Lecture 04: OOP Design

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

February 1, 2024

OOP Design

Student CGPA

Write code for a class Student. Each student registers a number of courses each semester. The class will have course IDs, course titles, credit hours and earned gpa of each of these courses as fields or attributes. It will also have a method add that will allow to add any courses with the mentioend information. In addition there will be a method calculateCGPA that will calculate cgpa using the stored information. Write the constructor and other needful for the implementation of this class.

- Solution 1: For each student
  - A list containing course ids
  - A list containing course titles
  - A list containing credit hours
  - A list containing earned gpas
  - Each list will have relevant information of the courses.

Solution 1

```
class Student:
    def __init__(self):
        self.course_IDs=[]
        self.course_title=[]
        self.credit_hour=[]
        self.earned_gpa=[]
    def add(self,course_id="",title="",crhr=0.0,gpa=0.0):
        self.course_IDs.append(course_id)
        self.course_title.append(title)
        self.credit_hour.append(crhr)
        self.earned_gpa.append(gpa)
    def calculateCGPA(self):
        pass
    def show(self):
    print(self.course_IDs,self.course_title,self.credit_hour,self.earned_g
s = Student()
s.add("DS 1115","OOP for Data Science",3.0,4.0)
s.add("DS 1116","OOP for Data Science Lab",1.0,4.0)
s.show()
```

Solution 2

- Create a new class, Course
- Course will have its own attributes
- The Student class will only have a single list of courses

Solution 2

```
class Student:
    def __init__(self):
        self.course_list=[]
    def add(self,course):
        self.course_list.append(course)
    def calculateCGPA(self):
        pass
    def show(self):
        print([str(s) for s in self.course_list])
class Course:
    def __init__(self,cid="",title="",crhr=0.0,gpa=0.0):
        self.id=cid
        self.title=title
        self.crhr=crhr
        self.gpa=gpa
    def __str__(self):
        return f"({self.id},{self.title},{self.crhr}{self.gpa})"
c1 = Course("DS 1115","OOP for Data Science",3.0,4.0)
c2 = Course("DS 1116","OOP for Data Science Lab",1.0,4.0)
s = Student()
s.add(c1)
s.add(c2)
s.show()
```

Date and Period

```python
class Date:
    def __init__(self,day,month,year):
        self.day=day
        self.month=month
        self.year=year
    def __str__(self):
        return f"{self.day}/{self.month}/{self.year}"
class Period:
    def __init__(self,start,end):
        self.start = start
        self.end=end
    def __str__(self):
        return f"{self.start}-{self.end}"
d1 = Date(31,1,2024)
d2 = Date(31,5,2024)
p = Period(d1,d2)
d1.year=2025
print(p)
```

Date and Period

- In the previous example, the output should be as follows.

```
31/1/2025-31/5/2024
```

- We note that d1 was altered.
- It resulted into changes in period.
- Often, we don't want this to happen.
- Solution - Deep Copy!

Date and Period - Solution

```
class Date:
    def __init__(self,day,month,year):
        self.day=day
        self.month=month
        self.year=year
    def __str__(self):
        return f"{self.day}/{self.month}/{self.year}"
    def copy(self):
        return Date(self.day,self.month,self.year)
class Period:
    def __init__(self,start,end):
        self.start = start.copy()
        self.end=end.copy()
    def __str__(self):
        return f"{self.start}-{self.end}"
d1 = Date(31,1,2024)
d2 = Date(31,5,2024)
p = Period(d1,d2)
d1.year=2025
print(p)
```

Modules

- Modules are used to organize the classes.
- And later use them, improves reusability and managing large projects.
- For each class or similar classes, we store them in .py files.

```
class Date:
    def __init__(self,day,month,year):
        self.day=day
        self.month=month
        self.year=year
    def __str__(self):
        return f"{self.day}/{self.month}/{self.year}"
    def copy(self):
        return Date(self.day,self.month,self.year)
```

Modules

- Use a separate testDate.py to use/test the class.

```python
from date import Date

d1 = Date(12,12,2023)
print(d1)
```

Documentation Strings (docstrings)

- **Declaring Docstrings:** The docstrings are declared using "'triple single quotes"' or """ triple double quotes""" just below the class, method, or function declaration. All functions should have a docstring.
- **Accessing Docstrings:** The docstrings can be accessed using the __doc__ method of the object or using the help function. The below examples demonstrate how to declare and access a docstring.

```python
class Date:
    """This is a date class"""
    def __init__(self,day,month,year):
        """This is a constructor of the date class"""
        self.day=day
        self.month=month
        self.year=year
    def __str__(self):
        """This is a __str__ fucntion of the Date class"""
        return f"{self.day}/{self.month}/{self.year}"
    def copy(self):
        """This is a deep copy method of the Date class"""
        return Date(self.day,self.month,self.year)
```

Documentation Strings (docstrings)

```python
def func1(x,y):
    '''This function takes x,y as input and
    returns their sum'''
    return x+y
help(func1)
```

```
Help on function func1 in module __main__:

func1(x, y)
    This function takes x,y as input and
    returns their sum
```

Documentation Strings (docstrings)

```python
def func1(x,y):
    '''This function takes x,y as input and
    returns their sum'''
    return x+y
print(func1.__doc__)
```

```
This function takes x,y as input and
    returns their sum
```