Pair Programming Exercise Python Classes

DSE5002 Module 7, HD Sheets, July 2024

Classes are part of object oriented program

When we define a class, we are defining a structure that holds data in an organized or structured way. In addition to defining functions that carry out operations on that data, these are called "methods" in python and "member functions" in many other languages.

Once we have a class definition, we can create instances of the class to create variables.

{When we use numpy or pandas, we are creating instances of classes defined in the libraries numpy or pandas. In addition to data storage, there are many defined "methods" or "member functions" defined in numpy and pandas}

Classes are thus useful for carefully creating both storage and functions.

Object oriented programming makes constant use of classes. You only interact with a class through it's member functions, not by directly interacting with the variables within the class. This means you don't need to know the internal details within a class to use it reliably. You are also unlikely to screw up a class by using it incorrectly or altering it in ways you shouldn't, if you use the member functions

Member functions can be "overloaded", meaning that each class can have it's own version of commmonly used functions like "print" or "plot", which replace the default functions for that particular class only.

When you use the dir() function on a python varialbe, it is showing you the methods for the class

Defining a Class

We define a class by stating what variables are in it and what it's methods are

We will always want at least on member function, an "initialization" function that sets up the stored variables in the class when an "instance" of the class is created

{Note: when you create a Pandas data frame in Python, you are created an "instance" of the class pandas dataframe. It may not be obvious this is happening, for example if you load a csv as a pandas frame, you don't really see the creation (or "instance") of the data frame, you just have a variable of type dataframe}

In [2]:
```
class two_int:
    """two int, a simple example of a class"""

    # create an intialization function, which must be named __init__
    # the term self.   means a variable within the class
    # this class contains two integers, i and j
    #
    # note that the variable "self" must be sent into the init function, then the t
    # notice I did set default values for i and j

    def __init__(self,i=0,j=0):
        self.i=i
        self.j=j

    #lets make a member print function as well

    def print(self):
        print("i is :",self.i," j is :",self.j)
```

In [3]:
```
# when we set x to be two_int(1,2) this calls the initialization member function

x=two_int(i=1,j=2)

# to call the method "print",  we use x.print()

x.print()
```

```
i is : 1  j is : 2
```

In [4]:
```
y=two_int()
y.print()
```

```
i is : 0  j is : 0
```

Class and instance variable

In the example above, i and j are "instance variables", each instance of two_int will have indepdent values of i and j

We can add variables that are common to all instance of the class, which are called Class variables

Here we will add a class variable called "name"

I'm also going to add "set" functions, which allow us to set the value of i in the class after it is created. To change a value within the class requires some form of a "set" function, although in python it is usually not named set.

My set_i function checks to be sure that the range of the i value is valid, I only want i to range from 0 to 100, since I want I to be percentage value. This is an advantage of a set function, it can be used to ensure no one enters an invalid value

In [5]:
```
class two_int:
    """two int, a simple example of a class"""
    # add a class variable
    kind="two_int, percentage"


    # create an intialization function, which must be named __init__
    # the term self.   means a variable within the class
    # this class contains two integers, i and j
    #
    # note that the variable "self" must be sent into the init function, then the t
    # notice I did set default values for i and j

    def __init__(self,i=0,j=0):
        self.i=i
        self.j=j

    #lets make a member print function as well

    def print(self):
        print("i is :",self.i," j is :",self.j)

    #a set function, allows the value of i to be changed after the variables are cr
    def set_i(self,i):
        if( (i<0)|(i>100)):
            print("Invalid i value entered")
        else:
            self.i=i
```

In [6]:
```
x=two_int(0.2,2)

x.print()


# try to enter an invalid i value
x.set_i(455)

x.print()

#set a valid i value this time

x.set_i(30)

x.print()
```

```
i is : 0.2  j is : 2
Invalid i value entered
i is : 0.2  j is : 2
i is : 30  j is : 2
```

We can also create get functions that return the values, we'll add a get function for j

and we can add other times of functions as well

In [7]:
```
class two_int:
    """two int, a simple example of a class"""
    # add a class variable
    kind="two_int, percentage"


    # create an intialization function, which must be named __init__
    # the term self.   means a variable within the class
    # this class contains two integers, i and j
    #
    # note that the variable "self" must be sent into the init function, then the t
    # notice I did set default values for i and j

    def __init__(self,i=0,j=0):
        self.i=i
        self.j=j

    #lets make a member print function as well

    def print(self):
        print("i is :",self.i," j is :",self.j)

    #a set function, allows the value of i to be changed after the variables are cr
    def set_i(self,i):
        if( (i<0)|(i>100)):
            print("Invalid i value entered")
        else:
            self.i=i

   # here is a get function for j
    def get_j(self):
        return self.j

   # here is a function to return i percent of j
    def get_per(self):
        return self.i*self.j/100
```

In [8]:
```
x=two_int(3,12)

x.get_j()

x.get_per()
```

Out[8]:
```
0.36
```

Why

Defining a class and methods for it means that the class can be shared and re-used

The user of a class doesn't need to understand or work with the internal structure of the class, only with the methods

If the internal structure needs to be changed or updated, or new methods added, existing code using the class won't break, as long as the inputs and outputs and names of the member functions (methods) are not altered.

This is a huge reliability issue when working with a team, particularly a large one

Many libraries are mostly sets of class definitions, that turns out to be key too working with data, carefully designed classes

Question/Action

Write a function called customer than contains a first name and a last name as instance variables

It should have an **init** function that has input self, first and last that stores first and last

Also create a print function that prints out the person's name

In [11]:
```
class Customer:
    def __init__(self, first, last):
        self.first=first
        self.last=last
    def __str__(self):
        return f"{self.first} {self.last}"
```

In [12]:
```
cust1 = Customer("John", "Doe")
print(cust1)
```

```
John Doe
```