(7.2) Variables and Method

1. Types of Variables

1. Instance Variables (Object Level Variables)
2. Static Variables (Class Level Variables)
3. Local variables (Method Level Variables)

1. Instance Variables

If the value of a variable is varied from object to object, then such type of variables are called instance variables.
For every object a separate copy of instance variables will be created.

i). Where we can declare Instance variables

1. Inside Constructor by using self variable
2. Inside Instance Method by using self variable
3. Outside of the class by using object reference variable

Inside Constructor by using self variable

We can declare instance variables inside a constructor by using self keyword. Once we creates object, automatically these variables will be added to the object

In [2]:
```
class Employee:
    def __init__(self):
        self.eno=100
        self.ename='Pankaj'
        self.wsal=10000

e = Employee()
print(e.__dict__)
```

```
{'eno': 100, 'ename': 'Pankaj', 'wsal': 10000}
```

Inside Instance Method by using self variable

We can also declare instance variables inside instance method by using self variable. If any instance variable declared inside instance method, that instance variable will be added once we call taht method.

In [3]:
```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20
    def m1(self):
        self.c = 30

t = Test()
print(t.__dict__)
t.m1()
print(t.__dict__)
```

```
{'a': 10, 'b': 20}
```

```
{'a': 10, 'b': 20, 'c': 30}
```

Outside of the class by using object reference variable

We can also add instance variables outside of a class to a particular object

In [4]:
```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20
    def m1(self):
        self.c = 30

t =Test()
t.m1()

t.d = 40
print(t.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

ii) How to access Instance variables

We can access instance variables with in the class by using self variable and outside of the class by using object reference.

In [5]:
```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20
    def display(self):
        print(self.a, self.b)

t = Test()
t.display()
print(t.a,t.b)
```

```
10 20
10 20
```

iii) How to delete instance variable from the object

1. Within a class we can delete instance variable as follows

del self.variableName

2. From outside of class we can delete instance variables as follows

del objectreference.variableName

In [7]:
```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20
        self.c = 30
    def m1(self):
        del self.c

t = Test()
print(t.__dict__)
```

```
t.m1()
print(t.__dict__)
del t.b
print(t.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30}
{'a': 10, 'b': 20}
{'a': 10}
```

Note:

The instance variables which are deleted from one object,will not be deleted from other objects.

If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects, because for every object we are separate copy of instance variables are available.

2. Static Variable

If the value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly but outside of methods. Such type of variables are called Static variables.

For total class only one copy of static variable will be created and shared by all objects of that class.

We can access static variables either by class name or by object reference. But recommended to use class name.

i) Various places to declare static variables

1. In general we can declare within the class directly but from out side of any method
2. Inside constructor by using class name
3. Inside instance method by using class name
4. Inside classmethod by using either class name or cls variable
5. Inside static method by using class name

In [49]:

```
class Test:
    a = 10    #static
    def __init__(self):
        Test.b = 20    #static
    def m1(self):
        Test.c = 30
    @classmethod
    def m2(cls):
        cls.d1 = 40
        Test.d2 = 50
    @staticmethod
    def m3():
        Test.e=50


print(Test.__dict__)

t=Test()
print(Test.__dict__)

t.m1()
print(Test.__dict__)
```

```
t.m2()
print(Test.__dict__)

Test.m3()
print(Test.__dict__)

Test.f=60
print(Test.__dict__)
```

```
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None}
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None, 'b': 20}
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None, 'b': 20, 'c': 30}
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None, 'b': 20, 'c': 30, 'd1': 40, 'd2': 50}
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None, 'b': 20, 'c': 30, 'd1': 40, 'd2': 50, 'e': 50}
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
D30>, 'm1': <function Test.m1 at 0x0000025FEF4A9CA0>, 'm2': <classmethod object at 0x00000
25FEF848850>, 'm3': <staticmethod object at 0x0000025FEF8482B0>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None, 'b': 20, 'c': 30, 'd1': 40, 'd2': 50, 'e': 50, 'f': 60}
```

ii) How to access static variable

1. inside constructor: by using either self or classname
2. inside instance method: by using either self or classname
3. inside class method: by using either cls variable or classname
4. inside static method: by using classname
5. From outside of class: by using either object reference or classnmae

In [41]:

```
class Test:
    a = 10
    def __init__(self):
        print(self.a)
        print(Test.a)
    def m1(self):
        print(self.a)
        print(Test.a)
    @classmethod
    def m2(cls):
        print(cls.a)
        print(Test.a)
    @staticmethod
    def m3():
        print(Test.a)
```

```
t=Test()
print(Test.a)
print(t.a)
t.m1()
t.m2()
t.m3()
```

```
10
10
10
10
10
10
10
10
10
```

iii) where we can modify the value of static variable

Anywhere either with in the class or outside of class we can modify by using classname.But inside class method, by using cls variable.

Message Passing

we use class variables to make a communication between two objects of same class

In [43]:
```
class Test:
    a=777
    def __init__(self):
        Test.a = 999
    def m1(self):
        Test.a = 111
    @classmethod
    def m2(cls):
        cls.a = 222
    @staticmethod
    def m3():
        Test.a = 333

print(Test.a)
t = Test()
print(Test.a)
t.m1()
print(Test.a)
t.m2()
print(Test.a)
t.m3()
print(Test.a)
```

```
777
999
111
222
333
```

If we change the value of static variable by using either self or object reference variable, then the value of static variable won't be changed,just a new instance variable with that name will be added to that particular object.

In [44]:
```
class Test:
    a = 10
```

```
    def m1(self):
        self.a = 888

t1 = Test()
t1.m1()
print(Test.a)
print(t1.a)
```

```
10
888
```

In [47]:

```
class Test:
    a = 10

t1 = Test()
t2 = Test()

t1.a = 999
print(Test.a)
print(t1.a)
print(t2.a)
```

```
10
999
10
```

iv) How to delete static variable of a class

We can delete static variables from anywhere by using the following syntax

del classname.variablename

But inside classmethod we can also use cls variable

del cls.variablename

In [48]:

```
class Test:
    a=10
    def __init__(self):
        Test.b=20
        del Test.a
    def m1(self):
        Test.c=30
        del Test.b
    @classmethod
    def m2(cls):
        cls.d=40
        del Test.c
    @staticmethod
    def m3():
        Test.e=50
        del Test.d


print(Test.__dict__)
t=Test()
print(Test.__dict__)
t.m1()
print(Test.__dict__)
Test.m2()
print(Test.__dict__)
```

```
Test.m3()
print(Test.__dict__)
Test.f=60
print(Test.__dict__)
del Test.e
print(Test.__dict__)
```

```
{'__module__': '__main__', 'a': 10, '__init__': <function Test.__init__ at 0x0000025FEF4A9
790>, 'm1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x00000
25FEF848EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '_
_dict__' of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>,
'__doc__': None}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'b': 20}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'c': 30}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'd': 40}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'e': 50}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'e': 50, 'f': 60}
{'__module__': '__main__', '__init__': <function Test.__init__ at 0x0000025FEF4A9790>, 'm
1': <function Test.m1 at 0x0000025FEF4A9820>, 'm2': <classmethod object at 0x0000025FEF848
EE0>, 'm3': <staticmethod object at 0x0000025FEF848F10>, '__dict__': <attribute '__dict__'
of 'Test' objects>, '__weakref__': <attribute '__weakref__' of 'Test' objects>, '__doc__':
None, 'f': 60}
```

Note: By using object reference variable/self we cannot delete static variable

In [50]:

```
class Test:
    a = 10

t1 = Test()
del t1.a
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_8280/285171195.py in <module>
      3
      4 t1 = Test()
----> 5 del t1.a

AttributeError: a
```

3. Local Variable

Sometimes to meet temporary requirements of programmer,we can declare variables inside a method directly,such type of variables are called local variable or temporary variables.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of method.

In [51]:
```
class Test:
    def m1(self):
        a = 1000
        print(a)

t = Test()
t.m1()
```

```
1000
```

In [52]:
```
t.a
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_8280/1554173408.py in <module>
----> 1 t.a

AttributeError: 'Test' object has no attribute 'a'
```

2. Types of Methods

Class Methods, Instance Methods, Static Methods

Bounded & Unbounded Methods

1. Instance Methods (Bounded to intsance)

Inside method implementation if we are using instance variables then such type of methods are called instance methods.

Inside instance method declaration,we have to pass self variable.

Within the class we can call instance method by using self variable and from outside of the class we can call by using object reference.

In [68]:
```
class A:
    #-----Instance Methods----Bounded to a Instance Scope----
    def __init__(self):
        self.name = "A's Object"
    def get_name(self):
        print(self.name)
    def set_name(self, name):
        self.name = name
    def display(self):
        self.get_name()  #we are calling instance method using self
    def __str__(self):
        return self.name
```

```
x = A()
y = A()
x.get_name()  # self -> x (bounded)
y.get_name()  # self -> y (bounded)
x.display()   # calling instance method using object reference
```

```
A's Object
A's Object
A's Object
```

In [69]:

```
print(x.__dict__)
print(y.__dict__)
```

```
{'name': "A's Object"}
{'name': "A's Object"}
```

Note: Instance method can only be called using an instance, bcz they are binded to a instance scope

In [70]:

```
A.display()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_8280/1262822627.py in <module>
----> 1 A.display()

TypeError: display() missing 1 required positional argument: 'self'
```

This will pass with instance x

In [71]:

```
A.display(x) # x.get_name(self=x, ...)
```

```
A's Object
```

Every instance have different scope

In [72]:

```
x = A() # x is an intsance of class A
y = A() # y is an instance of class A

print(id(x), id(y), sep='\n')
```

```
2611062026192
2611062024128
```

for class method is created only once when class is created

In [73]:

```
import time
i = 1
while i <= 5:
    print(A.get_name, id(A.get_name), flush=True)
    time.sleep(1)
    i += 1
```

```
<function A.get_name at 0x0000025FEF4A9820> 2611059791904
<function A.get_name at 0x0000025FEF4A9820> 2611059791904
<function A.get_name at 0x0000025FEF4A9820> 2611059791904
<function A.get_name at 0x0000025FEF4A9820> 2611059791904
<function A.get_name at 0x0000025FEF4A9820> 2611059791904
```

Whenever we call instance method it will create everytime

In [74]:

```
i = 1
```

```
while i <= 5:
    print(x.get_name, id(x.get_name), flush=True)
    time.sleep(1)
    i += 1
```

```
<bound method A.get_name of <__main__.A object at 0x0000025FEF6CAFD0>> 2611026562752
<bound method A.get_name of <__main__.A object at 0x0000025FEF6CAFD0>> 2611026560320
<bound method A.get_name of <__main__.A object at 0x0000025FEF6CAFD0>> 2611026505216
<bound method A.get_name of <__main__.A object at 0x0000025FEF6CAFD0>> 2611026560320
<bound method A.get_name of <__main__.A object at 0x0000025FEF6CAFD0>> 2611026505216
```

by defalut all methods are instance methods

2. Class Methods(Bounded to class)

Inside method implementation if we are using only class variables (static variables), then such type of methods we should declare as class method.

We can declare class method explicitly by using @classmethod decorator.

For class method we should provide cls variable at the time of declaration

In [76]:
```
class A:
    msg = "Dashboard\n" # class variables
    #----------Instance Methods----------
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return self.name
    def get_name(self):
        return self.name
    def set_name(self, new_name):
        self.name = new_name
    #-----Class Methods-----Bounded in class Scope
    @classmethod
    def get_msg(cls):
        print(cls.msg)
    @classmethod
    def set_msg(cls, msg):
        cls.msg = msg
```

In [77]:
```
a = A('rajat')
b = A('sachin')
```

In [78]:
```
a.get_msg()
```

```
Dashboard
```

In [79]:
```
a.set_msg('now you can see me from any where')
```

In [80]:
```
b.get_msg()
```

```
now you can see me from any where
```

We can call a class Method without any object

In [81]:

```
A.get_msg()
```

```
now you can see me from any where
```

Program to track the number of object created for a class

In [84]:
```
class Test:
    count = 0
    def __init__(self):
        Test.count = Test.count+1
    @classmethod
    def noOfObject(cls):
        print('The number of object are : ',cls.count)

t1 = Test()
t2 = Test()
Test.noOfObject()
t3 = Test()
t4 = Test()
t5 = Test()
Test.noOfObject()
```

```
The number of object are :  2
The number of object are :  5
```

3. Static Methods(Unbounded Simple Python Methods)

In general these methods are general utility methods.
Inside these methods we won't use any instance or class variables.
Here we won't provide self or cls arguments at the time of declaration.
We can declare static method explicitly by using @staticmethod decorator
We can access static methods by using classname or object reference

In [85]:
```
class A:
    msg = "Dashboard\n" # class variables
    #----------Instance Methods----------
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return self.name
    def get_name(self):
        return self.name
    def set_name(self, new_name):
        self.name = new_name
    #-----Class Methods-----Bounded in class Scope
    @classmethod
    def get_msg(cls):
        print(cls.msg)
    @classmethod
    def set_msg(cls, msg):
        cls.msg = msg
    #static methods are just normal function inside class as method
    @staticmethod
    def author():
        """Unbounded Methods"""
        print("Written by Pankaj Yadav")
    @staticmethod
    def pattern(n_rows):
```

```
        for row in range(1, n_rows+1):
            print("*"*row)
```

In [86]:
```
A.author()
```

```
Written by Pankaj Yadav
```

In [88]:
```
a = A('Pankaj')
a.author()
```

```
Written by Pankaj Yadav
```

In [89]:
```
a.pattern(10)
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Note

we use instance methods (default) to access and manipluate properties of an Instance / object
that's why every instance method is bounded to a instance scope

we use class methods (classmethod) to access and manipluate variable of an Class
that's why every class method is bounded to a class scope

we use static to provide some general information which does not depend on class attribures or instance attributes
that's why every static method is unbouded method or a simple python function

3. Setter and Getter Methods

We can set and get the values of instance variables by using getter and setter methods.

Setter Method

setter methods can be used to set values to the instance variables.
setter methods also known as mutator methods.

Getter Method

Getter methods can be used to get values of the instance variables. Getter methods also known as accessor methods.

In [75]:
```
class A:
    def __init__(self):
        self.name = "A's Object"
    def get_name(self):
        print(self.name)
    def set_name(self, name):
        self.name = name
    def display(self):
        self.get_name()
    def __str__(self):
        return self.name


x = A()
x.get_name()
x.set_name('Pankaj')
x.display()
```

```
A's Object
Pankaj
```

4. Passing Member of one class to another class

In [90]:
```
class Employee:
    def __init__(self,eno,ename,esal):
        self.eno=eno
        self.ename=ename
        self.esal=esal
    def display(self):
        print('Employee Number:',self.eno)
        print('Employee Name:',self.ename)
        print('Employee Salary:',self.esal)

class Test:
    def modify(emp):
        emp.esal=emp.esal+10000
        emp.display()

e=Employee(100,'Pankaj',10000)
Test.modify(e)
```

```
Employee Number: 100
Employee Name: Pankaj
Employee Salary: 20000
```

5. Inner Class

Sometimes we can declare a class inside another class,such type of classes are called inner classes.

Without existing one type of object if there is no chance of existing another type of object,then we should go for inner classes.

Example:

Without existing Car object there is no chance of existing Engine object. Hence Engine class should be part of Car class.

```
class Car:
```

```
.....
class Engine:
    ......
```

Example:

```
Without existing university object there is no chance of existing Department object

class University:
    .....
    class Department:
        ......
```

Note:

```
Without existing outer class object there is no chance of existing inner class
object. Hence inner class object is always associated with outer class object.
```

In [92]:
```
class Outer:
    def __init__(self):
        print("outer class object creation")
    class Inner:
        def __init__(self):
            print("inner class object creation")
        def m1(self):
            print("inner class method")


o=Outer()
i=o.Inner()
i.m1()
```

```
outer class object creation
inner class object creation
inner class method
```

In [94]:
```
i1 = Outer().Inner()
i.m1()
```

```
outer class object creation
inner class object creation
inner class method
```

In [95]:
```
Outer().Inner().m1()
```

```
outer class object creation
inner class object creation
inner class method
```

6. Problems

Question :- Chat Box

In [22]:
```
import time
```

In [23]:
```
time.ctime()
```

Out[23]:
```
'Thu Jan  6 18:39:59 2022'
```

In [26]:
```
import os
```

In [27]:
```
!rm chat.txt
```

```
'rm' is not recognized as an internal or external command,
operable program or batch file.
```

In [31]:
```
class Grras:
    if os.access('chat.txt', os.W_OK):
        with open('chat.txt') as file:
            msg = file.read()
    else:
        msg = "" # class variable
    #-------------Instance Methods---------------
    def __init__(self, name: str, courses: list):
        self.name = name.strip().title()
        self.courses = courses
    def __str__(self):
        return self.name
    def get_courses(self):
        return self.courses
    def set_courses(self, *args):
        for course in args:
            self.courses.append(courses)
    def set_msg(self, msg):
        msg = time.ctime() + f"\n{self.name}:" + msg
        self.update_dashboard(msg)
    #-------------Class Methods------------------
    @classmethod
    def update_file(cls, msg, fname='chat.txt'):
        with open(fname, 'a') as file:
            file.write(msg)
            file.close()
    @classmethod
    def show_dashboard(cls):
        print(cls.msg)
    @classmethod
    def update_dashboard(cls, msg):
        msg = f"\n\n{msg}"
        cls.update_file(msg)
        cls.msg += msg
    #-------------Static methods-------------------
    @staticmethod
    def version():
        return 'version 1.0.0'
    @staticmethod
    def author():
        return "Pankaj Yadav"
```

In [32]:
```
a = Grras('Sachin Yadav', [ 'Python', 'DS', 'ML', 'DL'])
b = Grras('Rajat Goyal', [ 'Linux', 'AWS', 'Ansible'])
```

In [34]:
```
print(a, b, sep='\n')
```

```
Sachin Yadav
Rajat Goyal
```

In [35]:
```
a.show_dashboard()
```

In [36]:
```
a.set_msg('Hello Guys! I will not be able to make it today... manage accordingly')
```

In [37]:
```
b.show_dashboard()
```

```
Thu Jan  6 18:56:19 2022
Sachin Yadav:Hello Guys! I will not be able to make it today... manage accordingly
```

In [38]:
```
b.set_msg("Okay Sachin! your batches will be off and we will manage! enjoy your vaction!")
```

In [39]:
```
a.show_dashboard()
```

```
Thu Jan  6 18:56:19 2022
Sachin Yadav:Hello Guys! I will not be able to make it today... manage accordingly

Thu Jan  6 18:56:42 2022
Rajat Goyal:Okay Sachin! your batches will be off and we will manage! enjoy your vaction!
```

In [40]:
```
!type chat.txt
```

```
Thu Jan  6 18:56:19 2022
Sachin Yadav:Hello Guys! I will not be able to make it today... manage accordingly

Thu Jan  6 18:56:42 2022
Rajat Goyal:Okay Sachin! your batches will be off and we will manage! enjoy your vaction!
```

In [41]:
```
a.set_msg("Ohh thats Great!")
```

In [42]:
```
a.show_dashboard()
```

```
Thu Jan  6 18:56:19 2022
Sachin Yadav:Hello Guys! I will not be able to make it today... manage accordingly

Thu Jan  6 18:56:42 2022
Rajat Goyal:Okay Sachin! your batches will be off and we will manage! enjoy your vaction!

Thu Jan  6 18:57:16 2022
Sachin Yadav:Ohh thats Great!
```

In [43]:
```
a.version()
```

Out[43]:
```
'version 1.0.0'
```

In [44]:
```
a.author()
```

Out[44]:
```
'Pankaj Yadav'
```

In [ ]: