Python Notes

# Python Class or Static Attributes

1. What is Python class attributes?
   * All variables which are assigned a value in class declaration are class variables.
   * And variables which are assigned values inside method are instance variables.
   * For class variables, all objects a single copy maintained at class level.
   * There is no difference between python class and static variables, both are same.
2. How can we able to create and access class attributes?
   * Inside Class
     1. Inside class directly
     2. Inside constructor by using class name
     3. Inside instance method by using class name
     4. Inside class method by using cls variable or class name
     5. Inside static method by using class name
   * Outside Class
     1. From outside of class by using class name

**class** StaticDemo:  
 *# 1. Inside class directly* a = 10  
  
 *# 2. Inside constructor by using class name* **def** \_\_init\_\_(self):  
 StaticDemo.b = 20  
  
 *# 3. Inside instance method by using class name* **def** m1(self):  
 StaticDemo.c = 30  
  
 *# 4. Inside class method by using cls variable or class name* @classmethod  
 **def** m2(cls):  
 *# 4a. using cls variable* cls.d = 40  
 *# 4b. Using class name* StaticDemo.e = 50  
  
 *# 5. Inside static method by using class name* @staticmethod  
 **def** m3():  
 StaticDemo.f = 60  
  
  
*# 6. From outside of class by using class name*StaticDemo.g = 70  
  
object\_1 = StaticDemo()  
object\_1.m1()  
*# StaticDemo.m2()*object\_1.m2()  
*# StaticDemo.m3()*object\_1.m3()  
print(**'Class Level :'**, StaticDemo.\_\_dict\_\_)  
print(**'\*'** \* 50)  
print(**'Object Level :'**, object\_1.\_\_dict\_\_)

1. How can we able to modify class attributes?
   * Inside Class
     1. Inside Constructor using Class name
     2. Inside Class method using Class name or cls variable
     3. Inside Static method using Class name
   * Outside Class
     1. Outside of class by using class name

**class** StaticDemo:  
 a = 10  
  
 *# 1. Inside Constructor using Class name* **def** \_\_init\_\_(self):  
 StaticDemo.a = 100  
 *# Important topic* self.a = 101  
  
 *# 2. Inside Class method using Class name or cls variable* @classmethod  
 **def** m1(cls):  
 *# 2a. using class name* StaticDemo.a = 200  
 *# 2b. using cls variable* cls.a = 300  
  
 *# 3. Inside Static method using Class name* @staticmethod  
 **def** m2():  
 StaticDemo.a = 400  
  
  
object\_1 = StaticDemo()  
*# StaticDemo.m1()*object\_1.m1()  
*# StaticDemo.m2()*object\_1.m2()  
  
*# 4. Outside of class by using class name*StaticDemo.a = 500  
print(**'Class Level :'**, StaticDemo.\_\_dict\_\_)  
print(**'Object Level:'**, object\_1.\_\_dict\_\_)

1. How can we able to delete class attributes?
   * Inside Class
     1. Inside Constructor using Class name
     2. Inside Class method using Class name or cls variable
     3. Inside Static method using Class name
   * Outside Class
     1. Outside of class by using class name

**class** StaticDemo:  
 a = 10  
 b = 20  
 c = 30  
 d = 40  
 e = 50  
 f = 60  
  
 *# 1. Inside Constructor using Class name* **def** \_\_init\_\_(self):  
 **del** StaticDemo.a  
  
 *# 2. Inside Class method using Class name or cls variable* @classmethod  
 **def** m1(cls):  
 *# 2a. Using class name* **del** StaticDemo.b  
 *# 2b. Using cls variable* **del** cls.c  
  
 *# 3. Inside Static method using Class name* @staticmethod  
 **def** m2():  
 **del** StaticDemo.d  
  
  
print(**'Before object creation :'**, StaticDemo.\_\_dict\_\_)  
object\_1 = StaticDemo()  
print(**'After object creation :'**, StaticDemo.\_\_dict\_\_)  
StaticDemo.m1()  
print(**'After calling m1() method :'**, StaticDemo.\_\_dict\_\_)  
StaticDemo.m2()  
print(**'After calling m2() method :'**, StaticDemo.\_\_dict\_\_)  
  
*# 4. Outside of class by using class name***del** StaticDemo.e  
print(**'Outside class del :'**, StaticDemo.\_\_dict\_\_)

# Getters and Setters

1. What is Python Getters and Setters and Property?
   * Getters are used for retrieving the data. Also known as ‘Accessors’.
   * Setters are used for changing the data. Also known as ‘Mutators’.
   * Property is Pythonic way to implement Getters and Setters.
2. Why do we need to use Getters and Setters?
   * The main purpose of getters and setters are Data Encapsulation. To avoid direct access of variables.
   * Adding validation logic while setting and getting variables.
3. How to implement Getters and Setters?
   1. Using normal functions / methods.

*# 1. Using normal functions / methods***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
  
 *# Getter method* **def** get\_a(self):  
 **return** self.\_\_a  
  
 *# Setter method* **def** set\_a(self, a):  
 self.\_\_a = a  
  
  
object\_1 = DataEncapsulation()  
print(**'Get value of a:'**, object\_1.get\_a())  
object\_1.set\_a(20)  
print(**'Get value of a:'**, object\_1.get\_a())

1. How to implement Property?
2. Using python Property() function

A property has 3 methods,

getter(),

setter() and

delete().

And has four arguments property(fget=None, fset=None, fdel=None, doc=None),

fget is a function for retrieving an attribute value,

fset is a function for setting an attribute value.

fdel is a function for deleting an attribute value.

doc creates a docstring for attribute.

*# 2. Python property() function***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
 self.\_\_b = 20  
  
 *# Getter method* **def** get\_a(self):  
 print(**'Inside get\_a()'**)  
 **return** self.\_\_a  
  
 *# Setter method* **def** set\_a(self, a):  
 print(**'Inside set\_a()'**)  
 **if** a <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_a = 1  
 **else**:  
 self.\_\_a = a  
  
 **def** del\_a(self):  
 print(**'Inside del\_a()'**)  
 **del** self.\_\_a  
  
 a = property(get\_a, set\_a, del\_a, **'Property function demo'**)  
  
  
object\_1 = DataEncapsulation()  
print(object\_1.\_\_dict\_\_)  
print(**'Get value of a:'**, object\_1.a)  
object\_1.a = -20  
print(**'Get value of a:'**, object\_1.a)  
**del** object\_1.a  
print(object\_1.\_\_dict\_\_)

1. Using @property decorators

Python @property is one of the built-in decorators. The main purpose of any decorator is to change your class methods or attributes in such a way so that the user of your class no need to make any change in their code.

*# 3. Using @property decorators***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
  
 *# Getter method* @property  
 **def** a(self):  
 print(**'Inside Getter method'**)  
 **return** self.\_\_a  
  
 *# Setter method* @a.setter  
 **def** a(self, a):  
 print(**'Inside Setter method'**)  
 **if** a <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_a = 1  
 **else**:  
 self.\_\_a = a  
  
 @a.deleter  
 **def** a(self):  
 print(**'Inside Deleter method'**)  
 **del** self.\_\_a  
  
  
object\_1 = DataEncapsulation()  
print(object\_1.\_\_dict\_\_)  
print(**'Get value of a:'**, object\_1.a)  
object\_1.a = -20  
print(**'Get value of a:'**, object\_1.a)  
**del** object\_1.a  
print(object\_1.\_\_dict\_\_)

Comparing Getters and Setters v/s Property() function and Property decorators

*# 1. Using normal functions / methods***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
 self.\_\_b = 20  
  
 *# Getter method* **def** get\_a(self):  
 **return** self.\_\_a  
  
 *# Setter method* **def** set\_a(self, a):  
 self.\_\_a = a  
  
 *# Getter method* **def** get\_b(self):  
 **return** self.\_\_b  
  
 *# Setter method* **def** set\_b(self, b):  
 self.\_\_b = b  
  
  
object\_1 = DataEncapsulation()  
 *# Perform addition*object\_1.set\_b(object\_1.get\_b() + object\_1.get\_a())  
print(object\_1.get\_b())

*# 2. Python property() function***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
 self.\_\_b = 20  
  
 *# Getter method* **def** get\_a(self):  
 print(**'Inside get\_a()'**)  
 **return** self.\_\_a  
  
 *# Setter method* **def** set\_a(self, a):  
 print(**'Inside set\_a()'**)  
 **if** a <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_a = 1  
 **else**:  
 self.\_\_a = a  
  
 **def** del\_a(self):  
 print(**'Inside del\_a()'**)  
 **del** self.\_\_a  
  
 a = property(get\_a, set\_a, del\_a, **'Property function demo'**)  
  
 *# Getter method* **def** get\_b(self):  
 print(**'Inside get\_b()'**)  
 **return** self.\_\_b  
  
 *# Setter method* **def** set\_b(self, b):  
 print(**'Inside set\_b()'**)  
 **if** b <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_b = 1  
 **else**:  
 self.\_\_b = b  
  
 **def** del\_b(self):  
 print(**'Inside del\_b()'**)  
 **del** self.\_\_b  
  
 b = property(get\_b, set\_b, del\_b, **'Property function demo'**)  
  
  
object\_1 = DataEncapsulation()  
 *# Perform addition*object\_1.b = object\_1.a + object\_1.b  
print(object\_1.b)

*# 3. Using @property decorators***class** DataEncapsulation:  
  
 **def** \_\_init\_\_(self):  
 *# private variable* self.\_\_a = 10  
 self.\_\_b = 20  
  
 *# Getter method* @property  
 **def** a(self):  
 print(**'Inside Getter method'**)  
 **return** self.\_\_a  
  
 *# Setter method* @a.setter  
 **def** a(self, a):  
 print(**'Inside Setter method'**)  
 **if** a <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_a = 1  
 **else**:  
 self.\_\_a = a  
  
 @a.deleter  
 **def** a(self):  
 print(**'Inside Deleter method'**)  
 **del** self.\_\_a  
  
 *# Getter method* @property  
 **def** b(self):  
 print(**'Inside Getter method'**)  
 **return** self.\_\_b  
  
 *# Setter method* @a.setter  
 **def** b(self, b):  
 print(**'Inside Setter method'**)  
 **if** b <= 0:  
 print(**'Negative value is set to default value.'**)  
 self.\_\_b = 1  
 **else**:  
 self.\_\_b = b  
  
 @a.deleter  
 **def** b(self):  
 print(**'Inside Deleter method'**)  
 **del** self.\_\_b  
  
  
object\_1 = DataEncapsulation()  
 *# Perform addition*object\_1.b = object\_1.a + object\_1.b  
print(object\_1.b)

The attributes of a class are made private to hide and protect them from other code.

Note: When you create private attributes the set getters and setters to public

# Programs

## Search String Recursively from Directories

**import** os  
  
  
**def** return\_all\_files(directory\_path, file\_types):  
 *"""  
 Method to return all the files, which matches the file type extension* **:param** *directory\_path: directory/folder location for searching files.  
 type: string* **:param** *file\_types: Type of file to perform search.  
 type: string  
 example: .txt, .log, .xml etc..* **:return** *tuple  
 """* **for** root, directories, files **in** os.walk(directory\_path):  
 **for** file **in** files:  
 **if** file.endswith(file\_types):  
 **yield** os.path.join(root, file)  
  
  
**def** return\_all\_files\_with\_search\_text(result\_files, search\_text):  
 *"""  
 Search all the files and if search\_text is present then write into 'Tech\_Made\_Me\_Lazy.log' file  
 If any error found then skip the file and log the error.* **:param** *result\_files: all the files to perform search operation  
 type: tuple* **:param** *search\_text: text to find in the files  
 type: string* **:return***: Writes down all logs to file Tech\_Made\_Me\_Lazy.log'  
 """* output\_file = open(**'Tech\_Made\_Me\_Lazy.log'**, **'w'**)  
 **for** file **in** result\_files:  
 **try**:  
 **with** open(file) **as** f:  
 **if** search\_text **in** f.read():  
 output\_file.write(**'Found text in :'** + file + **'\n'**)  
 **except** Exception **as** error:  
 output\_file.write(**'Skipping file:'** + file + **'\n'**)  
 output\_file.write(str(error) + **'\n'**)  
  
  
directory\_path = input(**'Enter directory or folder location to search : '**).strip()  
print(**'Directory :'**, directory\_path)  
  
file\_types = tuple(str(x).strip() **for** x **in** input(**'Enter file types to search :'**).split(**','**))  
print(**'File Types :'**, file\_types)  
  
search\_text = input(**'Enter text to search :'**)  
print(**'Search text :'**, search\_text)  
  
result\_files = [f **for** f **in** return\_all\_files(directory\_path, file\_types)]  
print(**'Total Number of files found :'**, result\_files.\_\_len\_\_())  
  
return\_all\_files\_with\_search\_text(result\_files, search\_text)  
  
*# --------------------------------------  
# How to run  
# C:\Tech\_Made\_Me\_Lazy\Python\Tech\_Made\_Me\_Lazy\_Python\programs>python search\_text.py  
# Enter directory or folder location to search : C:\Users\sdad\Downloads\Project Documents  
# Directory : C:\Users\sdad\Downloads\Project Documents  
# Enter file types to search :.txt, .log  
# File Types : ('.txt', '.log')  
# Enter text to search :the  
# Search text : the  
# Total Number of files found : 159*