Advanced Object-Oriented Programming

- PoPI

Inheritance

Motivation

- Suppose we have a class:

```
class Employee:
    def __init__(self, nm, prl):
        self._name = nm
        self._payrollNum = prl
        self._salary = "N/A"
    def setSalary(self, sal):
        self._salary = sal
    def statusReport(self):
        str = self._name + ": " +
              self._payrollNum + ", " +
              self._salary + "."
        return str
```

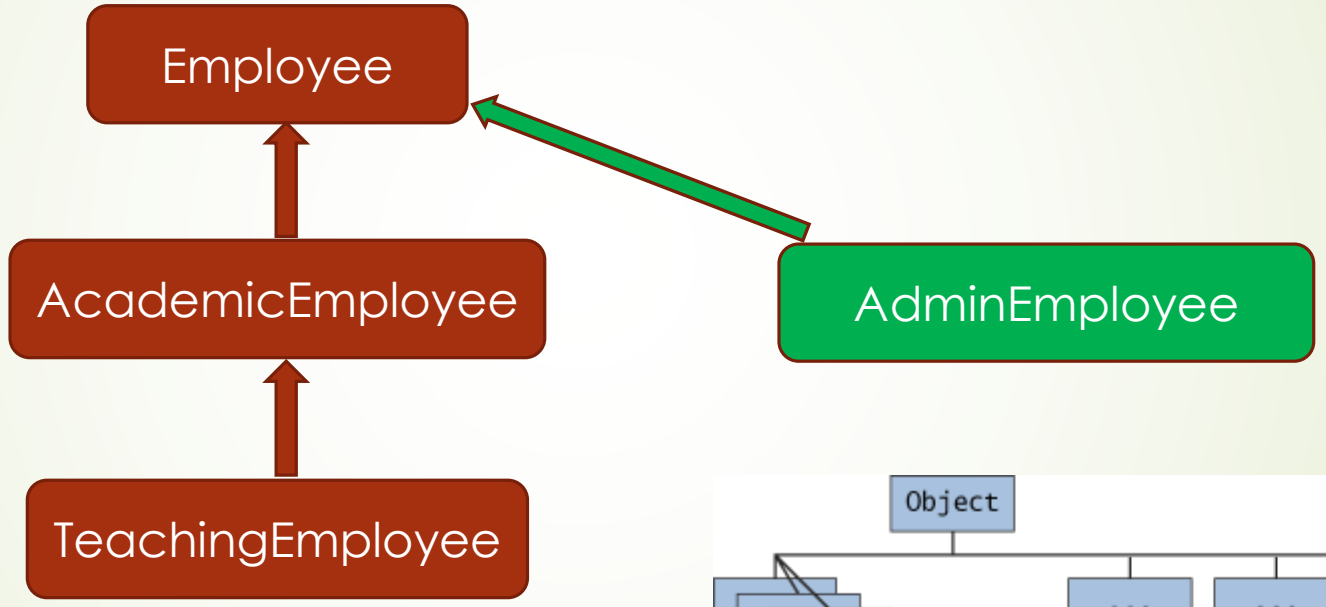
- Now we introduce AcademicEmployee
- Just like Employee but typically has a department assigned
- We can do:

```
class AcademicEmployee:
    def __init__(self, nm, prl):
        self._name = nm
        self._payrollNum = prl
        self._salary = "N/A"
        self._department = "N/A"
    def setDepartment(self, dept)
        self._department = dept
    def setSalary(self, sal)
        …
```

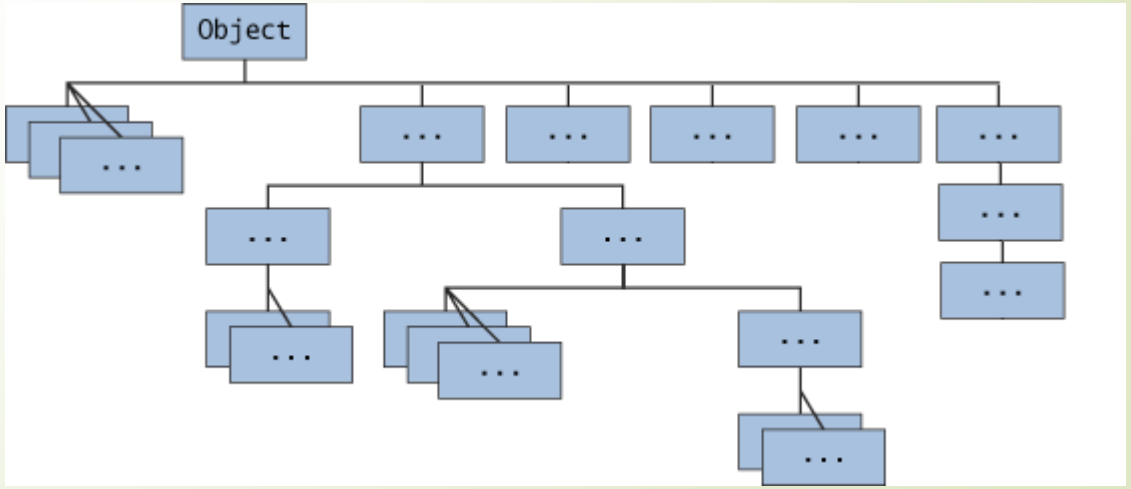
- Copy/paste 95% of the code of Employee

Motivation (cont.)

- Further we want TeachingEmployee
- Just like AcademicEmployee, but typically has a list of courses assigned



- AdminEmployee?
- Copy again?
  - Error prone in any serious application



Inheritance: Syntax

```
class Employee:
    def __init__(self, nm, prl):
        self._name = nm
        self._payrollNum = prl
        self._salary = sal
    def setSalary(self, sal):
        self._salary = sal
    def statusReport(self):
        str = self._name + ": " +
              self._payrollNum + ", " +
              self._salary + "."
        return str
```

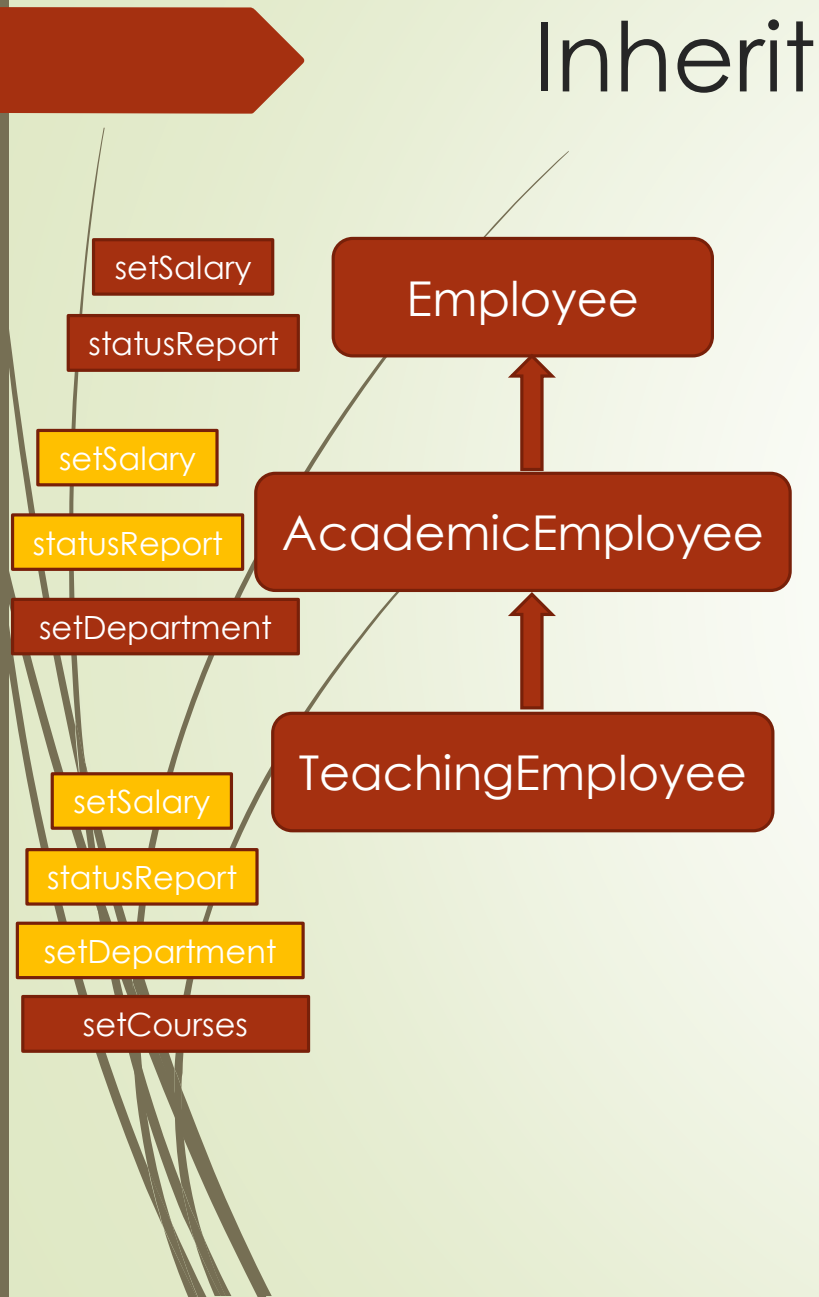
```
class AcademicEmployee(Employee):
    def __init__(self, nm, prl):
        super().__init__(self, nm, prl)
        self._department = "N/A"
    def setDepartment(self, dept)
        self._department = dept
```

- Short and nice! Further:

```
class TeachingEmployee(AcademicEmployee):
    def __init__(self, nm, prl):
        super().__init__(self, nm, prl)
        self._courses = "N/A"
    def setCourses(self, crss)
        self._courses = crss
```

super().super()._init_(self,nm,prl) would call the Employee class

Inheritance: Examples




```
def main():
    paul = Employee("Paul Cooper", 1111)
    roger = AcademicEmployee("Roger Johnson", 2222)
    keith = TeachingEmployee("Keith Mannock", 3333)
    paul.setSalary(25000)
    roger.setSalary(25000)
    keith.setSalary(30000)
    paul.setDepartment ("Computer Science") # error
    roger.setDepartment("Computer Science")
    keith.setDepartment("Computer Science")
    paul.setCourses("PoP1, PoP2") # error
    roger.setCourses("PoP1, PoP2") # error
    keith.setCourses("PoP1, PoP2")
```

Inheritance: Examples

```
class Employee:
        …
    def statusReport(self):
        str = self._name + ": " +
              self._payrollNum + ", "  +
              self._salary + "."
        return str
```

```
…
print(paul.statusReport())
>> Paul Cooper: 1111, 25000.
print(roger.statusReport())
>> Roger Johnson: 2222, 25000.
print(keith.statusReport())
>> Keith Mannock: 3333, 30000.
```

- Too limited!
  - There is more information that we can report on Keith
  - Suppose we are interested also in reporting taught courses

Overriding

```
class Employee:
    def __init__(self, nm, prnum):
        self._name = nm
        self._payrollNum = prnum
        self._salary = sal
    def setSalary(self, sal):
        self._salary = sal
    def statusReport(self):
        str = self._name + ": " +
              self._payrollNum + ", " +
              self._salary + "."
        return str
```

```
class AcademicEmployee(Employee):
    def __init__(self, nm, prnum, sal):
        super().__init__(self, nm, prnum, sal)
        self._department = "N/A"
    def setDepartment(self, dept)
        self._department = dept
```

add

```
class TeachingEmployee(AcademicEmployee):
    def __init__(self, nm, prnum):
        super().__init__(self, nm, prnum)
        self._courses = "N/A"
    def setCourses(self, crss)
        self._courses = crss
    def statusReport(self):
        str = self._name + ": " +
          self._payrollNum + ", " +
          self._salary + ", Teaches" + self._courses +"."
        return str
```

```
print(paul.statusReport())
>>Paul Cooper: 1111, 25000.
print(roger.statusReport())
>>Roger Johnson: 2222, 25000.
print(keith.statusReport())
>>Keith Mannock: 3333, 30000, Teaches PoP1, PoP2.
```

Polymorphism

- Python allows to write code that "magically" works correct for an object of any specialised class. You need to know only the top-level class
- Example:

```
def wealth_report(emp):
    if emp.getSalary() > 25000:
        print "This person is wealthy. Here are details\n" + emp.statusReport()
    else:
        print "This person is not wealthy. Here are details\n" + emp.statusReport()
```

- wealth_report does not care
  - how specialised emp is (as long as emp is Employee)
  - how statusReport() works
- With our previous example:

```
print(wealth_report(paul))
>> This person is not wealthy. Here are details
>> Paul Cooper: 1111, 25000.
print(wealth_report(keith))
>> This person is wealthy. Here are details
>> Keith Mannock: 3333, 30000, Teaches PoP1, PoP2.
```

Some Extras

Using Class constants

- Consider classical use of (global) constants

```
SPEED = 1.6
def distance_travelled_in(time)
    return SPEED*time
```

- Use them in classes too

```
class FastLunarLander:
    SPEED = 3.2
    def distance_travelled_in(time)
        return LunarLander.SPEED*time
```

```
class SlowLunarLander:
    SPEED = 1.6
    def distance_travelled_in(time)
        return LunarLander.SPEED*time
```

- FastLunarLander.SPEED and SlowLunarLander.SPEED will be distinct constant values

Universal Superclass object

- There is a standard superclass object in Python which every defined class inherits
  - The following are equivalent

```
class Employee:
....
```

```
class Employee(object)
....
```

- Provides some methods, one important is __repr__ that returns a string representation of an object used, e.g., for printing

```
lst1 = list([1,2,3])
print(lst1)          #result [1,2,3]
print(lst.__repr__())
```

this can be __str__ as well

- If we try print(keith) where keith was defined as an instance of Employee

  >> <filename.Employee object at 0xb7498d2c>

  - To make the output look as nice as for list we have to override __repr__ in Employee, e.g.,

```
class Employee
        …
        def __repr__(self):
                str = self._name + ": " + self._payrollNum + ", " + self._salary + "."
        return str
```

  - Then

```
print(keith)
>> Keith Mannock: 3333, 30000
```

None reference

- We said that a constructor needs to assign some values to all state variables

```
def __init__(self, nm, prnum, sal):
        self._name = nm
        self._payrollNum = prnum
        self._salary = sal
        self._department = "N/A"
```

- If _department is a string but we don't know initial value we an set "N/A"
- If _department was a list we could set it to []
- If _department is a complex class (say, Department), which dummy value can we use?
  - Use None
- self._department =None
- None keyword can be used anywhere in the program