National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

CS250: Data Structures and Algorithm

Class: BS Mathematics - 2021

Lab 02:
Introduction to Object Oriented Programming in Python

Classes and Objects

Date: 20th September, 2023

Time: 10:00 am - 01:00 pm

Instructor: Fauzia Ehsan

National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

Lab 02: Introduction to Object Oriented Programming System OOPS in Python

Introduction

The purpose of this lab is to have a brief introduction of Object-Oriented Programming by way of Python'sclasses.

.

Tools/Software Requirement
Python 3/ PyCharm IDE / VS Code

Contents

1 Classes and Objects

In Python, classes are a way to group together functions and attributes that are closely related to one another. A `class` is defined similarly to a function, though it uses the `class` keyword rather than `def`.

```
>>> class Test:
...     i = 5
...     def hi():
...         print('Hello world.')
...
>>> Test.i
5
>>> Test.hi()
Hello world.
```

Once a class has been declared, an *Object* or *Instance* of that class can be created. An object is an independent copy of the class, with all of the same attributes and functions that were declared with the class that the object was created from. This is the primary use for classes, as it allows for you to easily create a group of objects that store the same kind of data.

When doing this, it is important to declare a special function called `__init__` inside the class. This function is called when creating the new class and can be used to initialize values for the new object:

```
1  class Test:
2      def __init__(self):
3          self.x = 10
4
5      def set(self, x):
6          self.x = x
7
8  a = Test()
9  b = Test()
10 print(a.x, b.x) #10 10
11 a.set(20)
12 print(a.x, b.x) #20 10
```

Note the use of the variable `self` in the function declarations. This variable should be first in any function declared that needs to be used on a function level. This variable refers to the object that the function is being called on. For example, in line 11, `self` would refer to `a`. Note that `a.set(20)` is equivalent to `Test.set(a, 20)`. The former is preferred because it is shorter and more readable. Additionally, a function that refers operates on an instance of the class (that is, has `self` as the first argument) are called methods rather than functions.A ny variable referred to with `self.variablename` will stay with the object, while other variables will stay local to the function.

`__init__` can be passed additional arguments in addition to `self` so that the new objects can be initialized differently.

The special method `__str__(self)` can be defined to override how the object is displayed when printed. By default, printing an object will display something like this:

```
>>> class Test:
...     pass
...
>>> a = Test()
>>> print(a)
<__main__.Test object at 0x7f2b806cc9e8>
```

However, if we define the method, it will instead print whatever we want it to. `__str__` should always return a string.

```
class Test:
    def __init__
        (self):
        self.x = 5

    def __str__(self):
        return f' x = {self.x}'

a =
Test()
print(a)
```

Try running the above code. It should print out `x = 5`. Overriding `__str__` makes our output code far simpler when printing objects.

2 What is Encapsulation in Python?

Encapsulation is a technique that involves ***wrapping data and functions into a single unit or object***. This allows you to ***control the way the data and functions are accessed*** and helps to prevent accidental modification of the data.

For example, let's say we have a class called **BankAccount** that has attributes such as **balance** and **account_number** and methods such as **deposit()** and **withdraw()**. We want to ensure that the balance can only be modified through the **deposit()** and **withdraw()** methods and not directly. Here's what the code would look like:

```
class BankAccount:
    def __init__(self, balance, account_number):
        self.__balance = balance
        self.__account_number = account_number

    def deposit(self, amount):
        self.__balance += amount
```

```
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.__balance
```

In this example, the **balance** and **account_number** attributes are prefixed with two underscores, which makes them **private**. This means that they can only be accessed within the **BankAccount** class and not outside of it. The user can only modify the balance through the **deposit()** and **withdraw()** methods, ensuring that the data is secure.

3 Inheritance

A class can be built on another class. This is commonly done when multiple classes should logically be different but share common functionality. A class that inherits from another class is considered to be a subclass of its parent class. The subclass gains all the functions and attributes of its parent class.

An example might be that both Books and Articles could be considered Publications. Both would have a title and an author, but only the book would have a chapter count.

```
class Publication:
    def__init__(self, title,
        author):self.title = title
        self.author = author

    def__str__(self):
        return ' {} by {}' .format(self.title, self.author)

class Book(Publication):
    def__init__(self, title, author, chapters):
        Publication.__init__(self, title,
        author)self.chapters = chapters

class Article(Publication):
    def__init__(self, title, author, magazine):
```

```
        Publication.__init__(self, title, author)
        self.magazine = magazine

    def __str__(self):
        return '{} by {}, published in {}'.format(self.title,
            self.author, self.magazine)

b = Book('Title', 'Author', 100)
a = Article('Other Title', 'Other Author', 'Some Magazine')

print(b) #Title by Author
print(a) #Other Title by Other Author, published in Some Magazine
```

Note that `Article` creates its own version of the `__str__` method, while `Book` does not. This means that `Book` simply uses the method that it inherits from `Publication`.

Also note that each of the subclasses call the constructors of the parent class. This is required to properly initialize the attributes of the object. Though it is not strictly required, doing otherwise would require duplicating the contents of the parent class's constructor in the subclass. A class can inherit from multiple other classes. In that case, it would be expected to call the constructors for both of its parent classes.

Sometimes you will want to check if an object belongs to a specific class. This can be done with the `isinstance` function. This function will return `True` if the given object is an instance of the given class or of a subclass of that class.

```
class A:
    pass
class B:
    pass
class C(A):
    pass
class D(C, B):
    pass

print(isinstance(A(), A)) #True
print(isinstance(B(), A)) #False
print(isinstance(C(), A)) #True
print(isinstance(D(), B)) #True
```

Revision: Lists:

4 List Comprehensions: Concise Iteration

- Modifying Collections

List comprehensions can be used to create new lists by doing something to each element of a currently existing list or range. For example, if we want to make every string in a list of strings lowercase we could use a for-loop to call `.lower()` on each string.

```
>>> strings = ["Python", "N.M."]
>>> result = []
>>> for string in strings:
...     result.append(string.lower())
>>> result
["python", "n.m."]
```

List comprehensions are a more concise way of doing this.

```
>>> strings = ["Python", "N.M."]
>>> result = [string.lower() for string in strings]
>>> print(result)
["python", "n.m."]
```

We can process more complicated elements with list comprehensions. If we have a list of dictionaries containing lists, we can find the maximum element in each sublist by using this list comprehension:

```
>>> data = [{"d": [1,2,3]}, {"d": [4,5]},
...         {"d": [0,0,1]}]
...
>>> [max(dictionary["d"]) for dictionary in data]
[3, 5, 1]
```

- ## Filtering Elements

What if we want to exclude elements? In this code, we take a list of words and keep all words of length less than 2.

```
>>> strings = ["a", "ab", "abb", "aab", "aaa"]
>>> result = []
>>> for string in strings:
...     if len(string) <= 2:
...         result.append(string)
...
>>> result
["a", "ab"]
```

This can be written more concisely by using an `if` statement within a list comprehension.

```
>>> strings = ["a", "ab", "abb", "aab", "aaa"]
>>> [x for x in strings if len(x) <= 2]["a",
"ab"]
```

- ## Nested Comprehensions

We can also write multiple `for` statements and `if` statements in a list comprehension. This example creates a list of floating point numbers by dividing two integers.

```
>>> result = [y / x for x in range(3) if x != 0 for y in range(3)]
>>> print(result)
[0.0, 1.0, 2.0, 0.0, 0.5, 1.0]
```

These statements should be read from right to left. So the previous list comprehension is the same as these nested statements:

```
result = []
for x in range(10):
    if x != 0:
        for y in range(10):
            result.append(y / x)
```

Lab Tasks: 2-d Lists

Write a function called **`find_max`** that takes a two-dimensional list as a parameter and returns the number of the row that sums to the greatest value. For example, if you had the following list of lists:

`list = [[1, 2, 3], [2, 3, 3], [1, 3, 3]]`

The first row would be 6, the second 8 and the third 7. The function would therefore return 1.

You can assume the passed in list of lists has at least one row and one column. You cannot assume that it is square.

Tasks_Classes and Objects:

Task 1

Define a class in Python with the following description:

Private Members

A data member Flight number of type integer
A data member Destination of type string
A data member Distance of type float
A data member Fuel of type float

- A member function CALFUEL() to calculate the value of Fuel as per the following criteria

Distance	Fuel
<=1000	500
more than 1000 and <=2000	1100
more than 2000	2200

- A function FEEDINFO() to allow user to enter values for Flight Number, Destination, Distance & call function CALFUEL() to calculate the quantity of Fuel
- A function SHOWINFO() to allow user to view the content of all the data members

Use the following main() function for testing:

```
#Create an instance in main function and check all your functions
    obj= TEST(*)
    obj.SCHDULE()
    obj.DISPTEST()
```

Task 2

Define a class batsman with the following specifications:

Private attributes:

bcode	4 digits code number
bname	20 characters
innings, notout, runs	integer type
batavg	it is calculated according to the formula –

batavg =runs/(innings-notout)

- calcavg() Method to compute batavg
- readdata() Method to accept value from bcode, name, innings, notout and invoke the function calcavg()
- __repr__() Method to display the data members on the screen and their corresponding memory address

Use the following main() function for testing:

```
#create an instance and
Obj=batsman(*)
obj.readdata()
print(obj)
```

Task 3(Person.py)

Modify the following `Person` class to add a `repeat` method, which repeats the last thing said.

Hint: you will have to modify other methods as well, not just the `repeat` method.

```
class Person:
    """Person class.

    >>> steven = Person("Steven")
    >>> steven.repeat()        # initialized person has the below sta
rting repeat phrase!
    'I squirreled it away before it could catch on fire.'
    >>> steven.say("Hello")
    'Hello'
    >>> steven.repeat()
    'Hello'
    >>> steven.greet()
    'Hello, my name is Steven'
    >>> steven.repeat()
    'Hello, my name is Steven'
    >>> steven.ask("preserve abstraction barriers")
    'Would you please preserve abstraction barriers'
```

```
    >>> steven.repeat()
    'Would you please preserve abstraction barriers'
    """
    def __init__(self, name):
        self.name = name
        "*** YOUR CODE HERE ***"

    def say(self, stuff):
        "*** YOUR CODE HERE ***"
        return stuff

    def ask(self, stuff):
        return self.say("Would you please " + stuff)

    def greet(self):
        return self.say("Hello, my name is " + self.name)

    def repeat(self):
        "*** YOUR CODE HERE ***"
```

Grade Criteria

This lab is graded. Min marks: 0. Max marks: 30

Tasks Shown during Lab (At least 2 Tasks must be shown to LE during lab working) 10 marks	Modern Tool Usage 5 marks	Lab Ethics 5 Marks	Lab Report and Tasks Final Submission 5 marks	Lab Viva/Quiz 5 Marks
Total Marks: 30	Obtained Marks: _____			

Happy Coding!

Deliverables

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

You should submit your codes and report as a compressed zip file. It should contain all files used in the exercises for this lab.

The submitted file should be named cs250_firstname_lastname_lab02.zip