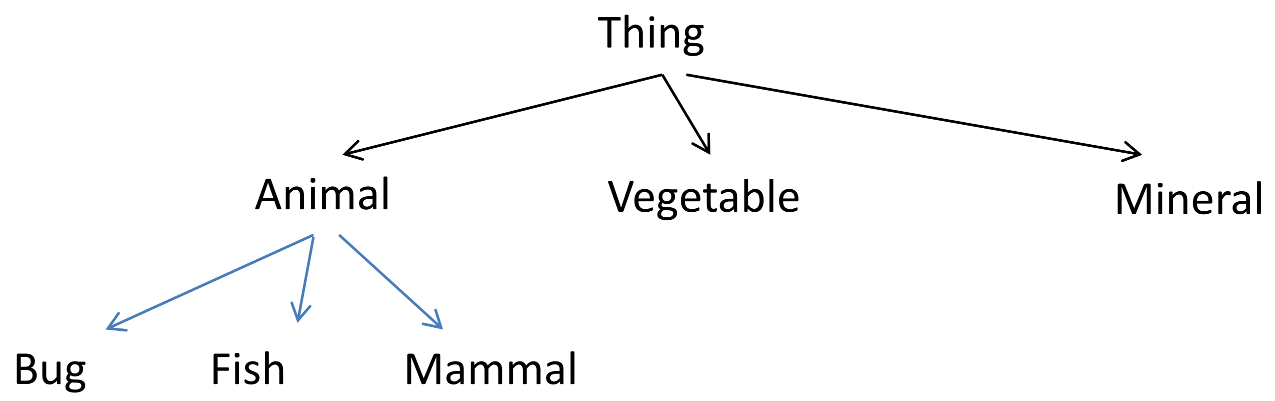
Subclasses

Our last big topic for classes in *subclasses*. These model situations where there are hierarchies of information.

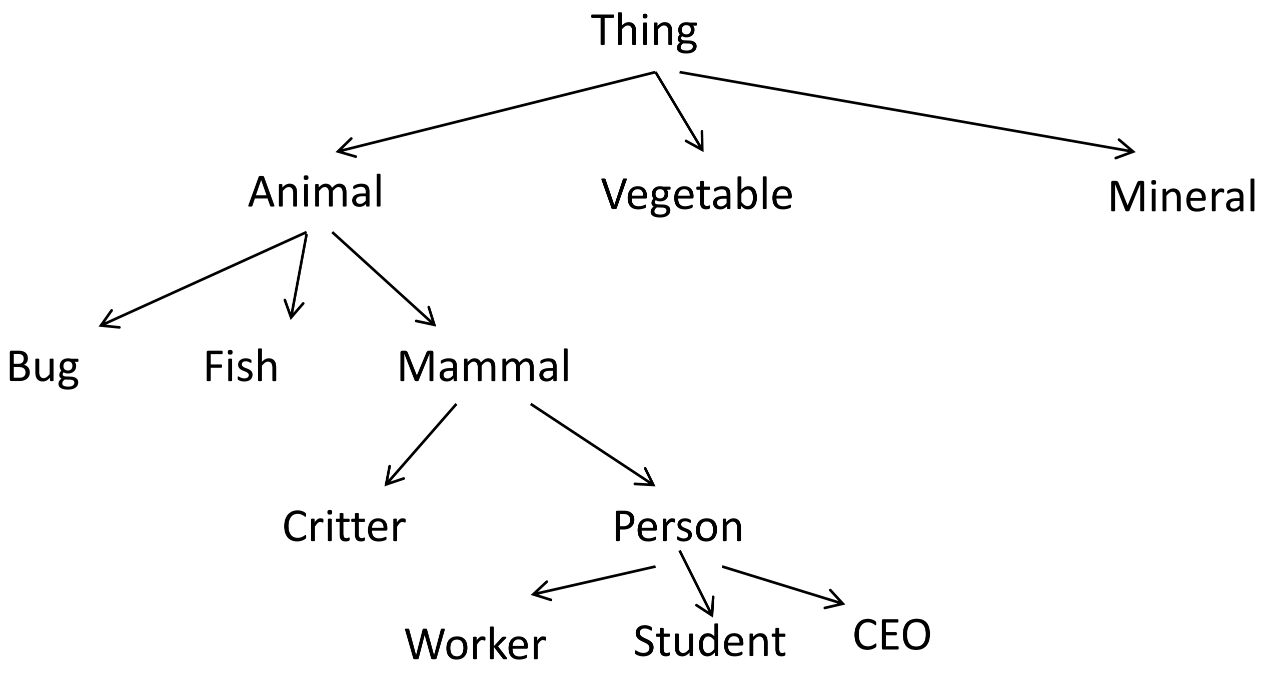
For example, every child knows that Things can be Animals, Vegetables, or Minerals:

$$\begin{tikzcd}
& \text{Thing} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Animal} & \text{Vegetable} & \text{Mineral}
\end{tikzcd}$$

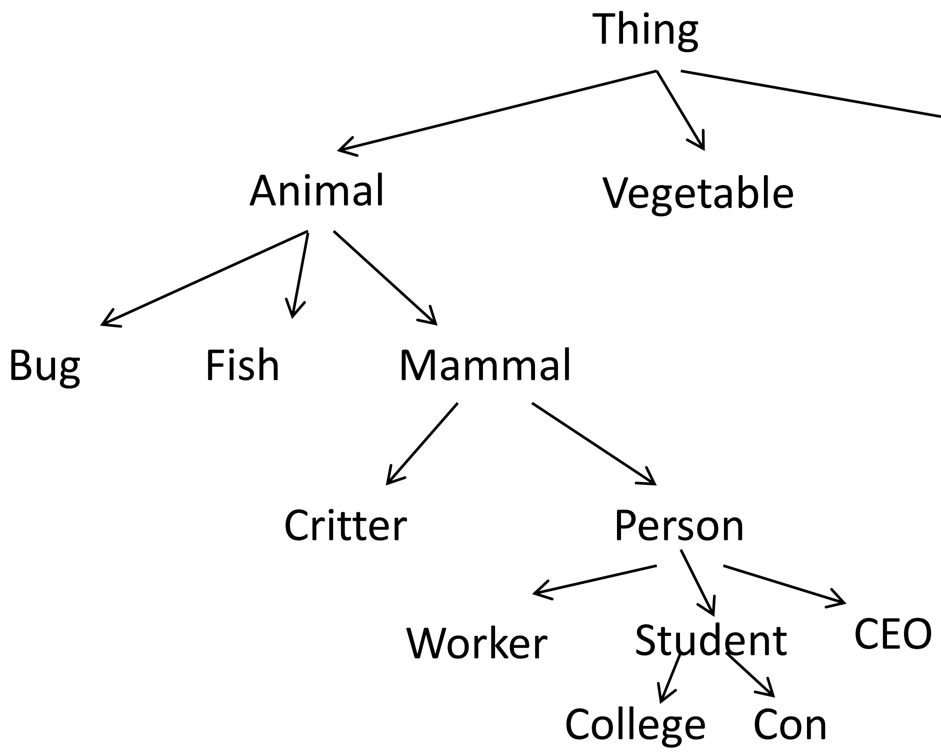
We could subdivide each of these classes. Animals, for example might be bugs or fish or mammals:



Mammals can be divided into people and critters; people can be workers or students or CEO's:

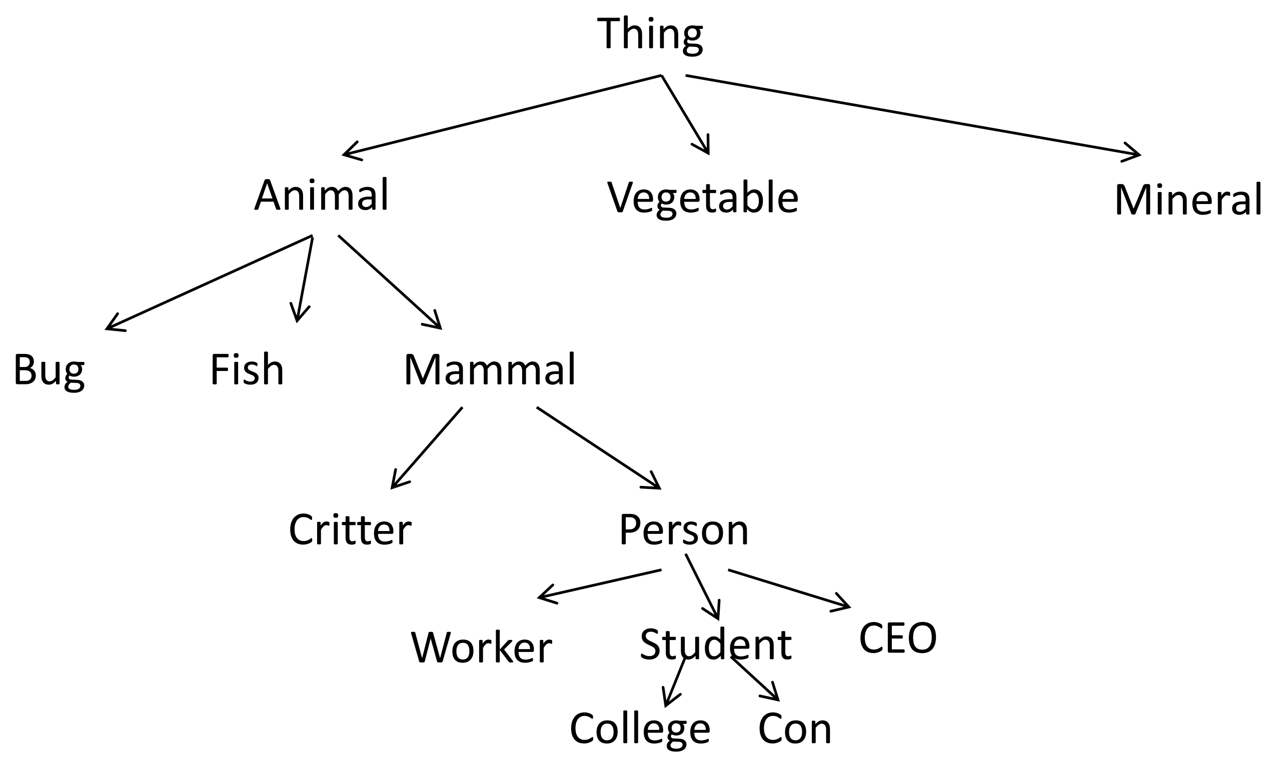


Students, of course, can be College or Con:

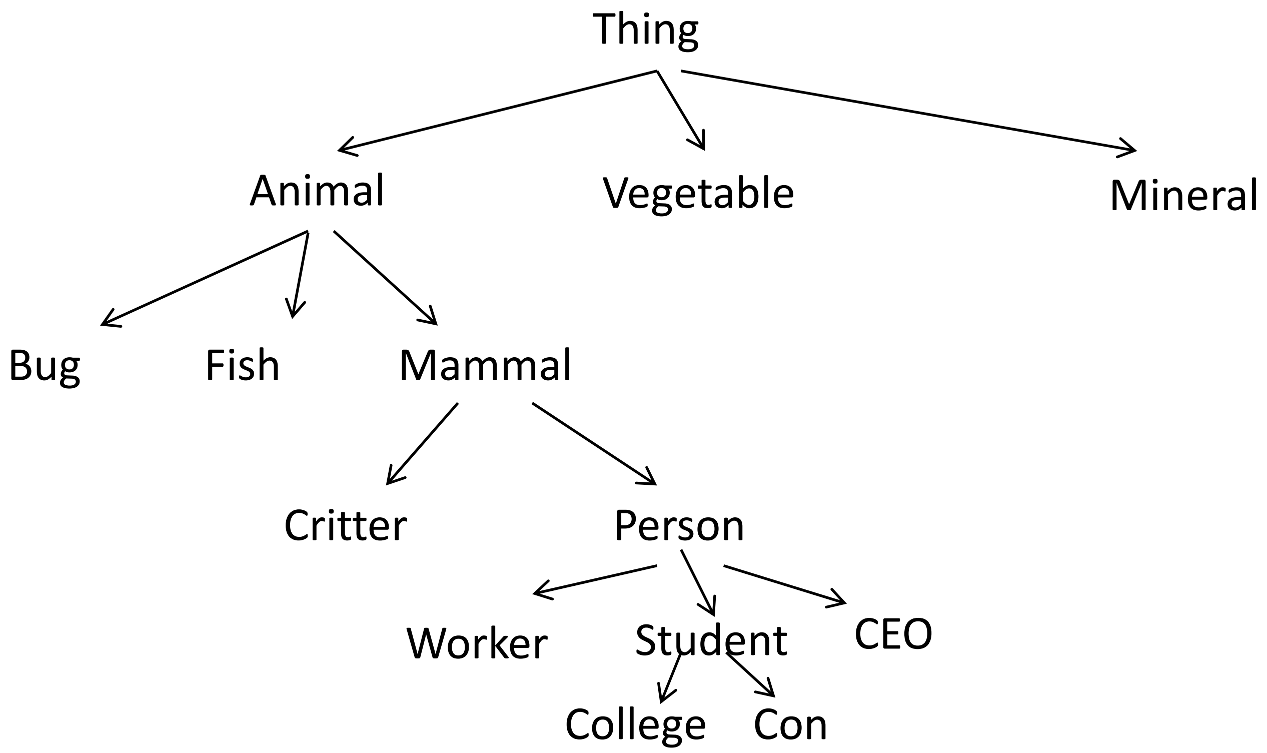


We could add many more refinements to this.

Think about the properties of these classes. Things have location and not much else. Persons have names and phone numbers. Con students have instruments.



Any property that a class has is also held by all of the classes below it. Since Persons have names, so do Workers, Students, and CEO's. It doesn't work the other way -- Con Students have instruments, but Persons don't necessarily, nor do Mammals or Animals.



There is an "is a" relationship as we go down the hierarchy: every Animal is a Thing, every Mammal is an Animal, every Person is a Mammal, every Student is a Person. It doesn't work the other way -- not every Person is a Student.

Now consider just one pair of these classes

$$\begin{tikzcd}\text{Person} \arrow[d] \\ \text{Student}\end{tikzcd}$$

Persons have names and phone numbers. Because all students are Persons, all Students have names and phone numbers. If we are listing the properties of a class, we don't need to say that Students have names; it is enough to know that Persons have names, and Students are Persons.

$$\begin{tikzcd}\text{Person} \arrow[d] \\ \text{Student}\end{tikzcd}$$

When we model this with classes in Python, we say that class Person is a *superclass* or *parent class*, while Student is a *subclass* or *child class*. A subclass in Python *inherits* all of the properties (instance variables and methods) of its parent class. The subclass may *override* this inheritance. For example, our Person class may have Print(self) method that prints the person's name. The Student class will inherit is method, but we may give the student class a different Print(self) method that prints both the student's name and year.

Making subclasses in Python is easy.

```
class Person:
    <blah blah blah>
    <defs of class Person's methods>

class Student( Person ):
    <defs of class Student's methods>
    <Class Student has all of class Person's methods>
    < without them being restated here>
```

If students are people, it would be good for class Student constructor to call the class Person constructor. Suppose the Person constructor takes one argument, the Person's name. How does the Student constructor call it?

A)
```
def __init__(self, name):   # Student constructor
        self.__init__(name)
```

B)
```
def __init__(self, name):    # Student constructor
        Person.__init__(name)
```

C)
```
def __init__(self, name): # Student constructor
        Person.__init__(self, name)
```