Agenda

• Goal
  – Use objects to represent everyone in this course
  – Include:
    • Instructors
    • Students
    • Teaching assistants

• Learn about
  – Inheritance
  – Multiple inheritance
  – Private Variables
class Academic:
    def __init__(self, name, university, status, year, area, salary, gpa):
        self.name = name
        self.university = university
        self.status = status
        self.year = year
        self.area = area
        self.salary = salary
        self.gpa = gpa
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
        print 'Year:' + str(self.year)
        print 'Area of Study:' + self.area
        print 'Salary:' + str(self.salary)
        print 'GPA:' + str(self.gpa)
>>> Student1 = Academic("Louis Sobel", "MIT", "Student", 3, "Computer Science", 0, 5.0)
>>> Student1.identify()
Name: Louis Sobel
Uni: MIT
Status: Student
Year: 3
Area Of Study: Computer Science
Salary: 0
GPA: 5.0
>>> professor = Academic(“Tomas Lozano", "MIT", "Professor", 0, "Computer Science", 1000, 0)

>>> professor.identify()

Name:Tomas Lozano
Uni:MIT
Status:Instructor
Year:0
Area of Study: Computer Science
Salary: 1000
GPA: 0

But we have extra fields we do not always need
```python
class Student:
    def __init__(self, name, university, status, year, area, gpa):
        self.name = name
        self.university = university
        self.status = status
        self.year = year
        self.area = area
        self.gpa = gpa

    def identify(self):
        print('Name: ' + self.name)
        print('Uni: ' + self.university)
        print('Status: ' + self.status)
        print('Year: ' + str(self.year))
        print('Area Of Study: ' + self.area)
        print('GPA: ' + str(self.gpa))
```
class Professor:

def __init__(self, name, university, status, area, salary):
    self.name = name
    self.university = university
    self.status = status
    self.area = area
    self.salary = salary

def identify(self):
    print 'Name:' + self.name
    print 'Uni:' + self.university
    print 'Status:' + self.status
    print 'Area of Study' + self.area
    print 'Salary' + str(self.salary)

But wait, Students and Instructors share similar attributes!
What’s wrong with both approaches?

One class
- Sometimes variables are irrelevant
- A very large, complex class

Two classes
- Repeated behavior
- Increases our work
- We might not implement shared features the same way in the two classes
Inheritance

Academic

Student

Professor
class Academic:
    def __init__(self, name, university, status, area):
        self.name = name
        self.university = university
        self.status = status
        self.area = area
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
        print 'Area of Study:' + self.area
class Student(Academic):
    def __init__(self, name, university, year, area, gpa):
        Academic.__init__(self, name, university, "Student", area)

        self.year = year
        self.gpa = gpa

    def identify(self):
        Academic.identify(self)
        print 'Year:' + str(self.year)
        print 'GPA:' + str(self.gpa)

class Professor(Academic):
    def __init__(self, name, university, area, salary):
        Academic.__init__(self, name, university, "Instructor", area)

        self.salary = salary

    def identify(self):
        Academic.identify(self)
        print 'Salary:' + self.salary
>>> Student1 = Student("Louis Sobel","MIT",3, ‘Computer Science’, 5.0)
>>> Student1.identify()
Name:Louis Sobel
Uni:MIT
Status:Student
Area of Study: Computer Science
Year:3
GPA: 5.0

>>> prof = Instructor("Tomas Perez", "MIT", “Computer Science”, 1000)
>>> prof.identify()
Name:Tomas Perez
Uni:MIT
Status:Instructor
Area of Study: Computer Science
Salary: 1000
Agenda

• Goal
  – Use objects to represent everyone in this course
  – Include:
    • Instructors
    • Students
    • Students who assist instructors

• Learn about:
  – Inheritance
  – Multiple inheritance
  – Private Variables
class Academic:
    def __init__(self, name, university, status, area):
        self.name = name
        self.university = university
        self.status = status
        self.area = area
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
        print 'Area of Study:' + self.area
class Student(Academic):
    def __init__(self, name, university, year, area, gpa):
        Academic.__init__(self, name, university, "Student", area)
        self.year = year
        self.gpa = gpa
    def identify(self):
        Academic.identify(self)
        print 'Year:' + str(self.year)
        print 'GPA:' + str(self.gpa)

class Professor(Academic):
    def __init__(self, name, university, area, salary):
        Academic.__init__(self, name, university, "Instructor", area)
        self.salary = salary
    def identify(self):
        Academic.identify(self)
        print 'Salary:' + self.salary
class TeachingAssistant(Student, Professor):
    def __init__(self, name, university, year, area,
gpa, salary):
        Professor.__init__(self, name, university,
area, salary)
        Student.__init__(self, name, university, year,
gpa)
        self.status = "Teaching Assistant"

>>> TA = TeachingAssistant("Jovana Knezevic",
        "MIT", 5, "Computer Science", 5.0, 100)

>>> TA.identify()
Name: Jovana Knezevic
Uni: MIT
Status: Teaching Assistant

>>> print TA.year
4

>>> print TA.salary
100
class NN:
    def __init__(self):
        self.n = 0
    def get(self):
        self.n += 1
        return str(self.n)
    def reset(self):
        self.n = 0

class NS(NN):
    def get(self, s):
        return s + NN.get(self)

foo = NS()
print foo.get('a') a1
print foo.get('b') b2
foo.reset()
print foo.get('c') c1
Agenda

• Goal
  – Use objects to represent everyone in this course
  – Include:
    • Instructors
    • Students
    • Students who assist instructors

• Learn about
  – Inheritance
  – Multiple inheritance
  – Private Variables