Object Oriented Programming

Introduction to Programming

Dr. Paul Vrbik

November 27, 2018

Programming Paradigms

Imperative programming

Programmer says what to do by

1. Procedural – grouping instructions into functions.
2. Object-oriented – grouping instructions into functions with state memory.

Note: Imperative (adjective) means giving an authoritative command.

Programming Paradigms

Declarative programming

Programmer declares what they want through

1. functional a series of function applications.
2. logic a question about a system of facts and rules.
3. mathematical optimization.

Object Oriented Programming

The fundamental building block of OOP is the class or object. In Object Oriented Programming (OOP) the design principle is to solve a problem by creating objects that interact with one another.

Definition (Object)

An object is a collection of data called fields or attributes along with code grouped into methods. An object can reference and change itself and has a notion of self.

First Object

```
>>> class Point():
...         def __init__(self):
...             self.x = 0
...             self.y = 0
>>> p = Point()
>>> p
<__main__.Point object at 0x10a9e0dd8>
>>> type(p)
<class '__main__.Point'>
>>> p.x = 2
>>> p.y = 3
>>> (p.x, p.y)
(2, 3)
```

Motivating Question

Question

Suppose we had to represent a group of students and store data about them. How can we store this data in Python?

Nested Lists

```
>>> data = [[name1, gpa1, utoid1],
            [name2, gpa2, utoid2],
            ...,
            [namek, gpak, utoidk]]
```

Problems

1. Attribute order must be memorized.
2. Finding a student is hard.
3. List elements do not have meaningful names. "`data[3]`"
4. No easy way to compare students.
5. Cannot support multiple types of students.

Dictionary of Lists

```
>>> data = { name1: [gpa1, utoid1],
             name2: [gpa2, utoid2],
             ...,
             namek: [gpak, utoidk]}
```

Problems

1. Attribute order must be memorized.
2. No easy way to compare students
3. Cannot support multiple types of students

Dictionary of Lists

```
>>> data = {name1: {"gpa": {gpa1}, "id": {utoidk}},
            name2: {"gpa": {gpa1}, "id": {utoidk}},
            ...,
            namek: {"gpa": {gpak}, "id": {utoidk}}
```

Problems

1. Gets messy (i.e. Assignment 3).
2. Hard to modify.

First Object

```
>>> class Point():
...         def __init__(self, x:int, y:int):
...             self.x = 0
...             self.y = 0

>>> p = Point(2, 3)
>>> (p.x, p.y)
(2, 3)
```

Question

Implement the student class.

Object Methods

```
>>> class Person:
...     def __init__(self, name, age):
...         self.name = name
...         self.age  = age

...     def foo(self):
...          print("Hi! My name is {}.".format(self.name) )
...          return None

>>> p = Person("Slim Shady")
>>> p.foo()
Hi! My name is Slim Shady.
```

Initializes the object.

```
>>> class Counter:
...     def __init__(self) -> None:
...         self._value = 0

...     def get_value(self) -> int:
...         return self._value

...     def click(self) -> None:
...         self._value = self._value + 1

...     def reset(self) -> None:
...         self._value = 0

>>> sally = Counter()
```

Private Variables

The underscore on `_value` in the previous slide is used to denote this name as private indicate that programmers should never manipulate this value outside the object.

Question

Implement the methods `add_student`, `drop_student`, and `is_passing` in the student class.

Next Time

1. More objects.