Classes and Object-Oriented Programming

CMSC 104 Section 2
November 4, 2025

Administrative Notes

Quiz 3 is next Thursday, November 13

- Sorry if the side note in the syllabus was misleading - it’s NOT today

Project 3 will be assigned on Thursday, November 6

- You will have two weeks to do it
-

What is “Object-Oriented Programming?”

A paradigm for developing software that “models” real-world objects in software.

The goal is to make software that is easy to reuse, and hard to make errors that can’t easily be fixed

The programmer tends to be the “problem” with software development, so anything that makes the programmer “better” at the job is a win

Why we don’t emphasize this topic

Python is not normally thought of as an “object-oriented” language

E.g, C++ or Java

Until recently, Python didn’t handle inheritance “properly”

Python is becoming more object-oriented as it develops/matures

But it’s just not where you focus when you start learning Python

Terminology you need to know

Class - a blueprint for creating objects that model entities in your program; for assigning initial values; and for describing behavior

Object - specific instantiation of a class; a member of that class. In a sense, a “variable” of that “type” Also referred to as an “instance” of the class

Method - a procedure or function that is defined as part of a class and can be invoked/executed by instances/objects of that class

Inheritance - when you create a class that is a subclass of another class, members of the subclass automatically get values/methods of the parent class

Subclasses

A “subclass” is a refinement of a larger class

- It has all the properties of its parent class
- And it has additional properties that do not apply to all class members

Examples:

- A class “pets”
  - A subclass “dogs”
    - Dogs are pets, but they are specific types of pets. Cats, birds, etc. can be “pets” but they aren’t “dogs”.

Example: the class “Dog”

```
class Dog:
    def __init__(self, name, age, breed, tricks):
        self.name = name
        self.age = age
        self.breed = breed
        self.tricks = tricks

    def learn_tricks(self,new_tricks):
        for i in range(len(new_tricks)):
            self.tricks.append(new_tricks[i])
```

Example: the class “Patient”

```
class Patient:
   def   init  (self, name, age, gender, insurance,record):
        self.name = name
        self.age = age
        self.gender = gender
        self.insurance = insurance
        self.record = record

   def admit patient  (self, adm date):
        self.record["admit"] = adm_date

#create a patient named "Joe"
patient 1 = Patient("Joe", 54, "M", {"co":"InsCo", "acct":"12345"}, {})
# now add an admission date
patient 1.admit patient("02/15/2024")
# now check to see if it worked
print(patient_1.record)
```

Adding a method to add a second insurance company

```
def add_insurance (self, addl_co, addl_acct):
   self.insurance["second"] = addl_co
   self.insurance["acct2"] = addl_acct
```

Note - this has to be in the class definition section

```
#add a second insurance company
patient_1.add_insurance("NuCo", "98765")
print(patient_1.insurance)
```

This goes in the program code

Now a subclass: “Pediatric_Patient”

```
class Patient:
   def   init  (self, name, age, gender, insurance,record):
       self.name = name
       self.age = age
       self.gender = gender
       self.insurance = insurance
       self.record = record

   def admit patient (self, adm date):
       self.record["admit"] = adm_date

class Pediatric Patient(Patient):
   def   init  (self, name, age, gender, insurance, record, guardian):
       super().  init  (name, age, gender, insurance, record)
       self.guardian = guardian



#create a patient named "Joe"
patient 1 = Patient( "Joe", 54, "M", {"co":"InsCo", "acct":"12345"}, {})
# now add an admission date
patient_1.admit_patient( "02/15/2024")

# now check to see if it worked
print(patient_1.record)

p2 = Pediatric Patient("Sue",7,"F",{"co":"InsCo", "acct":"12345"}, {}, "Steve" )
print(p2.guardian)
```

Pediatric patients must have a guardian named in their records who can make decisions for them

Why do we cover this topic?

When you're writing long, complex programs

- It's easier to think about "objects" that have certain properties
- You just treat your software like you would that "object" in real life
- It makes it easier to group large amounts of related data together
    - A step beyond "lists" and "dictionaries"

When you complete this class, we don't expect you to be an expert in "object-oriented software"

- But we do expect you to be familiar with the term and have an idea of what it is and why it's popular

Now, let’s take a simpler case and walk through development

```
# A simple example of classes and inheritance

class Person:
    """Represents a general person."""

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        """Print a short introduction."""
        print(f"Hi, my name is {self.name} and I am
{self.age} years old.")

#create a couple of persons

joe = person(“Joe”, 35)
mohammed = person(“Mohammed”, 54)
```

We start by defining just the parent class - a “person”

- What do all “persons” have in common that we care about?
- For our purposes, only their name and their age

Next, define subclasses

```
# Subclass (inherits from Person)
class Student(Person):
    """Represents a student, which is a
kind of person."""

    def __init__(self, name, age, major):
        # Call the constructor of the
parent class
        super().__init__(name, age)
        self.major = major

    def study(self):
        """Describe what the student is
studying."""
        print(f"{self.name} is studying
{self.major}.")
```

Now, we'll add a subclass called "Student". This is a subclass of "Person" because every Student is a Person, but not every Person is a Student.

Student inherits all the properties of a person

Then we add an additional property, "major" which represents the student's major field of study.

We define a new method, which allows the student to describe their major area of study.

Another subclass: Teacher

```
# Another subclass
class Teacher(Person):
    """Represents a teacher, which is
also a kind of person."""

    def __init__(self, name, age,
subject):
        super().__init__(name, age)
        self.subject = subject

    def teach(self):
        """Describe what the teacher
teaches."""
        print(f"{self.name} is teaching
{self.subject}.")
```

Again - every Teacher is a Person; not every Person is a Teacher.

Teachers have a subject that they teach.

Teachers also have a method that allows them to explain what they teach.

Let’s declare some Students and some Teachers

```
# Create objects
alice = Student("Alice", 19,
"Psychology")
bob = Teacher("Bob", 45,
"Mathematics")

# Call methods
alice.introduce()
alice.study()

bob.introduce()
bob.teach()
```

Important point:

- Members of a subclass have access to all the methods of their parent class, as well as to all the methods of their own subclass
- They do NOT have access to methods belonging to a different subclass of the same parent class
- What happens if we try:

  ```
  alice.teach()
  ```

  Or

  ```
  bob.study()
  ```

Adding methods to a class

First off, don’t do this. That’s the general answer

- Just define a new subclass that inherits all the existing methods, and adds the new method you want to use

You can edit your program to change the definition of your original class or subclass

```
# Another subclass
class Teacher(Person):
    """Represents a teacher, which is also a kind of
person."""

    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def teach(self):
        """Describe what the teacher teaches."""
        print(f"{self.name} is teaching
{self.subject}.")

     def add_course(self, cls):
         self.subject = self.subject + " " + cls
         self.subject.split()
```