Computing Bootcamp

Object-Oriented Programming Examples

Practice 4

Hyung-Sin Kim

SNU Graduate School of Data Science

Review

- Class vs. Class object

- Method vs. Function

- Object-oriented programming
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
  - Computing 1 for DS covers OOP more deeply

Object-oriented Programming in Python

Let’s type together!

SchoolMember – Member Class

```
class Member:
    def __init__(self, name: str, address: str, email: str, DoB: str, affiliation: str) -> None:
        self.name = name
        self.address = address
        self.email = email
        self.DoB = DoB
        self.affiliation = affiliation
        self.infoList = [self.name, self.address, self.email, self.DoB, self.affiliation]

    def printInfo(self):
        print(self.infoList)
```

SchoolMember – Student Class

- class Student(Member):
- def __init__(self, name: str, address: str, email: str, DoB: str, affiliation: str, **student_num: str**) -> None:
- **super()**.__init__(name, address, email, DoB, affiliation)
- self.student_num = student_num
- self.advisor = “”
- self.courses_taken = []
- self.courses_taking = []
- self.GPA = 0
- **self.infoList** += [self.student_num, self.advisor, self.courses_taken, self.courses_taking, self.GPA]

SchoolMember – Faculty Class

- class Faculty(Member):
-     def __init__(self, name: str, address: str, email: str, DoB: str, affiliation: str, **faculty_num: str**) -> None:
-         **super()**.__init__(name, address, email, DoB, affiliation)
-         self.faculty_num = faculty_num
-         self.advisees = []
-         self.courses_teaching = []
-         **self.infoList** += [self.faculty_num, self.advisees, self.courses_teaching]

SchoolMember – Making a Class Object

- >>> hyungsin = Faculty(“Hyung-Sin Kim”, “my addr”, “my email”, “my DoB”, “Data Science”, “my faculty_num”)
- >>> type(hyungsin)
- <class ‘__main__.faculty’>
- >>> type(hyungsin) == Faculty
- True

- We now have a new type!

SchoolMember – Testing Inheritance

- Create a new student object and a new faculty object

- Execute printInfo function of each object and see that it works even though Student/Faculty class definition does not define printInfo!

- See if Student and Faculty objects have different items in their infoList but both have member's items

SchoolMember – Testing Polymorphism

```
class Member:
    def switch_affiliation(self, new_affiliation: str):
        print(”Member”, self.name, “changes affiliation from”, self.affiliation, “to”, new_affiliation)
        self.affiliation = new_affiliation
class Student(Member):
    def switch_affiliation(self, new_affiliation: str):
        print(”Student”, self.name, “changes affiliation from”, self.affiliation, “to”, new_affiliation)
        self.affiliation = new_affiliation
class Faculty(Member):
    def switch_affiliation(self, new_affiliation: str):
        print(”Faculty”, self.name, “changes affiliation from”, self.affiliation, “to”, new_affiliation)
        self.affiliation = new_affiliation
```

SchoolMember – Testing Polymorphism

- Execute switch_affiliation function of a Student object and a Faculty object

- See if they prints different messages even though member class already has switch_affiliation’s definition

Cartesian Plane

- Write two classes **Point** and **Line** that give the same output as below:
  - >>> line = Line(Point(1, 1), Point(3, 2)) # a line comprised of two 2D points
  - >>> line.slope()
  - 0.5
  - >>> line.length()
  - 2.23606797749979

For Your OOP

- You need to understand your environment and problem **thoroughly**
- You need to be able to define objects by using related information and behaviors
- You need to figure out relationship between objects
- Finally, drawing a block diagram would be enough for others to understand your program

Thanks!