# PYTHON NOTES

# for

# Data Analysts/

# Business Analysts/

# Product Analysts

## BY MAZHER KHAN - IIT

### Data Science & Analytics - 6 years

| | |
|---|---|
| Telegram Link- | https://t.me/+XTjv6r80eDc5ZWU1 |
| Practice Workbook - 100 Days Challenge | https://docs.google.com/spreadsheets/d/1eP8evU2JIsawAVJ7GH_NNd_2xNLOT7abpJTs5O9iUQI/edit#gid=775777503 |
| Roadmap to Become a Data Analyst | https://topmate.io/mazher_khan/907723 |
| Follow on Linkedin | https://www.linkedin.com/in/mazher-khan/ |
| Follow on Instagram | https://www.instagram.com/khan.the.analyst |
| Book 1:1 Mentorship Plan - 1,3, 6 Months | https://www.preplaced.in/profile/mazher-khan |
| Book for Career Guidance, CV review & interview tip | https://topmate.io/mazher_khan |
| Follow on Youtube | https://youtube.com/@imzhr.?si=KdMGmWt-vTy12hxV |

# PYTHON: PROGRAMING LANGUAGE

1. **Key Features/Benefits of Python:**
   a. Free and open source
   b. Interpreted language – No complier required like C
   c. Dynamically Typed – No need to defined variable type

2. **Python Libraries:** NumPy, Pandas, Matplotlib, Sckit

   - **Pandas Data frames:** A pandas data frame is a data structure in pandas which is mutable.
   - **Pandas Series:** Series is one dimensional pandas' data structure which can store data of any type
   - **Pandas Group by:** A feature supported by Pandas used to split and group an object
     df.groupby('Type').count()
     Aggregate functions : Sum(), Mean(), Count(), Std()

3. **Pep8 (Python Enhancement Proposal)**
   Set of rules that specify how to format python code for maximum readability.

4. **Python Namespace:** Refers to the name which is assigned to each object (Variables and functions)

5. **Decorators:** Decorators are functions that take another function as argument to modify its behaviour without changing the function itself. (@)

6. **Indentation is required for Python**
7. **Python 2 require () while printing whereas Python 3 does.**
8. **How memory managed in Python?**
   i. All Python objects and data structures are in a private heap
   ii. The allocation of heap space for Python objects is done by Python's memory manager

9. **PYTHON Modules**
   i. Python modules are files containing Python code which can either be functions classes or variables.
   ii. A Python module is a .py file containing executable code.
   iii. OS / SYS / MATH

10. **.py V/S pyc files**
    .py: Python Source Code files
    .pyc: Bytecode of python files (created when code is imported from other source)

11. **Data Types:**
    a. Numbers: Integers/Floats
    b. List:
       i. An ordered sequence of items is called a list
       ii. The elements of a list may belong to different data types.

iii. Ex: [5,'market',2.4]

c. Tuple:
   i. It is also an ordered sequence of elements
   ii. Unlike lists, tuples are immutable, which means they can't be changed
   iii. Ex: (3,'tool',1)

d. Sets:
   i. Collection of unique items that are not in order
   ii. Ex: {7,6,8}

e. Dictionary:
   i. A dictionary stores values in key and value pairs where each value can be accessed through its key
   ii. Ex: {1:'apple',2:'mango}

   df.keys(): will give list of keys

f. Boolean: True or False

| Mutable | Immutable |
|---|---|
| List | Strings |
| Sets | Tuples |
| Dictionaries | Numbers |

**12. List vs Tuples**

| LIST | TUPLES |
|---|---|
| Mutable i.e they can be edited | Immutable (tuples are lists which can't be edited) |
| Slower | Faster |
| Syntax: list_1 = [10, 'Chelsea', 20] | Syntax: tup_1 = (10, 'Chelsea' , 20) |

**13. Array vs List**

**i.** Arrays: Only Single Data type can hold
ii. List: Any Data type can hold

**14. Dictionary and List comprehensions:** Another concise way to define dictionaries and lists.
List: x = [i for i in range (5)]
Dictionary: x=[i : i+2 for i in range(5)]

**15. Slicing:**

i. Slicing is used to access parts of sequences like lists, tuples, and strings.
ii. Syntax :> **[start:end:step]**

**16. Keyword:**

i. Reserved words that have special meaning
ii. And/OR/NOT

**17. Literals:**

**i.** Literals are the raw data that are assigned to variables or constants while programming.
**ii.** String Literals/Numeric Literals/Boolean Literals/Special Literals

18. **How to combine data frames in pandas**
    i. Concatenating Horizontally/Vertically
    ii. Joining

19. **Type conversion in Python**

| Data Type | From | To |
|---|---|---|
| int() / float | Any data | Int/float |
| ord() | Character | Integer |
| hex() | Integer | Hexadecimal |
| oct() | Integer | Octal |
| Tuple/Set/List/Dict | Any data | Tuple/Set/List/Dict |
| str() | Integer | String |

Convert list to an array : np.array()

20. **_INIT_**
    i. It is a method is automatically called to allocate memory when a new object/ instance of a class is created

21. **Lambda function**
    i. An anonymous function is known as a lambda function
    ii. This function can have any number of parameters but, can have just one statement.
    iii. a = lambda x,y : x+y
    iv. print (a (5, 6))

22. **Break:** Allows loop termination when some condition is met
23. **Continue:** Allows skipping some part of a loop when some specific condition is met
24. **Pass:** This is basically a null operation. Nothing happens when this is executed

25. **range vs xrange**

| range() | xrange() |
|---|---|
| Returns a list of integers | Returns a generator object |
| Execution speed is slower | Execution speed is faster |
| All arithmetic operations can be performed as it returns a list. | Such operations cannot be performed on xrange(). |

26. **split () –** uses a regex pattern to “split” a given string into a list.
27. **Sub ()** – finds all substrings where the regex pattern matches and then replace them with a different string

28. **Python Packages:** Contains modules

29. **How to delete a file?**
    i. os.remove("xyz.txt")

30. **Remove values from Python arrays?**

i. pop: a.pop(3)
ii. Remove: a.remove(1)

**31. NumPy Arrays over Nested Lists**

| NumPy Arrays | Nested Lists |
|---|---|
| Vectorized operations are allowed | Not allowed |
| Faster | Slower |

**32. How to add values to python array?**

i. Append (): a.append(2)
ii. Extend (): a.extend([1,2])
iii. Insert (): a.insert(2,4)

**33. Split ():** Separates a given string> a.split()

**34. Deep vs Shallow Copy**

i. Deep: Deep copy is used to store the values that are already copied
ii. Shallow: Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance.

**35. Comments in Python**

Single line: #
Multi line: ‘’’

**36. How do you reverse a string in Python?**

Stringname[::-1]

```
Name='Mazher'
Name[len(name)::-1]
```

**37. Tools present to perform statistical analysis.**

- Pychecker
- Pylint

**38. NumPy vs SciPy**

- NumPy: Numerical Python >Simple Maths problem
- SciPy: Scientific Python > Complex problems like Integration and Optimization

**39. How do you check if a Python string contains another string?**

“Python Programming” contains “Programming”

**40. How do concatenate two tuples?**

- tup1 = (1,”a”,True)  tup2 = (4,5,6)
- tup1+tup2

**41. Access first 5 and last 5 records from data frame?**

- Df.head(5)
- Df.tail(5)

**42. List to Tuple conversion**

```
my_list = [50, "Twenty", 110, "Fifty", "Ten", 20, 10, 80, "Eighty"]
```

```
my_tuple = (my_list[0], my_list[len(my_list) - 1], len(my_list))
print(my_tuple)
```

43. **List to array**

```
my_array = np.array(my_list)
```

44. **Check if a Python string contains another string**

```
a_string="Python Programming"
substring1="Programming"
print(a_string.find(substring1))
```

45. **How to create a data frame from lists?**

```
df=pd.DataFrame()
bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]
cars=["lamborghini","masserati","ferrari","hyundai","ford"]
df["cars"]=cars
df["bikes"]=bikes
```

46. **How to create a data frame from dictionary?**

```
bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]
cars=["lamborghini","masserati","ferrari","hyundai","ford"]
d={"cars":cars,"bikes":bikes}
df=pd.DataFrame(d)
```

47. **How to create a new column in pandas by using values from other columns?**

```
df["Sum"]=df["col1"]+df["col2"]
```

48. **How to delete a column or group of columns in pandas?**

```
df=df.drop(["col1"],axis=1)
```

49. **Given the following data frame drop rows having column values as A.**

```
df=df[df.col1!=1]
```

50. **find the highest paid player in each college in each team**

```
df.groupby(["Team","College"])["Salary"].max()
```

51. **find the min max and average salary of a player collegewise and teamwise**

```
df.groupby(["Team","College"])["Salary"].max.agg([('max','max'),('min','min'),('count','count'
),('avg','min')])
```

52. **Vstack() : function to align rows vertically**

```
print(np.vstack((n1,n2)))
```

53. **How to remove spaces from a string in Python?**

Use replace function

54. **Write a program in Python to execute the Bubble sort algorithm.**

```
def bs(a):
```

```
# a = name of list
  b=len(a)-1nbsp;
# minus 1 because we always compare 2 adjacent values
  for x in range(b):
      for y in range(b-x):
          a[y]=a[y+1]

  a=[32,5,3,6,7,54,87]
  bs(a)
```

**55. Write a program in Python to produce Star triangle.**

```
def pyfunc(r):
   for x in range(r):
      print(' '*(r-x-1)+'*'*(2*x+1))
pyfunc(9)
```

**56. Write a program to produce Fibonacci series in Python.**

```
# Enter number of terms neededbsp;#0,1,1,2,3,5....
a=int(input("Enter the terms"))
f=0;#first element of series
s=1#second element of series
if a=0:
  print("The requested series is",f)
else:
 print(f,s,end=" ")
  for x in range(2,a):
      print(next,end=" ")
      f=s
      s=next
```

**57. Write a program in Python to check if a number is prime.**

```
a=int(input("enter number"))
if a=1:
  for x in range(2,a):
      if(a%x)==0:
       print("not prime")
  break
  else:
    print("Prime")
else:
  print("not prime")
```

**58. Write a program in Python to check if a sequence is a Palindrome.**

```
a=input("enter sequence")
b=a[::-1]
if a==b:
 print("palindrome")
else:
```

```
print("Not a Palindrome")
```

**59. Write a sorting algorithm for a numerical dataset in Python.**

```
list = ["1", "4", "0", "6", "9"]
list = [int(i) for i in list]
list.sort()
print (list)
```

**60. Looking at the below code, write down the final values of A0, A1, …An.**

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
print(A0,A1,A2,A3,A4,A5,A6)
```

**61. Plot charts**

| **import seaborn as sns** | |
|---|---|
| Line | sns.lineplot(data=loan_amnt) |
| Bar | sns.barplot(x=cr_data[‘cb_person_default_on_file’], y=cr_data[‘loan_int_rate’]) |
| Heat Maps | sns.heatmap(num_data.corr(), annot=True) |
| Scatter Plot | sns.scatterplot(x=cr_data[‘loan_amnt’], y=cr_data[‘person_income’]) |
| Distribution Chart | sns.heatmap(num_data.corr(), annot=True) |

**CODE**

```
import numpy as np

my_list=[1,2,3]
my_list

my_list_array=np.array(my_list)
```

**###Built-in methods to generate Arrays**

```
np.arange(1,8)
np.arange(0,11,3)
np.zeros(3)
np.zeros((3,3))
np.ones(3)
np.ones((3,3))
np.linspace(0,10,30)
np.eye(3)
```

**#Create an array of the given shape and populate it with random samples from a uniform** distribution over [0, 1).

```
np.random.rand(5)
np.random.rand(5,5)
#Return a sample (or samples) from the "standard normal" distribution
np.random.randn(5)
np.random.randn(5,5)

arr=np.arange(25)
arr.reshape(5,5)
arr.max()
arr.min()
arr.dtype()
```

**###Numpy Indexing and Selection**

```
a=np.arange(2,8)
a[2]
a[:2]
a[2:]
a[2:5]

arr_2d = np.array(([5,10,15],
                   [20,25,30],
                   [35,40,45]))
arr_2d[1][1:3]


b=np.arange(1,8)
```

```
x=b<4
b[x]
b[b<3]
b[b>3]
```

**#######OPERATIONS**

```
c=np.arange(5)
c=c+c
np.sqrt(c)
np.max(c)
np.min(c)
```

**########SERIES**

```
import numpy as np
import pandas as pd

labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}

pd.Series(data=my_list)
pd.Series(data=my_list,index=labels)
pd.Series(data=my_list,index=arr)
pd.Series(d,labels)
pd.Series(labels)
pd.Series(my_list)
pd.Series(arr)
pd.Series(d)
```

**#Index**

```
ser1=pd.Series(my_list,index=['US','INDIA','CHINA'])
ser1['US']
ser2=pd.Series(my_list,index=['US','INDIA','GERMANY'])
Sera=ser1+ser2
ser1
ser2
Sera
```

**##############DATA FRAMES**

```
from numpy.random import randn
np.random.seed(101)
df = pd.DataFrame(randn(3,3),index='A B C'.split(),columns='X Y Z'.split())
df[['X','Y']]
type(df[['X','Y']])
```

```
df['X+Y']=df['X']+ df['Y']
df.drop('X+Y',axis=1,inplace=True)
df
df.drop('C',axis=0)
df
```

**#Selecting rows**

```
df.loc['B']
df.iloc[1]
df
df['Y'].loc['B']
df['Y'].iloc[1]
df
df[df<0]
df[df['Y']<0]
States='USA CHINA INDIA'.split()
df['New']=States
df
df.set_index('New',inplace=True)
df
```

**###MISSING DATA**

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
df.dropna()
df.dropna(axis=1)
df.dropna(axis=0)
df.dropna(thresh=2,axis=0)
df.fillna(value=0)
df
df['B'].fillna(value=df['B'].mean())
df['A'].fillna(value=df['A'].mean())
```

**######GROUP BY**

```
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
        'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
        'Sales':[200,120,340,124,243,350]}

df = pd.DataFrame(data)
df
sf=df.groupby('Company')
sf.mean()
sf.min()
sf.max()
sf.std()
sf.count()
```

```
sf.describe().transpose()['FB']
```

**###MERGING, JOINING AND CONCAT**

```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])

df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
```

**#Concat**

```
pd.concat([df1,df2,df3])
pd.concat([df1,df2,df3],axis=0)
```

**#MERGE**

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})

pd.merge(left,right,how='inner',on=['key','key'])
pd.merge(left,right,how='outer',on=['key','key'])
```

**#JOIN**

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                       index=['K0', 'K2', 'K3'])


left.join(right)
```

```
left.join(right, how='outer')
```

**#######OPERATIONS**

```
df
df.head(3)
df['Company'].unique()
df['Company'].nunique()
df['Company'].count()
df['Company'].value_counts()
```

**#Selecting DATA**

```
df[(df['Company']=='GOOG') & (df['Sales']>200)]

def times2(x):
   return x*2

df['Sales'].apply(times2)
df['Company'].apply(len)
df['Sales'].sum()
df.columns

df.sort_values(by='Sales', ascending=False)
df.isnull()
df.fillna(value=0)
```

**#PIVOT**

```
data = {'A':['foo','foo','foo','bar','bar','bar'],
    'B':['one','one','two','two','one','one'],
      'C':['x','y','x','y','x','y'],
      'D':[1,3,2,5,4,1]}

df = pd.DataFrame(data)
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
```

**###READING INPUT_OUTPUT**

```
#INPUT
df = pd.read_csv('example')
pd.read_excel('Excel_Sample.xlsx',sheetname='Sheet1')
df = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')
#OUTPUT
df.to_csv('example',index=False)
df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```