Lecture #6: Everything's an Object!

Presented by Jamal Bouajjaj

2023–10–02

For University of New Haven's Fall 2023 CSCIxx51 Course

Definitions

A *namespace* is the mapping from a name to an object, for example `a = 5`, a is the namespace for the object that was just created.

Objects?

Objects!

On a core programming level, an object is a *thing* you can intract with.
Similar to C++

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings: YES (trick question, same as list: sequences)

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings: YES (trick question, same as list: sequences)
- Integers?:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings: YES (trick question, same as list: sequences)
- Integers?: YES

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings: YES (trick question, same as list: sequences)
- Integers?: YES
- You???:

POP QUIZ

POP QUIZ: Are the following an object?

- Class Instances: YES
- Functions: YES
- Lists: YES
- Strings: YES (trick question, same as list: sequences)
- Integers?: YES
- You???: wat

You PASS

That's right, EVERYTHING in Python is an object and is treated as such.

Object Attributes

Each object can have attributes (or attribube references), which are variables pertaining to the object. The attributes can be called with the dot syntax:

```
object.attriube
```

Object Methods

Each object can have methods, which are functions inside the object.

```
object.method()
```

Wait...we've seen this before

Remember things `"Hello!".endswith('1')`? Well that is because `"Hello!"` is a string object, and `endswith()` is a method of that object.

Classes

Definition

Classes is a building block to making your own objects.

To make your own class, we can define one like below as an example

```
class A:
  b = 5

  def test(self):
    return "test good"
```

Where `b` is an *attribute* and `test` is a *method* of that class.

Class Instance

After declaration of your class, you can create instances of the class.

```
b = A() # instance
print(b.b)
b.test()
```

Variables

A class variables is one that is defined in the class, and is shared by all classes

```
class Fruit:
  is_fruit = True

orange = Fruit()
apple = Fruit()

print(orange.is_fruit)
print(apple.is_fruit)
```

Variables Caveat

There is a caveat when objects are not re-definable (i.e mutable)

```
class Fruit:
  fruit_types = []
  name = None

orange = Fruit()
apple = Fruit()

orange.name = "orange"     # New object is created
orange.fruit_types.append('orange') # wait a minute...
print(apple.name)                   # ...
print(apple.fruit_types)            # wat
```

We'll get to how to fix this soon!

Instance Objects

In classes, you can have instance-specific objects refered to with `self` in the class.

```
class Fruit:
  def append_fruit(self, f_name):
    if not hasattr(self, 'b'):
      self.b = []
    self.b.append(f_name)

orange = Fruit()
orange.append_fruit('orange')
apple = Fruit()
apple.append_fruit('apple')
orange.append_fruit('orange2')

print(apple.b)
```

Functions and Self

Each class function is automatically given one argument: `self`. This refers to the object's own instance.

You don't pass in a `self` when you call the class instance's function

Technically it's just an argument, so it doesn't have to be named `self`, but that is convention.

Special Class Name

In Python, there are some class names that gets called automatically depending on what is done to the object or around it, which are *special names*.

The most popular one is `__init__()`, where it gets called when a class object is initialized, and is useful to define some stuff

Init

```
class Fruit:
  def __init__(self, name):
    print("-> Initializing")
    self.name = name

orange = Fruit('orange')
print(orange.name)
```

Fun ones

There are some fun ones, such as when objects get added:

```
class Fruit:
    def __init__(self, name):
        self.type = name
    def __add__(self, other):
        if hasattr(other, 'type'):
            if other.type == 'orange' and self.type == 'apple':
                return 'banana'
        return None

apple = Fruit('apple')
orange = Fruit('orange')

print(apple + orange)
print(orange + apple)
```

No Privacy!

In Python there is not such thing as a "private" variable as you would in C++.

Convension is to have "private" variables start with an underscore, and hope nobody accesses it!

Well...Sort of

There is a not-known feature known as name mangling, where a method with two leading underscores (but no more than 1 trailing one) like `__var` will be replaced with `_classname__var`.

You can still access the `_classname__var` method.

So sort of useful I guess, but not really used though.

Subclassing

Classes can be sub-classes, which the sub-class will inherit all attribues and methods. You can overide them, or re-call them with `super()`

```
class Fruit:
    def buy(self):
      print("Give Kromer")
    def mix(self):
      print("Mixing")
    def eat(self):
      print("Eating...yummy")

class Watermelon(Fruit):
    def eat(self):
      super().eat()
      #Fruit.eat(self)  # possible, don't use!
      print("\tVery water-y")
    def mix(self):
      print("Not mixable")

w = Watermelon()
w.buy()
w.mix()
w.eat()
```

End

The end