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

CS250: Data Structures and Algorithm

Class: BS Mathematics - 2021

Lab 03:

Introduction to Object Oriented Programming in Python

Classes and Objects

Inheritance

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 03: Introduction to Object Oriented Programming System OOPS in Python Inheritance and Abstraction

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 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
```

Lab Tasks:

Task 1. (Fraction.py)

Download the fraction code file and add a method to overload the multiplication operator and add methods such that it is printed in reduced gcd form.

- Fractions should always be stored in reduced form; for example, store 4/12 as 1/3 and 6/-9 as -2/3.
  - Hint: A GCD (greatest common divisor) function may help.

Task 2. (Person.py).

Consider the Person class from Lab02. Suppose now that we wanted to define a class called `DoubleTalker` to represent people who always say things twice. Define a class `DoubleTalker` that inherit from the `Person` class. Define a method say to exhibit it's behavior.
Test cases:

```
>>> steven = DoubleTalker("Steven")
>>> steven.say("hello")
"hello hello"
>>> steven.say("the sky is falling")
"the sky is falling the sky is falling"
```

Task 3. (Accounts.py).

A. Modify the following `Account` class so that it has a new attribute, `transactions`, that is a list keeping track of any transactions performed. See the doctest for an example.

```
class Account:
    interest = 0.02
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
    def deposit(self, amount):
         self.balance = self.balance + amount
         print("Yes!")
    def withdraw(self, amount):
         """Decrease the account balance by amount and return
the
         new balance.
         """
         if amount > self.balance:
             return 'Insufficient funds'
         self.balance = self.balance - amount
         return self.balance
```

Test Cases:

```
eric_account = Account('Eric')
eric_account.deposit(1000000) #depositing my paycheck for
the week
eric_account.transactions
   #Output: [('deposit', 1000000)]
eric_account.withdraw(100)# buying dinner
#999900
eric_account.transactions
  #[('deposit', 1000000), ('withdraw', 100)]
```

B. **Define a `class CheckingAccount` that inherits from Account class. Define it's own constructor and method deposit that make use of superclass deposit method and prints a message "Have a nice day"**

Use the following test cases:

```
a = Account("Billy")
a.balance

______

c = CheckingAccount("Eric")
c.balance

______

a.deposit(30)

c.deposit(30)

c.interest
```

Task 4 (Time.py)

Download the Time.py file

a. Add to class Time a read-write property **`time`** in which the getter returns a tuple containing the values of the `hour, minute and second` properties, and the setter receives a tuple containing hour, minute and second values and uses them to set the time. In separate file, import Time class, create a **`Time`** object and test the new property.

b. Modify the Time class to provide a read-only property **`universal_str`** that returns a string representation of a Time in 24-hour clock format with two digits each for the hour, minute and second, as in '22:30:00' (for 10:30 PM) or '06:30:00' (for 6:30 AM). Test your new read-only property

Task 05 (Library.py) **Create a module called library. This module should include two classes as well as methods for each. The provided file library_test.py will use the contents of library.py. Do not edit library_test.py for your submission, though you are free to comment out lines using parts of library.py that you have not implemented yet. The contents of library.py should be as follows:**

class Library A `Library` object stores multiple `Book` objects and has methods to return statistics about those books. When printed, the object should list all of the books it contains in alphabetical order by author (as listed. You do not need to split into first and last names).

def add_book(self, book) This method takes in either a line of text or a book object. If the input is a line of text, then it creates a new `Book` object out of it. The line of text will be formatted as are the lines in `library.txt` and should be parsed as needed to form a `Book` object. The object (whether given or created) is then stored inside the library.

def get_authors(self) This method returns a list of all the authors who have works contained in the library. No author should appear more than once.

def get_books_per_author(self) This method returns a dictionary. Each author should be a key in the dictionary. The value for each author should be the number of books by that author in the library.

class Book A `Book` object stores the author and title of a book. When printed, the object should display both the title and author of the book. The constructor should take in two arguments: the title of the book and the author of the book.

You are free (and encouraged) to include any additional methods that you may need to accomplish the tasks given.

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_lab03.zip