LECTURE 5

OBJECT-ORIENTED PROGRAMMING

SUBCLASSES AND INHERITANCE

MCS 275 Spring 2023

Emily Dumas

LECTURE 5: SUBCLASSES AND INHERITANCE

Reminders and announcements:

- There will be no Homework 2
- Homework 3 to be posted Thurs, due Tue Jan 31
- Project 1 has new deadline: Fri Feb 10

IMPROVED POINT2 AND VECTOR2

I added new features to our `plane` module between lectures. These are explored a bit in Worksheet 3. Included:

- Can multiply `Vector2` by integer or float
- `abs(Vector2)` gives length
- *and more...*

INHERITANCE

Instead of starting a class definition from scratch we can indicate that it should **inherit** all the methods and attributes of some other class. Then we only need to specify the *changes*.

If new class `B` inherits from existing class `A` in this way, we say:

- `B` is a **subclass** of `A` (or *child* of `A`)
- `A` is a **superclass** of `B` (or *parent* of `B`)

WHY SUBCLASS?

Some common reasons:

- To change behavior of an existing class
  (e.g. a dict that only allows certain kinds of keys)
- To avoid duplication by moving common code to a superclass with several subclasses
- To formalize relationships between classes

Subclassing should express an "is-a" relationship. Dog and Cat might be subclasses of Pet.

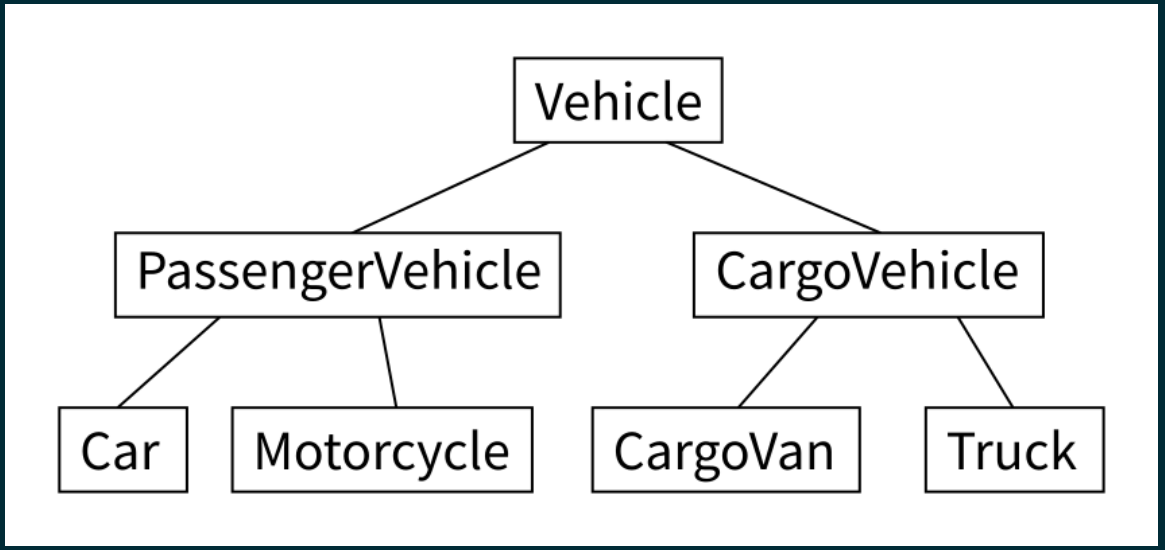
PYTHON SUBCLASS SYNTAX

Specify a class name to inherit from in the class definition:

```
class ClassName(SuperClassName):
    """Docstring of the subclass"""
    # ... subclass contents go here ...
```

Now, all the methods of the superclass are immediately available as part of `self`.

CLASS HIERARCHIES



Inheritance patterns are often shown in diagrams. Lines represent inheritance, with the superclass appearing above the subclass (usually).

LIVE CODING

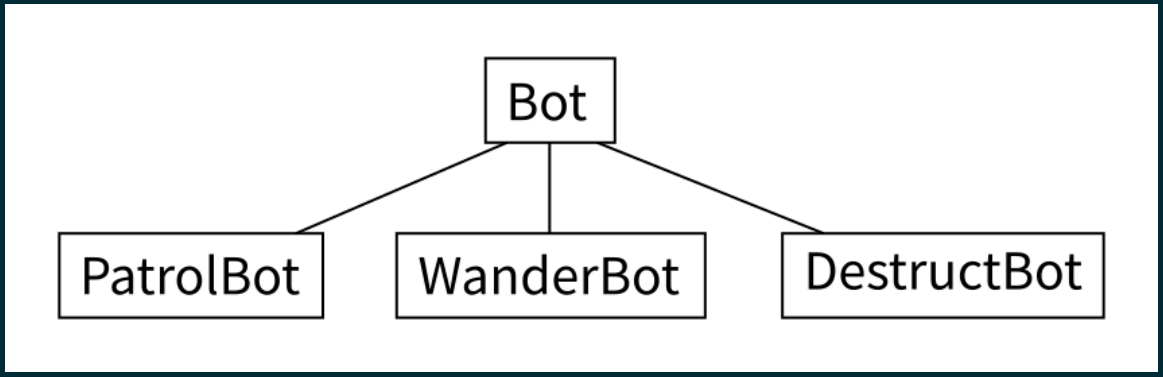
Let's build a class hierarchy for a simple robot simulation.

Every type of robot will be a subclass of `Bot`.

`Bot` has a `position` (a `Point`), boolean attribute `active`, and method `update()` to advance one time step.

Subclasses give the robot behavior (e.g. movement).

PLANNED BOT HIERARCHY



- `PatrolBot` walks back and forth.
- `WanderBot` walks about randomly.
- `DestructBot` sits in one place for a while and then deactivates.

ROBOT SIMULATION TEMPLATE

We haven't built any of the `Bot` subclasses yet, but I have already created:

- A start on module `bots` containing one class `Bot`. It sits in one place. In `bots.py` in the sample code repository.
- A script `botsimulation.py` to run the simulation and show it with simple text-based graphics.

SUPER()

If you define a method in a subclass, it *replaces* any method of the same name in the superclass.

That's usually very helpful. But what if the replacement wants to call the method it is replacing?

`super().method_name(...)` is the syntax for this; it calls `method_name(...)` of the superclass even if that method is redefined in the subclass.

WARNING

You only need `super()` under specific circumstances:

- Superclass and subclass have a method with the same name, and
- In the subclass, you need to call the superclass version of that method for some reason.

This is rare. More common: Needing to call a method of the superclass that isn't redefined in the subclass.

That's easier: Just use `self.method_name(...)`.

FROM

The `from` keyword can be used to import individual symbols from a module into the global scope.

```
import mymodule
# ...
mymodule.useful_function()  # module name needed
```

is equivalent to

```
from mymodule import useful_function
# ...
useful_function()  # no module name needed
```

Please use `from` very sparingly!

REFERENCES

- I discussed inheritance in MCS 260 Fall 2021 Lecture 27
- See *Lutz*, Chapter 31 for more discussion of inheritance.
- *Lutz*, Chapters 26-32 discuss object-oriented programming.

REVISION HISTORY

- 2022-01-21 Last year's lecture on this topic finalized
- 2023-01-25 Updated version for spring 2023