Objects - Part 3

Python for Everybody

UNIVERSITY OF MICHIGAN

Charles Severance Ph.D.

Clinical Associate Professor of Information
School of Information

Object Lifecycle

- Objects are created, used and discarded
- We have special blocks of code (methods) that get called
  - At the moment of creation (constructor)
  - At the moment of destruction (destructor)
- Constructors are used a lot
- Destructors are seldom used

Sometimes you can be
a little more explicit.

```
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)
```

```
$ python party4.py
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

far this part's the same and
this part's the same.

Constructor

- In object oriented programming, a constructor in a class is a special block of statements called when an object is created

http://en.wikipedia.org/wiki/Constructor_(computer_science)

So the constructor is a special block of statements that are called at

Many Instances

- We can create lots of objects - the class is the template for the object
- We can store each distinct object in its own variable
- We call this having multiple instances of the same class
- Each instance has its own copy of the instance variables

Now we're going to talk about what happens when you have more than one instance.

```
class PartyAnimal:
  x = 0
  name = ""
  def __init__(self, z):
    self.name = z
    print(self.name,"constructed")

  def party(self) :
    self.x = self.x + 1
    print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

party5.py

Constructors can have additional parameters. These can be used to set up instance variables for the particular instance of the class (i.e., for the particular object).

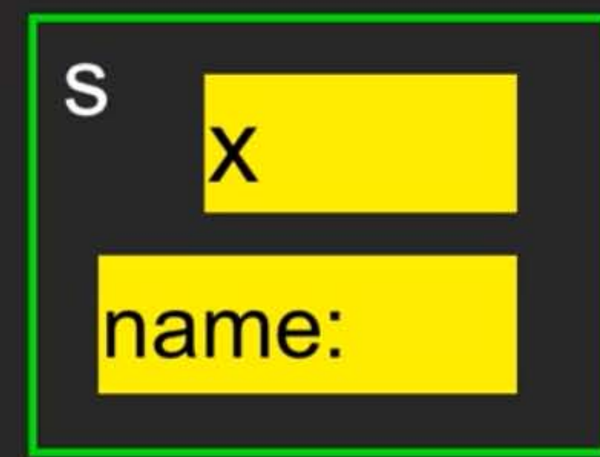
And so, it's going to be pretty much the same as what we've been doing.

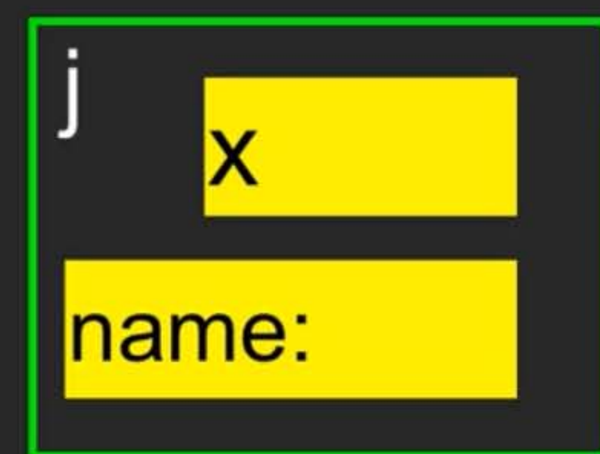
```
class PartyAnimal:
  x = 0
  name = ""
  def __init__(self, z):
    self.name = z
    print(self.name,"constructed")

  def party(self) :
    self.x = self.x + 1
    print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```





We have two independent instances.

We don't have an destructor in this particular one.

Inheritance

http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

of defining the capabilities
of objects called inheritance.