

CS1210 Lecture 25

Oct. 20, 2021

- DS 7 due today
- HW 6 due Thursday, 10/28
- HW 4 still not quite done. Quite a few cases of Academic Honesty Policy violation (and there will still be some HW2 and HW3 changes for AHP violations)

Last time

- Continued classes and object oriented programming (Ch 17, 18, 19)
 - Demonstrate “object-oriented” style of class use - with `__init__` and other methods defining the interface to use of a class

Today

- class attributes (not in interactive text but in 18.2 of pdf version of text)
- Ch 19: inheritance

HW 6 hints

– Q1: think carefully about touches (and contains) – draw pictures

- Think dimension by dimension – three 1D problems
 - if they don't overlap in x, they don't overlap
 - » Express this in terms of center x's and half-widths
 - if ..
 - if ..



– Q2

- Ensure legal moves – i.e. if user enters an illegal choice, print something appropriate and ask for a new choice.
- Computer gameplay can be random (but must be legal). You can use, for instance, `random.randint(...)` and `random.choice(...)` to choose (non-zero) number of balls and (non-empty) heap. (also fine, of course, if you make yours smarter than random)

– Q3 is very very easy compared to Q1 and Q2

Last time: time1.py, time2.py, time2Alt.py

- Look at implementation of
 - incrementTime(self)
 - laterTime(self)methods in **time2.py**. Same basic code as in time1.py but now in OO style. First argument to a method is always object that invokes the method, and standard practice is to use var name 'self'
- Nice feature of classes: you can **overload** operators. That is, you can define how +, -, <, etc. apply to objects of classes that you define
 - `__add__` for + (and `__radd__`)
 - `__lt__` for <
 - `__eq__` for ==, etc.

See how these are used in time2.py

Start with two simple classes, Cat and Dog: catdog.py

Each class has:

- a simple constructor (with optional name as argument, and default if nothing provided)
- a `__repr__` so objects will display readably
- a few methods: `speak`, `setName`, `getName`, `fetch` (only Dog)

```
>>> c1 = Cat()
>>> c2 = Cat()
>>> d = Dog("spot")
>>> for animal in [c, c2, d]:
    print(animal.getName())
    animal.speak()
```

```
fluffy
meow
fluffy
meow
spot
woof
```

```
testCatDog()
```

Class attributes

(not in interactive text but in 18.2 of pdf version of text)

Consider the basic Cat and Dog classes in catdog.py

Each cat and dog has a name but names aren't very unique. We can't distinguish previous example's c1 and c2 (both "fluffy") using just their simple name prop

How we give each cat a unique ID number? **Class attributes** make this easy.

```
class Cat ():  
    scientificName = 'felis catus'  
    numCats = 0
```

scientificName and numCats are attributes "owned" by the class, and shared by all instances

Class attributes

To uniquely identify cats,

- use a class attribute, numCats, initially 0
- each time a Cat object is created (in `__init__`)
 - increment numCats class attribute
 - assign value of numCats to new Cat as id
- Also, update `__repr__` to show id in printed representation

```
>>> c1 = Cat()
```

```
>>> c2 = Cat()
```

```
>>> c2
```

```
<Cat named fluffy. id: 2>
```

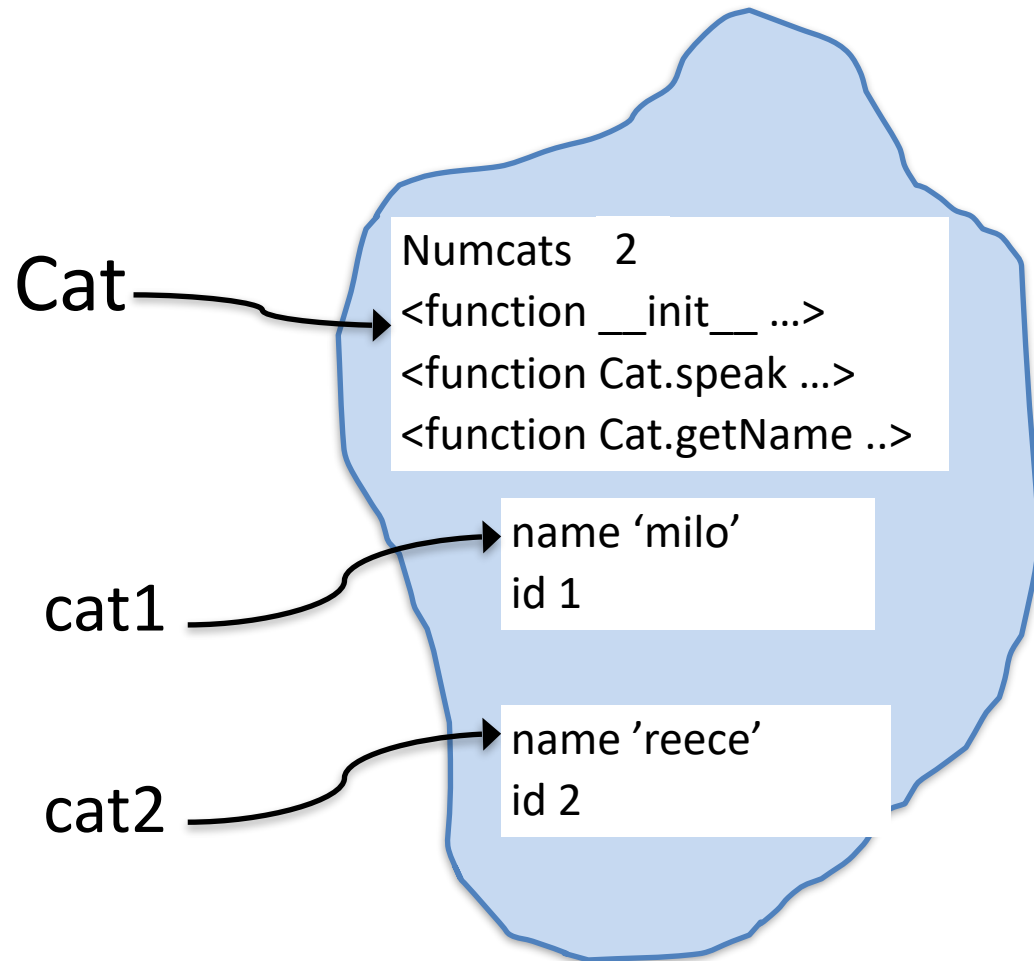
```
>>> c1
```

```
<Cat named fluffy. id: 1>
```

catdogV2.py

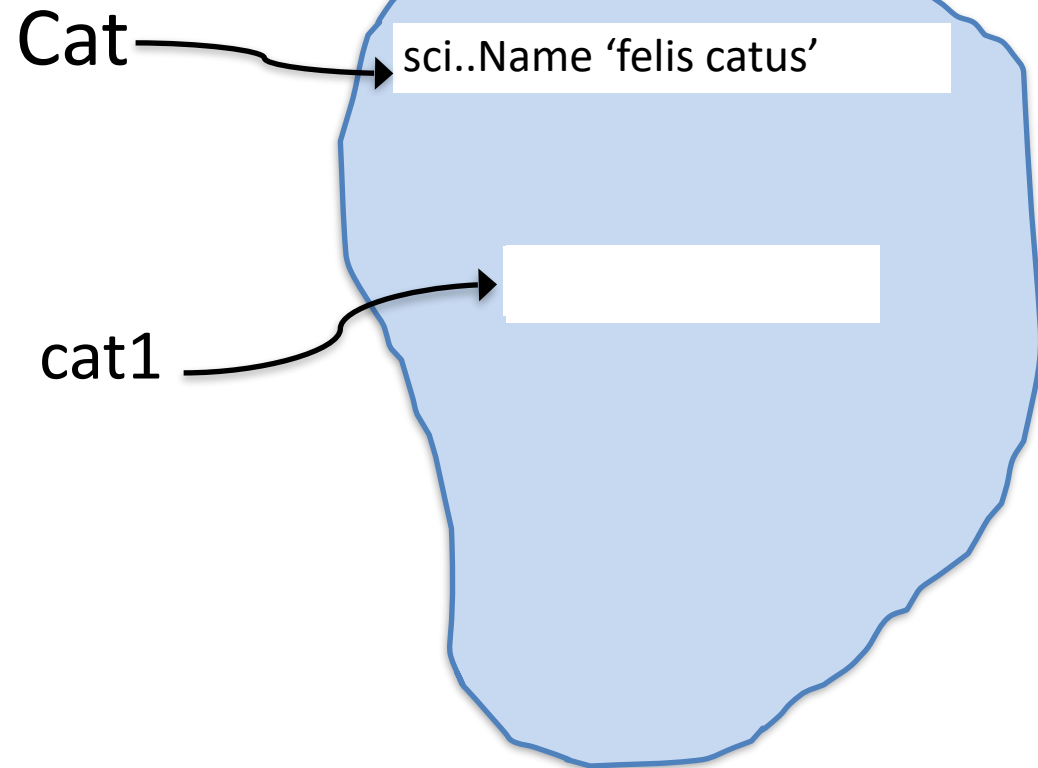
```
>>> class Cat():
    numCats = 0
    def __init__(...):
        ...
    def speak(...):
        ...
    ...

>>> cat1 = Cat('milo')
>>> cat2 = Cat('reece')
```



Be careful when assigning to class attributes

```
class Cat ():  
    scientificName = 'felis catus'  
  
>>> Cat.scientificName  
'felis catus'  
>>> cat1 = Cat()  
>>> cat1.scientificName  
'felis catus'  
>>> cat1.scientificName = 'kitty'  
>>> Cat.scientificName  
'felis catus'  
>>> cat1.scientificName  
'kitty'
```



When you reference an attribute from an instance, Python first checks if the instance contains the attribute. If not, it checks if the class has it. So, the first `cat1.scientificName` above yields 'felis catus'. BUT, when you **assign** to an attribute from an instance, Python uses the namespace of the object. So `cat1.scientificName` does not modify the class attribute. Instead, it creates a new attribute (of the same name) in the instance.

GENERAL RULE: refer to class attributes using class name – `Cat.scientificName` – rather than instance

Ch 19 - Inheritance

When creating classes like Cat and Dog, some properties and methods might naturally be the same in both.

Inheritance, in object-oriented programming languages, is the ability to define a new class that is a modified version of an existing class.

```
class SubClass (SuperClass):
```

...

The new class SubClass **inherits** all methods (and properties) of SuperClass but can also:

- can add new methods (ones not defined in SuperClass)
- redefine (**override**) methods inherited from SuperClass

It's also common to say
"derived class" and "base class"

Ch 19 - Inheritance

```
>>> class Foo ():
    def doSomething(self):
        print('hi')
>>> class Bar(Foo):
    def doSomething2(self):
        print('bye')
>>> b = Bar()
>>> b.doSomething()
hi
>>> b.doSomething2()
Bye
>>> f = Foo()
>>> f.doSomething()
hi
>>> f.doSomething2()
```

Error

Inheritance

```
>>> class Foo ():
    def doSomething(self):
        print('hi')
    def doSomethingElse(self):
        print('something else')
>>> class Bar(Foo):
    def doSomething(self):
        print('hello')

    def doSomething2(self):
        print('bye')
>>> b = Bar()
>>> b.doSomething()
hello
>>> f = Foo()
>>> f.doSomething()
hi
```

Inheritance

```
>>> class Foo ():  
    def __init__(self):  
        self.x = 0
```

```
>>> class Bar(Foo):  
    def __init__(self):  
        self.y = 0
```

```
>>> b = Bar()
```

```
>>> b.y
```

```
0
```

```
>>> b.x
```

Error

Can we inherit the properties of the superclass Foo? Yes, by calling superclass' `__init__`. Not required BUT highly recommended/best practice is for `__init__` of subclass to always first call `__init__` of superclass.

Inheritance

```
>>> class Foo ():
    def __init__(self):
        self.x = 0

>>> class Bar(Foo):
    def __init__(self):
        Foo.__init__(self)
        self.y = 0

>>> b = Bar()
>>> b.y
0
>>> b.x
0
```

Highly recommended/best practice is for `__init__` of subclass to always first call `__init__` of superclass.

Inheritance demo

Use inheritance and define Dog and Cat as subclasses of new Animal class.

- Demo `animals.py`, `testAnimal()`
- Note that the Animal class manages the IDs of all animals
- Animal's `getNumLegs()`, `getName()`, `setName()` methods are **inherited** and used by all subclasses
- Subclasses Dog and Cat have their own `speak` methods, which **override** Animal's default `speak` method
- Cat has a `getFurColor` method. Dog does not
- Dog has a `fetch()` method. Cat does not

Next topics:

- algorithmic complexity/analysis
- randomization