fillna(0)
```

Out[91]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	0.0
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	0.0	10	0.0	4.0

In [92]:
```
#use forward filling (using the value from previous value)
store_items.fillna(method='ffill', axis=1)
```

Out[92]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20.0	30.0	35.0	15.0	8.0	45.0	45.0
store 2	15.0	5.0	10.0	2.0	5.0	7.0	50.0
store 3	20.0	30.0	35.0	35.0	10.0	10.0	4.0

In [93]:
```
#use backward filling (using the value from previous value)
store_items.fillna(method='backfill', axis=0)
```

Out[93]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	50.0
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

In [94]:
```
#using interpolation to replace nans
store_items.interpolate(method='linear', axis=0)
```

Out[94]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	2.0	10	7.0	4.0

In [95]:
```
store_items.interpolate(method='linear', axis=1)
```

Out[95]:

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20.0	30.0	35.0	15.0	8.0	45.0	45.0
store 2	15.0	5.0	10.0	2.0	5.0	7.0	50.0
store 3	20.0	30.0	35.0	22.5	10.0	7.0	4.0

In [96]:
```
##########################################################################
```