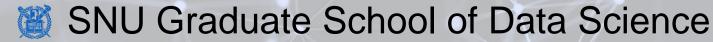
# Object-Oriented Programming Examples

Practice 4

Hyung-Sin Kim



### 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]
```

6

# 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!