14.8: OBJECT LIFECYCLE

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

In the previous examples, we are defining a class (template) and using that class to create an instance of that class (object) and then using the instance. When the program finishes, all the variables are discarded. Usually we don't think much about the creation and destruction of variables, but often as our objects become more complex, we need to take some action within the object to set things up as the object is being constructed and possibly clean things up as the object is being discarded.

If we want our object to be aware of these moments of construction and destruction, we add specially named methods to our object:

CODE 14.8.1 (PYTHON):

```
%%python3

class PartyAnimal:
   x = 0

   def __init__(self):
     print('I am constructed')

   def party(self) :
     self.x = self.x + 1
     print('So far',self.x)

   def __del__(self):
     print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)

# Code: http://www.py4e.com/code3/party4.py
```

run restart

When this program executes, it produces the following output:

```
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

As Python is constructing our object, it calls our `__init__` method to give us a chance to set up some default or initial values for the object. When Python encounters the line:

```
an = 42
```

It actually 'thows our object away' so it can reuse the `an` variable to store the value `42`. Just at the moment when our `an` object is being 'destroyed' our destructor code (`__del__`) is called. We cannot stop our variable from being destroyed, but we can

do any necessary cleanup right before our object no longer exists.

When developing objects, it is quite common to add a constructor to an object to set in initial values in the object, it is relatively rare to need a destructor for an object.