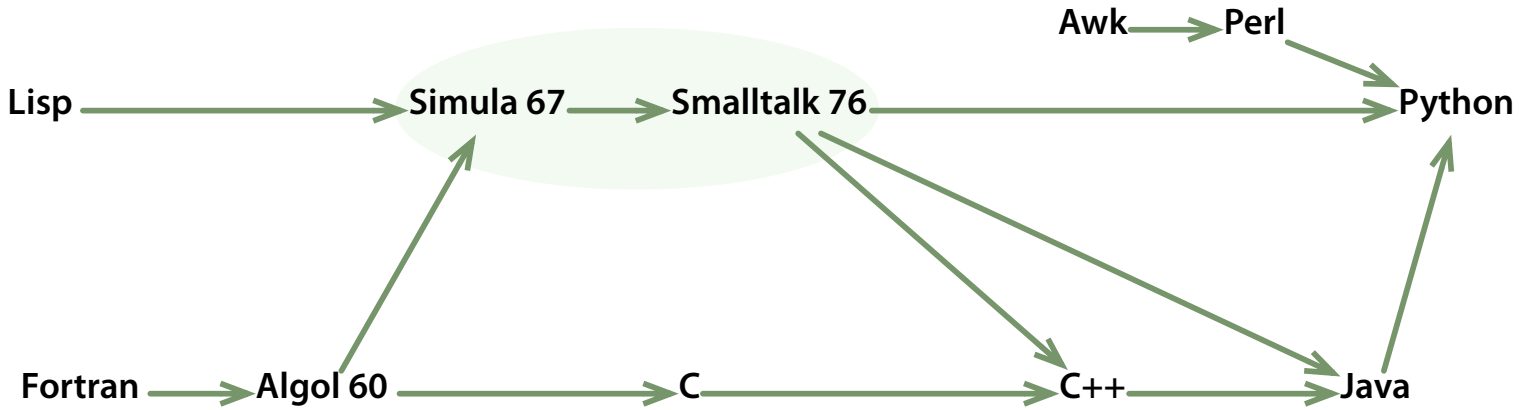
Class Hierarchies

Subclasses with inheritance and overriding

A bit of OO history ...



"Object-oriented" features (classes and objects) were developed in SmallTalk and spread to many other languages

Originally for simulation (Simula 67), then for modularity

Polymorphism

Poly = many

morph = form

: the quality or state of existing in or assuming different forms: such as
a (1) : existence of a species in several forms independent of the variations of sex
(2) : existence of a gene in several allelic forms; *also* : a variation in a specific DNA sequence
(3) : existence of a molecule (such as an enzyme) in several forms in a single species

(Merriam-Webster dictionary)

Polymorphism in method dispatch

same method call can work for several different kinds of data

Example: suppose we have a list of shapes, including triangles, squares, and circles. We want to add up their areas.

We don't want this:

```
total = 0;
for shape in shapelist:
    if is_square(shape) :
        total = total + square_area(shape)
    elif is_triangle(shape) :
        total = total + tri_area(shape)
    elif is_circle(shape):
        total = total + circ_area(shape)
```

Imagine we have a lot of code like this, and then we need to add support for ellipses. Yuck.

Better:

```
total = 0;
for shape in shapelist:
    total = total + shape.area()
```

There is still an area function for squares, and an area function for triangles, and one for circles ... but each one is a method in the corresponding class.

If we need to add a class for ellipses,
no change needed here!

We've localized some kinds of program changes, which is a GOOD THING (even worth putting up with wacky syntax)

You're already using polymorphism

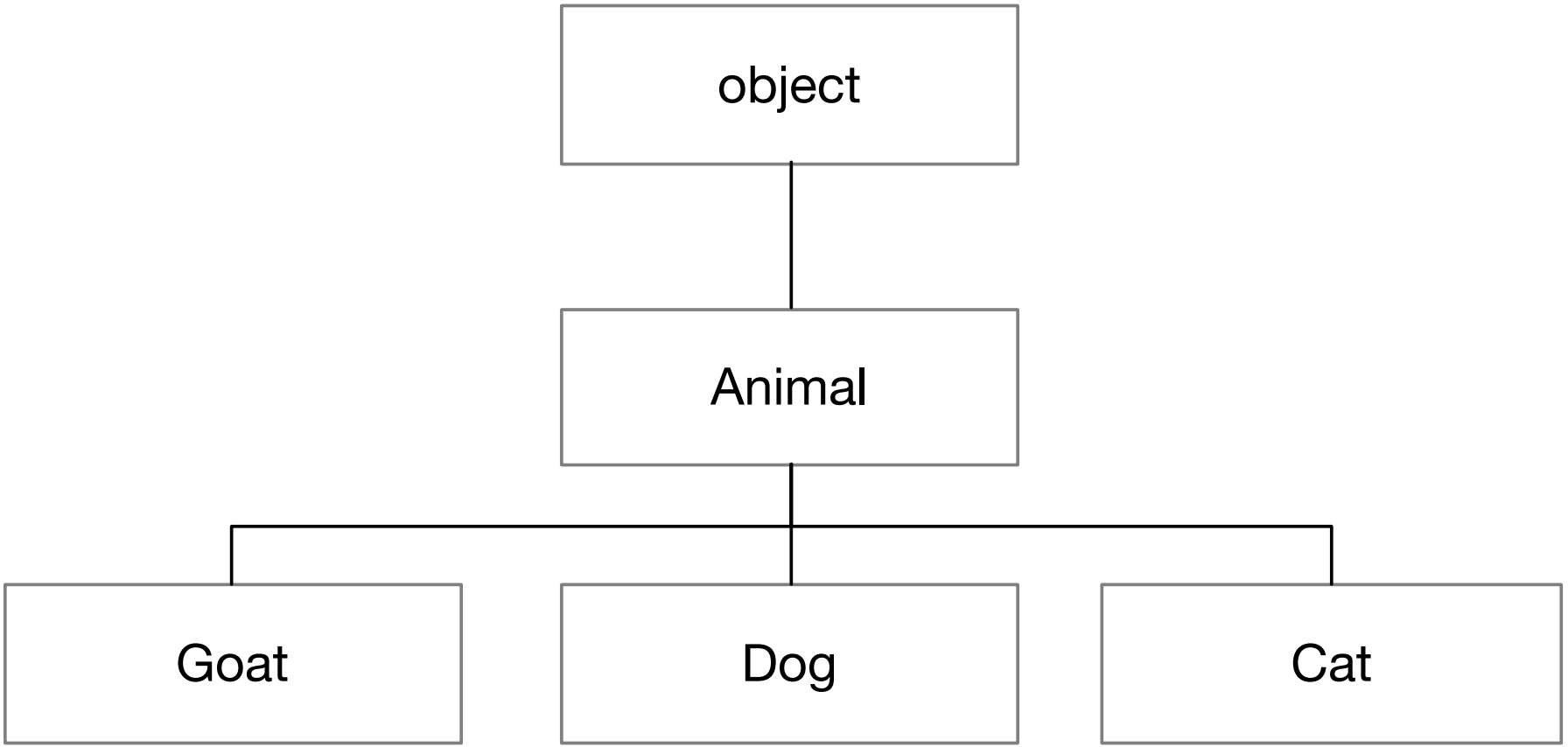
```
def ordered(ar):
    """Check that items are in ascending order.
    Args:
     ar:  list of items, all of the same comparable type
    Returns:
     True iff the items in ar are in ascending order.
    """
    if len(ar) == 0:
        return True
    prev = ar[0]
    for item in ar:
       # print("Comparing {} to {}".format(item,prev))
       if item < prev :
          return False
       prev = item
    return True
```

You're already using polymorphism

```
def ordered(ar):
    """Check that items are in ascending order.
    Args:
     ar:  list of items, all of the same comparable type
    Returns:
     True iff the items in ar are in ascending order.
    """
    if len(ar) == 0:
        return True
    prev = ar[0]
    for item in ar:
        # print("Comparing {} to {}".format(item,prev))
        if item < prev :
            return False
        prev = item
    return True
```

"x < y" is actually x.__lt__(y)

Let's build …



Each kind of animal will have its own sound ('arf', 'meow', etc)

(switch here to live coding)