19CSE201 :Advanced Programming

Lecture 23
Inheritance in Python

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from Matthes, E. (2019). Python crash course: A hands-on, project-based introduction to programming. No Starch Press. and multiple sources from the internet

A Quick Recap

- Classes
- Objects
- Access Specifiers
- Garbage Collection
- Method Overloading
- Operator Overloading

Single Inheritance

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)


x = Person("John", "Doe")
x.printname()
```

```
#inherits person
class Student(Person):
  pass
#create an object of student
# and execute printname()
x = Student("Mike", "Olsen")
x.printname()
```

Super() Function

- super() function will make the child class inherit all the methods and properties from its parent

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
```

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    #adding a new property
    self.graduationyear = 2019

x = Student("John", "Grisham")
print(x.graduationyear)
print(x.firstname)
print(x.lastname)
```

Adding a child method

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname,
  self.lastname)
```

```
class Student(Person):
  def __init__(self, fname, lname,
  year):
    super().__init__(fname, lname)
    self.graduationyear = year

  #adding a new child method()
  def welcome(self):
    print("Welcome", self.firstname,
  self.lastname, "to the class of",
  self.graduationyear)
```

```
#calling the function - the main()
x = Student("John", "Grisham", 2023)
 x.welcome()
```

Checking Class Relationships

- The `issubclass(sub, sup)` boolean function returns true if the given subclass sub is indeed a subclass of the superclass sup.
  - `print(issubclass(Student,Person))`
- The `isinstance(obj, Class)` boolean function returns true if obj is an instance of class Class or is an instance of a subclass of Class
  - `print(isinstance(x,Student))`
  - `print(isinstance(x,Person))`

Method Overriding - Overview

- Method overriding allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.

- When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Method Overriding – Overview Cont.

- The version of a method that is executed will be determined by the object that is used to invoke it.
- If an object of a parent class is used to invoke the method, then the version in the parent class will be executed.
- if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

Method Overriding - Example

```
class Parent():
  def __init__(self):
     self.value = "Inside Parent"

  def show(self):
     print(self.value)
```

```
class Child(Parent):
   def __init__(self):
      self.value = "Inside Child"
    def show(self):
      print(self.value)
```

```
obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

Example - Multiple Inheritance

```
# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)
```

```
# Base class2
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)
```

```
# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
```

```
s1 = Son()
s1.fathername = "FATHER"
s1.mothername = "MOTHER"
s1.parents()
```

Example - Multiple Inheritance Overriding

```
#base class 1
class Parent1():
  def show(self):
       print("Inside Parent1")
```

```
#child class
class Child(Parent1, Parent2):
  def show(self):
       print("Inside Child")
```

```
#base class 2
class Parent2():
  def display(self):
       print("Inside Parent2")
```

```
#Driver code
obj = Child()
obj.show()
obj.display()
```

Example - Multilevel Inheritance

```
# Base class
class Grandfather:
    def __init__(self, grandfathername):
        self.grandfathername = grandfathername
```

```
# Intermediate class
class Father(Grandfather):
  def __init__(self, fathername, grandfathername):
      self.fathername = fathername

      # invoking constructor of Grandfather class
      Grandfather.__init__(self, grandfathername)
```

```
# Derived class
class Son(Father):
  def __init__(self,sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

  def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)
```

```
# Driver code
s1 = Son('Prince', 'Rampal',
  'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

Example - Multilevel Inheritance Overriding

```
#base class
class Parent():
  def display(self):
       print("Inside Parent")
```

```
#child class
class GrandChild(Child):
  def show(self):
       print("Inside GrandChild")
```

```
#Intermediate class
class Child(Parent):
  def show(self):
       print("Inside Child")
```

```
g = GrandChild()
g.show()
g.display()
```

Calling parent method within overridden method

```
class Parent():

  def show(self):
       print("Inside Parent")

class Child(Parent):

  def show(self):
       parent.show(self)
      #try super().show() instead as well
       print("Inside Child")
```

```
obj = Child()
obj.show()
```

Class method

- A class method receives the class as implicit first argument.
- It's a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class.
  - For example it can modify a class variable that will be applicable to all the instances.
- Generally used to to create factory methods.
  - Factory methods return class object (similar to a constructor) for different use cases.

Static method

- A static method does not receive an implicit first argument.
- It's also a method which is bound to the class and not the object of the class.
- It can't access or modify class state.
- It is present in a class because it makes sense for the method to be present in class.
- We generally use static methods to create utility functions.
  - Eg. len(), sort(), etc.

Example – Class & Static methods

```
from datetime import date
class Person:
  def __init__(self, name, age):
        self.name = name
        self.age = age
  @classmethod
  def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)
  @staticmethod
  def isAdult(age):
        return age > 18
```

```
person1 = Person('mayank', 21)
person2 =
  Person.fromBirthYear('mayank',
  1996)
print(person1.age)
print(person2.age)
print(Person.isAdult(22))
```

Exercise

```
1  class A():
2      x = 1
3
4      def __init__(self, n):
5          self.y = n
6          A.x += 1
7
8      def p(self):
9          print(self.y)
10         self.y += 3
11         self.r()
12
13     def r(self):
14         self.y += 2
15         print(self.y)
16
17 class B(A):
18     x = 10
19
20     def __init__(self, n):
21         super().__init__(n)
22         sum = self.y + B.x
23         self.m = sum
24
25     def r(self):
26         self.y += self.x
27         print(self.m)
28
29 a = A(1)
30 b = B(2)
31 a.p()
32 b.p()
```

1. What will be printed when python executes line 31?
2. What will be printed when python executes line 32?

Exercise

```
class MenuItem():
    """An instance represents an item on a menu."""
    def __init__(self, name, is_veggie, price):
        """A new menu item called name with 3 attributes:
        name:      a non-empty str, e.g. 'Chicken Noodle Soup'
        is_veggie: a Bool indicating vegetarian or not
        price:     an int > 0 """
        self.name = name
        self.is_veggie = is_veggie
        assert price > 0
        self.price = price

class LunchItem(MenuItem):
    """An instance represents an item that can also be served at lunch"""
    def __init__(self, name, is_veggie, price, lunch_price):
        """A menu item with one additional attribute:
        lunch_price:  an  int > 0 and <= 10"""
        super().__init__(name, is_veggie, price)
        assert lunch_price > 0
        assert lunch_price <= 10
        self.lunch_price = lunch_price
```

1. Write a python assignment statement that stores in variable item1 the ID of a new MenuItem object whose name is "Tofu Curry", a vegetarian dish costing 24 dollars.
2. Write a python assignment statement that stores in variable item2 the ID of a new LunchItem object whose name is "Hamburger", a non-vegetarian dish that costs 12dollars, but only 8 dollars at lunch.

Quick Summary

- Method Overloading
- Inheritance in python
- Types of inheritance
  - Examples
- Super() function
- Checking class relationships
- Static and Class methods
- Examples
- Exercises

Up Next

STLs