Inheritance_subclass

December 3, 2022

Inheritance and subclass

In this section, we will learn about how to inherit the methods of **parent** class in the subclass as well as try to call the **static method** defined in base class in the subclasses.

[16]:
```
class Employee:
    num_empl = 0
    raise_amt = 1.05

    def __init__(self, first, last, pay):

        self.first = first
        self.last = last
        self.pay = int(pay)

        Employee.num_empl += 1

    def full_name(self):
        return "{} {}".format(self.first, self.last)

    def apply_raise(self):
        return int(self.pay * self.raise_amt)

    # define a static method
    @staticmethod
    def anything(strings):
        print(strings)

empl = Employee("Sad", "Sikei", 65000)
empl.anything("I am going to something great!")
```

```
I am going to something great!
```

In the next, a subclass **developer** inherits all functionalities of parent class **Employee** by using the base class as an argument, in order to get some info regarding the functionalities of subclass, we can use print(help(developer))

[22]:
```
class Developer(Employee):

```

```
    raise_amt = 1.10
    def __init__(self, first, last, pay, prog):
        super().__init__(first, last, pay)
        # or Employee.__init__(self, first, last, pay)
        self.prog = prog

    def program(self):

        print(self.full_name(), "likes programming in", self.prog)


# make an instance
empl = Developer("Smit", "Sali", 65000, "Python")
empl.program()
```

```
Smit Sali likes programming in Python
```

in the next, we create a subclass called **manager** that possesses some functionalites other than developer has.

[28]:
```
class Manager(Employee):


    def __init__(self, first, last, pay, employees = None):
        super().__init__(first, last, pay)

        if employees is None:
            self.employees = []
        else:
            self.employees = employees

    # if employee is not in the list, add them to it
    def add_empl(self, empl):
        if empl not in self.employees:
            self.employees.append(empl)

    def remove_empl(self, empl):
        if empl in self.employees:
            self.employees.remove(empl)

    def show_empl(self):
        for empl in self.employees:

            print("-->", empl.full_name())
            # call the staticmethod using super().
            super().anything("is allowd to implement any idea he/she thinks of.")
```

```
dev_1 = Developer("Smit", "Sali", 65000, "Python")
dev_2 = Developer("Smith", "Jahn", 75000, "C++")

mgr_1 = Manager("Lanh", "Nik", "85000", [dev_1])

#mgr_1.remove_empl(dev_1)
#mgr_1.show_empl()
```

```
--> Smit Sali
is allowd to implement any idea he/she thinks of.
Lanh Nik None
```

Here we call the method **anything()** defined in the base class using **@staticmethod**. This static function can be called in the subclass **Manager**, which gives addtional functionality to the subclass as well. In order to call the **staticmmethod** in the subclass, we have to add `super().staticmethod()`, as shown in the cell above. e.g. `super().anything()`. Then it works fine.

[29]:
```
print(mgr_1.full_name(), mgr_1.show_empl())
```

```
--> Smit Sali
is allowd to implement any idea he/she thinks of.
Lanh Nik None
```

Python comes in with some useful built-in functions like **isinstance**, **issubclass** to check the inheritance of the subclasses.

[68]:
```
print(isinstance(mgr_1, Manager), isinstance(mgr_1, Employee), isinstance(dev_1,␣
 ↪Developer), isinstance(mgr_1, Developer))
# or
print(issubclass(Developer, Employee), issubclass(Manager, Developer))
```

```
True True True False
True False
```

In addition, one can check the **attributes** and **instances** of a class object using following:

[89]:
```
mgr_obj = Manager("Lanh", "Nik", 85000)

#print(mgr_obj.__dict__)

#for attr in dir(mgr_obj):
    #print(attr, getattr(mgr_obj, attr))
```