# Class Attributes in OOP

Some attribute values are shared across all objects of a given class. Such attributes are associated with the class itself, rather than any individual instance of the class. For instance, let us say that a bank pays interest on the balance of accounts at a fixed interest rate. That interest rate may change, but it is a single value shared across all accounts.

Class attributes are created by assignment statements in the suite of a `class` statement, outside of any method definition. In the broader developer community, class attributes may also be called class variables or static variables. The following class statement creates a class attribute for `Account` with the name `interest`.

```
>>> class Account(object):
        interest = 0.02            # A class attribute
        def __init__(self, account_holder):
            self.balance = 0
            self.holder = account_holder
        # Additional methods would be defined here
```

This attribute can still be accessed from any instance of the class.

```
>>> tom_account = Account('Tom')
>>> jim_account = Account('Jim')
>>> tom_account.interest
0.02
>>> jim_account.interest
0.02
```

However, a single assignment statement to a class attribute changes the value of the attribute for all instances of the class.

```
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
>>> jim_account.interest
0.04
```

**Attribute names.** We have introduced enough complexity into our object system that we have to specify how names are resolved to particular attributes. After all, we could easily have a class attribute and an instance attribute with the same name.

As we have seen, a dot expressions consist of an expression, a dot, and a name:

```
<expression> . <name>
```

To evaluate a dot expression:

1. Evaluate the `<expression>` to the left of the dot, which yields the *object* of the dot expression.
2. `<name>` is matched against the instance attributes of that object; if an attribute with that name exists, its value is returned.
3. If `<name>` does not appear among instance attributes, then `<name>` is looked up in the class, which yields a class attribute value.
4. That value is returned unless it is a function, in which case a bound method is returned instead.

In this evaluation procedure, instance attributes are found before class attributes, just as local names have priority over global in an environment. Methods defined within the class are bound to the object of the dot expression during the third step of this evaluation procedure. The procedure for looking up a name in a class has additional nuances that will arise shortly, once we introduce class inheritance.

**Assignment.** All assignment statements that contain a dot expression on their left-hand side affect attributes for the object of that dot expression. If the object is an instance, then assignment sets an instance attribute. If the object is a class, then assignment sets a class attribute. As a consequence of this rule, assignment to an attribute of an object cannot affect the attributes of its class. The examples below illustrate this distinction.

If we assign to the named attribute `interest` of an account instance, we create a new instance attribute that has the same name as the existing class attribute.

```
>>> jim_account.interest = 0.08
```

and that attribute value will be returned from a dot expression.

```
>>> jim_account.interest
0.08
```

However, the class attribute `interest` still retains its original value, which is returned for all other accounts.

```
>>> tom_account.interest
0.04
```

Changes to the class attribute `interest` will affect `tom_account`, but the instance attribute for `jim_account` will be unaffected.

```
>>> Account.interest = 0.05    # changing the class attribute
>>> tom_account.interest      # changes instances without like-named instance attributes
0.05
>>> jim_account.interest      # but the existing instance attribute is unaffected
0.08
```

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#class-attributes