

CSC 241 Notes
Yosef Mendelsohn

With many thanks to Dr. Amber Settle for the original version of these notes.

Prof Settle's notes in turn are based on the Perkovic text.

# Programmer-defined classes

## Review: Namespaces and classes

The general form of a Python class statement is as follows:

class <name> (superclass, …) # assign name and inheritance

 data = value # shared class variables

 def method (self, …) # shared class methods

 self.member = value # per-instance variables

It is very helpful to bear in mind that in Python a class essentially defines a namespace.

The attributes of a class are just names (identifiers) defined in the namespace of the class.

For example, the class attributes of MyClass can be accessed by using MyClass as the namespace:

>>> print(**MyClass.version**)

1.0

>>> print(**MyClass.setTo**)

<function setTo at 0x01EE3C00>

It’s important to emphasize that version and setTo only exist in the MyClass namespace:

>>> print(**version**)

Traceback (most recent call last):

 File "<pyshell#22>", line 1, in <module>

 version

NameError: name 'version' is not defined

>>> print(**setTo**)

Traceback (most recent call last):

 File "<pyshell#23>", line 1, in <module>

 setTo

NameError: name 'setTo' is not defined

Further, every instance object that you create (or, to use proper terminology: "instantiate") gets its own namespace. It also inherits class attributes from the namespace of its class.

Restated: Every time we instantiate a new instance of type MyClass, a new namespace gets created. This new namespace will inherit the attributes from the namespace of its class:

>>> mc1 = MyClass()

>>> mc1.version

1.0

>>> mc1.setTo

<bound method MyClass.setTo of <\_\_main\_\_.MyClass object at 0x01EFF350>>

It’s important to understand that the names version and setTo have been defined in the MyClass namespace.

When the Python interpreter evaluates the expression mc1.version, it will attempt to find the name **version** starting with the namespace of mc1.

Since no name **version** has been explicitly defined in the namespace of mc1, the Python interpreter will then continue its search for the name version in the namespace(s) that mc1 inherits from.

It will find version in the namespace of the class MyClass.

Recall: WhenPython works with classes, it really works with namespaces:

Python automatically maps the invocation of a method by an instance of a class (e.g. mc1.setTo() ) to a call to a function defined in the class namespace on the instance argument.

In other words, a method invocation like:

instance.method(arg1, arg2, …)

is translated by the Python interpreter into:

class.method(instance, arg1, arg2, …)

For example, instead of using

>>> x.setTo(3)

you could use – and the Python interpreter does end up using after an automatic translation – the following:

>>> MyClass.setTo(x, 3)

This explains **the self variable in the class definition**.

The self argument in the class methods refers to the instance making the method call, and the self.xxxx prefix in front of instance variables (or functions) refers to the instance namespace.

def setTo(self, value):

 'sets the instance variable data to value'

 self.data = value

def get(self):

 'returns the value of the instance variable data'

 return self.data

**Exercise 1**: Fill in the missing statements:

>>> x = MyClass()

>>> y = MyClass()

>>> x.version

???

>>> y.version

???

>>> MyClass.version

???

>>> MyClass.version = 1.01

>>> MyClass.version

???

>>> x.version

???

>>> x.version = 'test'

>>> x.version

???

>>> MyClass.version

???

>>> y.version

???

**Exercise 2**: Edit the class **practice\_class\_v1.py**, add a method to the class MyClass called ntimes(). The method will take a parameter n and will reset the value of the instance variable data to n times its starting value.

See **practice\_class\_v2.py** for the solutions.

## String representations for overloaded operators

There are two overloaded operators that return a **string representation of the object**: \_\_repr()\_\_ and \_\_str\_\_().

The operator \_\_repr()\_\_ is supposed to return the canonical string representation of the object.

This means that ideally, but not necessarily, if the function **eval()** is called with the string representation as input, it would return back the original object.

In other words, eval(repr(o)) **should give back the original object o**.

The \_\_repr\_\_() method is automatically called when an expression evaluates to an object in IDLE and this object needs to be shown in the IDLE window.

To illustrate the difference between the two, consider the following (artificial) class:

class Representation(object):

 def \_\_repr\_\_(self):

 return 'canonical string representation'

 def \_\_str\_\_(self):

 return 'pretty string representation'

It would be used as follows:

>>> rep = Representation()

>>> rep

canonical string representation

>>> print(rep)

pretty string representation

**Exercise**: Add a \_\_repr\_\_ method to the MyClass definition.

# An exercise

To review the basics of classes, we’ll do an exercise.

**Exercise**: Develop a new class called Animal that abstracts the concepts of animals and supports three methods:

1. setSpecies(): Takes a species (a string) as a parameter and sets the species of the animal object to it
2. setLanguage(): Takes a language (a string) as a parameter and sets the language of the animal object to it.
3. speak(): Take no parameters and returns a message from the animal shown as below.

An example of how the Animal class would be used is provided below:

>>> snoopy = Animal()

>>> snoopy.setSpecies('dog')

>>> snoopy.setLanguage('bark')

>>> snoopy.speak()

'I am a dog and I bark'

The solutions are in **animal.py**.

# Writing constructors, string, and representation methods

Recall: The method \_\_init\_\_ is a class method that is automatically called when a new instance object is being constructed.

**Exercise 1**: Add a constructor to the Animal class that takes two parameters, representing the species and language of the animal.

**Exercise 2**: Modify the constructor of the Animal class so that it can take two parameters (species, language), one parameter (species only), or no parameters. If a parameter is missing the string ‘default’ will be used for the missing value.

**Exercise 3**: Add definitions for \_\_str\_\_ and \_\_repr\_\_ to the Animal class. Write code to test the new methods.

The solutions for all of them are in the file **animal.py**.