Object-oriented paradigm

Object-oriented paradigm

In general, programs are instructions to do something on given data. Programs' two main components are:

- *Data* (or variables to store data).
- Statements that define *computation* (or something to do).

Object-oriented paradigm

- In conventional programming, data and computation are defined separately.
- In object-oriented programming, data and computation must be defined together.

What is an object?

In a nut shell, an object is data equipped with relevant computation. Formally, an *object* is defined by:

- A set of attributes that define data.
- A set of operations (or behaviors) that define actions on data.

What is an object?

Example. Lucy, a dog, is an object defined by the following:

- Attributes: female, Golden Retriever, 22 in., 60 lbs., 5 years old.
- Operations (or behaviors): bark, walk, run, sit, jump, swim, ...

What is an object?

Example. Joe, a colored rectangle, is an object defined by the following:

- Attributes: color = red, length = 10 in., width = 5 in.
- Operations: change its color, compute and print its area, ...

What is an object?

Example. Rusty, a dog, is an object defined by the following:

- Attributes: male, German Shepard, 25 in., 70 lbs., 7 years old.
- Operations (or behaviors): bark, walk, run, sit, jump, swim, ...

Objects

Many objects share common attribute types and operations. For example, all dogs share:

- Attributes: gender, breed, height, weight, age.
- Operations (or behaviors): bark, walk, run, sit, jump, swim, ...

In fact, all dog objects can be defined by these attributes and operations.

What is a class?

In a nut shell, a class is an abstract definition of objects that share common attribute types and operations. Formally, a *class* is defined by:

- A set of attributes without specific data.
- A set of operations (or behaviors) that define actions on data.

Classes and objects

- A class is an abstract description of objects that share common attribute types and operations.
- A class may be regarded as simply collection of objects that share common attribute types and operations.
- An object is an instance of a class. For example, the object Lucy is an instance of the class Dog.

Classes and objects

- In Python, an object must be defined as an instance of a class.
- Every Python class has a *constructor* that instantiates an object.
- Attributes are often called *fields*, and operations (or behaviors) are called *methods*.

Python class

```
class Dog(object):
```

Constructor

Methods (operations or behaviors)

Constructor

- A *constructor* is a special function that instantiates an object.
- Objects are created from a class by invoking a constructor.
- Constructors usually have parameters, and constructors create objects according to the values of parameters.

Defining a constructor

```
class Dog(object):

  def __init__(self, b = 'German Shepherd',
    g = 'male', h = 20, w = 65, a = 5):
    self.breed = b
    self.gender = g
    self.height = h
    self.weight = w
    self.age = a
```

Methods (operations or behaviors)

Instantiating objects

```
>>> rusty = Dog()
>>> print rusty.breed
German Shepherd
>>> print rusty.age
5

>>> lucy = Dog('Golden Retriever',
    'female', 22, 60, 6)
>>> print lucy.breed
Golden Retriever
>>> print lucy.age
6
```

Methods

- *Methods* define objects' operations, similar to the way functions define basic computational tasks.
- Many (but not all) methods return values, just like many functions return values.

Defining methods

```
class Dog(object):

  def __init__(self, b = 'German Shepherd',
    g = 'male', h = 20, w = 65, a = 5):
    self.breed = b
    self.gender = g
    self.height = h
    self.weight = w
    self.age = a

  def get_older(self):
    self.age = self.age + 1

  def gain_weight(self, w):
    self.weight = self.weight + w
```


Invoking methods

```
>>> rusty = Dog()
>>> print rusty.age
5
>>> print rusty.weight
65
>>> rusty.get_older()
>>> print rusty.age
6
>>> rusty.gain_weight(10)
>>> print rusty.weight
75
```

Defining methods

```
class Dog(object):

  def __init__(self, b = 'German Shepherd',
    g = 'male', h = 20, w = 65, a = 5):
    self.breed = b
    self.gender = g
    self.height = h
    self.weight = w
    self.age = a

  def age_difference(self, another_dog):
    d = self.age - another_dog.age
    if d < 0: return -d
    else: return d
```


Invoking methods

```
>>> rusty = Dog()
>>> lucy = Dog('Golden Retriever',
    'female', 22, 60, 6)
>>> print rusty.age
5
>>> print lucy.age
6
>>> print rusty.age_difference(lucy)
1
```


Defining methods

```
class Dog(object):

  def __init__(self, b = 'German Shepherd',
    g = 'male', h = 20, w = 65, a = 5):
    self.breed = b
    self.gender = g
    self.height = h
    self.weight = w
    self.age = a

  def older_dog(self, another_dog):
    if self.age > another_dog.age:
      return self
    else:
      return another_dog
```

Invoking methods

```
>>> rusty = Dog()
>>> lucy = Dog('Golden Retriever',
    'female', 22, 60, 6)
>>> print rusty.age
5
>>> print lucy.age
6
>>> smart_dog = lucy.older_dog(rusty)
>>> print smart_dog.breed
Golden Retriever
```


Invoking methods

```
>>> lucy = Dog('Golden Retriever',
    'female', 22, 60, 6)
>>> lucy_copy = lucy
>>> lucy_copy.get_older()
>>> print lucy_copy.age
7
>>> print lucy.age
7
```

```
class Time(object):
  '''Represents the time of day in the 24-hour notation.
     Attributes: hour, minute, second'''

  def __init__(self, hour=0, minute=0, second=0):
  '''Instantiates a Time object.  Unless specified otherwise, it is
     initialized to 00:00:00.'''
    self.hour = hour
    self.minute = minute
    self.second = second

  def __str__(self):
  '''Returns a string representation of a Time object.'''
    return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)

  def to_hours(self):
    return self.hour

  def to_minutes(self):
    return self.hour * 60 + self.minute

  def from_hours(self, hours):
    self.hour = hours % 24

  def from_minutes(self, minutes):
    self.hour = (minutes / 60) % 24
    self.minute = minutes % 60

  def h_increment(self, hours):
    self.from_hours(self.to_hours() + hours)

  def m_increment(self, minutes):
    self.from_minutes(self.to_minutes() + minutes)
```