Getter and Setter in Python

Getters and Setters in python are often used when:

- We use getters & setters to add validation logic around getting and setting a value.
- To avoid direct access of a class field i.e. private variables cannot be accessed directly or modified by external user.

```
class Getset:
    def __init__(self, age):
        self.__age = age

        # getter method
    def get_age(self):
        return self.__age

        # setter method
    def set_age(self, x):
        if x > 10 :
            self.__age = x
        else:
            self.__age = 2
raj = Getset(10)
print(raj.get_age())   # (10) retrieving age using getter before
                         setting
raj.set_age(21)        # setting the age using setter
print(raj.get_age())   # (21) retrieving age using getter
```

Using property () method

```
class Geeks:
    def __init__(self,age):
        self.__age = age
        print(self.__age)        # 1

    # function to get value of _age
    def get_age(self):
        print("getter method called")
        return self.__age

    # function to set value of _age
    def set_age(self, a):
        print("setter method called")
        self.__age = a
    a = property(get_age, set_age)


mark = Geeks(1)
mark.a = 20
print(mark.a)                   # 10
```

Accessing Attributes and Methods in Python

Attributes of a class can also be accessed using the following built-in methods and functions:

1. **getattr()** – This function is used to access the attribute of object.
2. **hasattr()** – This function is used to check if an attribute exist or not.
3. **setattr()** – This function is used to set an attribute. If the attribute does not exist, then it would be created.
4. **delattr()** – This function is used to delete an attribute. If you are accessing the attribute after deleting it raises error "class has no attribute".
5. The syntax of setattr() method is:

   setattr(object, name, value)

```
class emp:
    name = 'Harsh'
    salary = '25000'

    def show(self):
        print(self.name)
        print(self.salary)

e1 = emp()

# Use getattr instead of e1.name
print(getattr(e1, 'name'))     # Harsh

# returns true if object has attribute
print(hasattr(e1, 'name'))     # True

# sets an  attribute
setattr(e1, 'height', 152)

# returns the value of attribute name height
print(getattr(e1, 'height')) # 152

# delete the attribute
delattr(emp, 'salary')
```

PANDAS

Pandas deals with the following three data structures –

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, sizeimmutable.
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array.

1.**pandas.Series**

A pandas Series can be created using the following constructor –

```
 pandas.Series( data, index, dtype, copy)
```

```
# Using list
data = np.array(['a','b','c','d'])

s = pd.Series(data,index=[100,101,102,103])
print (s)
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print (s)

# using scalar
s = pd.Series(5, index=[0, 1, 2, 3])
print(s)     # 4 time 5

# retrieve the first element
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print s[0]   # 1
print s['a'] # 1 using Label
```

2.pandas.DataFrame

A pandas DataFrame can be created using the following constructor -

pandas.DataFrame(data, index, columns, dtype, copy)

Create DataFrame

A pandas DataFrame can be created using various inputs like Lists,dict,Series,Numpy ndarrays and Another DataFrame

```
import pandas as pd

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'])

print(df)
```

Its output is as follows -

```
      Name      Age

0     Alex      10

1     Bob       12

2     Clarke    13
```

# different operations on data frame

```
import pandas as pd
import matplotlib.pyplot as plt
df =pd.DataFrame([[1,1,3,4],[5,1,7,8],[9,10,11,12],[14,15,16,17]]
,columns=['A','B','C','D'],index=['row1','row2','row3','row4'])
print(df)

# Data frame
         A    B    C    D
row1     1    2    3    4
row2     5    6    7    8
row3     1   10   11   12
row4    14   15   16   17

df = df.reindex(['a', 'b', 'c', 'd’])
df['A']        # Selecting single column

pd.A           # Selecting single column

df[['A','C']] # Selecting TWO column

df['E']=df['A']+100  # new column E addition

drop_col=df.drop('B',axis=1) # axis=1 for clumn drop and axis=0 for
                               row drop

drop_row = df.drop('row1', axis=0) # because numpy shape of 2d array
always (0,1) row with ZERO index and column with ONE index
```

```
sel_row= df.loc['row2']            # selecting specific rows by name
sel_mutliple_row=df.loc[['row1','row2']] # selecting multiple rows
sel_perticular_value = df.loc['row3','B']  # 10 output
sel_row_column = df.loc[['row1','row2'],['A','E']]  # Selecting rows
                                                     and columns

sel_row_index = df.iloc[1] # selecting row with specific index
print('hi',df.iloc[:,1].values) # values used to exclude index
colums from data frame
print(df.iloc[:,:0]) # indexing start from dataframe index columns
d=df[df>11]      # value greater then 11 and NA for all other

d.dropna()       # drop all NA included rows because default axis=0
d.dropna(axis=1)# drop all NA included columns because axis=1
d.dropna(thresh=2)          # row having minimum 2 non NA entry

d.fillna(value='Alert')     # fill NA with Alert
```

Fill NA Forward and Backward

```
df.fillna(method='pad') # fill with previous row value
df.fillna(method='backfill') # fill with next row value

df['A'].unique()            # return unique value of specific column
df['A'].nunique()           # length of unique values

S=df[df['A']<10]            # drop as per condition in data frame
df.pop('two')
df.append(df2)
df['B'].value_counts()   # total repeated values in specific columns

# creating table as we want index and columns
f=df.pivot_table(values='A',index=['C','D'],columns='B')
df.read_csv('abc.csv')
# While converting to csv index should be false else it will create
index as column
df.to_csv('abc',index=False)
df.columns           # list the  columns
len(df.index)        # total rows in data frame

df.info()            # number of columns and rows/number of entry in
                       each row.
df.describe()        # detail min,max ,mean,std etc
```

```
df.shape   # size of dataframe (row.col)
df.size    # Total count values(ex.(2,5)=10)
df.head(2) # First two rows from datafarme
df.tail(2) # Last Two rows
df.T       # Transpose
df.ndim    # array dimension
df.sum()
```

count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
abs()	Absolute Value
prod()	Product of Values

Pandas sorting

There are two kinds of sorting available in Pandas. They are -

By label

By Actual Value

By Label

```
df.sort_index()
df.sort_index(ascending=False)
df.sort_values(by='col1') # col1 values are sorted and the respective
                  col2 value and row index will alter along with col1.
```

Pandas Apply,applymap,pipe

Table wise Function Application: pipe()

Row or Column Wise Function Application: apply()

Element wise Function Application: applymap()

```
f=lambda x:x+2
df=pd.DataFrame({'1':[100,2,3,4,5],'2':[200,5,6,7,8]})
print(df.apply(f))
t=df['A'].apply(lambda x:x*2)   # apply function or values
```

Drop_duplicates

# Drop duplicate in specified column (means duplicate entire row will have removed in that columns)

```
df_concat.drop_duplicates('name')
```

Pandas Operation on string

```
s = pd.Series(['Tom', 'William Rick', 'John','Alber@t','SteveSmith'])
print(s.str.lower())
s.str.upper()
s.str.len()
s.str.findall('e')
s.str.swapcase() # TOM --> MOT
```

cat(sep=pattern)

```
s = pd.Series(['Tom ', ' William Rick', 'John', 'Alber@t'])
print s.str.cat(sep='_')
```

Its output is as follows -

```
Tom _ William Rick_John_Alber@t
```

replace(a,b)

```
1)s = pd.Series(['Tom ', ' William Rick', 'John', 'Alber@t'])

      print ("After replacing @ with $:")

      print s.str.replace('@','$')

      After replacing @ with $:

      0   Tom

      1   William Rick

      2   John

      3   Alber$t
```

```
2)  replace all eg:1 values to 1000 in data frame
       df.replace(to_replace=1,value=1000))
```

Cancat

```
one = pd.DataFrame({

   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

   'subject_id':['sub1','sub2','sub4','sub6','sub5'],

   'Marks_scored':[98,90,87,69,78]},index=[1,2,3,4,5])

two = pd.DataFrame({

   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],

   'subject_id':['sub2','sub4','sub3','sub6','sub5'],

   'Marks_scored':[89,80,79,97,88]},index=[1,2,3,4,5])
```

print pd.concat([one,two])

Its output is as follows -

```
    Marks_scored     Name   subject_id

1             98     Alex         sub1

2             90      Amy         sub2

3             87    Allen         sub4

4             69    Alice         sub6

5             78   Ayoung         sub5

1             89    Billy         sub2

2             80    Brian         sub4

3             79     Bran         sub3

4             97    Bryce         sub6

5             88    Betty         sub5
```

```
pd.concat([one,two],keys=['x','y'])
```

Its output is as follows -

```
x  1  98    Alex    sub1
   2  90    Amy     sub2
   3  87    Allen   sub4
   4  69    Alice   sub6
   5  78    Ayoung  sub5
y  1  89    Billy   sub2
   2  80    Brian   sub4
   3  79    Bran    sub3
   4  97    Bryce   sub6
   5  88    Betty   sub5
```

```
pd.concat([one,two],keys=['x','y'],ignore_index=True)
```

Its output is as follows -

```
    Marks_scored     Name    subject_id
0             98     Alex          sub1
1             90      Amy          sub2
2             87    Allen          sub4
3             69    Alice          sub6
4             78   Ayoung          sub5
5             89    Billy          sub2
6             80    Brian          sub4
7             79     Bran          sub3
8             97    Bryce          sub6
9             88    Betty          sub5
```

Merge: pd.merge(left,right,on='id')

Merge Method	SQL Equivalent	Description
left	LEFT OUTER JOIN	Use keys from left object
right	RIGHT OUTER JOIN	Use keys from right object
outer	FULL OUTER JOIN	Use union of keys
inner	INNER JOIN	Use intersection of keys

```
pd.merge(left, right, on='subject_id', how='left')
```

```
# pandas with visualization
df['A'].hist()
df.plot.area()
df.plot.bar()
df.plot.scatter(x='A',y='B',c='red',figsize=[8,2])
df.plot.box()
df.plot.kde()
plt.show()
```