Announcements

- Homework
  - Remember that HW8 was postponed to be due next week!
- Be working on your Midterm projects/scripts
  - Due Friday night at midnight
- I'm trying desperately to get caught up on grading, but it has been an uphill battle
- I'm aiming to get updated grade reports pushed to WISE as soon as I can, but if you need to know where you are at before Friday so you can make an educated decision about C/NC, just let me know and I'll make you a priority.
- Polling: `rembold-class.ddns.net`

Review Question

Suppose you run the bit of code to the right. What would the printed output be?

A) `Spot is 8!`
B) `Dog(Spot,8)`
C) `Waffles is 6!`
D) This code would give an error.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __repr__(self):
        return f'Dog({self.name},{self.age})'
    def __str__(self):
        return f'{self.name} is {self.age}!'

A = Dog('Waffles', 6)
B = Dog('Spot', 8)
if str(A) == str(B):
    print(A)
else:
    print(B)
```

Defining vs Using

Defining the Class

- Implement a new object type with a class
  - define the class
  - define data attributes
    - What is the object?
  - Define methods
    - How to use the object?

Using the Class

- Using the new object type in the code
  - Create instances of the object type
  - Do operations with them

Type vs Instance

The Type

- Class name is the type
- Class is defined generically
  - Use `self` to refer to some instance
  - `self` is a parameter to our methods
- The class defines data and method common across all instances

The Instance

- Instance in one specific object
- Data attributes vary between instances
- Instance has the same structure as the class

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition
    - `print(a1.age)`

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition
    - `print(a1.age)`
  - We can alter data attributes from outside the class definition

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition
    - `print(a1.age)`
  - We can alter data attributes from outside the class definition
    - `a1.age = 5000`

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition
    - `print(a1.age)`
  - We can alter data attributes from outside the class definition
    - `a1.age = 5000`
  - We can create new data attributes from outside the class definition

Internal Representation

- Fundamental idea of types was internal representation
- We interact with the object through methods, the class takes care of manipulating the internal representation.
- Python is not great at enforcing this:
  - We can access data attributes from outside the class definition
    - `print(a1.age)`
  - We can alter data attributes from outside the class definition
    - `a1.age = 5000`
  - We can create new data attributes from outside the class definition
    - `a1.is_awesome = True`

Just because you CAN...

- In practice, it is generally a poor idea to do any of these things
- A programmer may need to change the internal representation to implement a new feature or fix a bug
  - If your program directly accessed that representation, it is probably now broken.
- Methods are our way of interacting with an object, so if you want to get or set a data attribute, write a method for it!
  - Commonly called getters and setters

Private and Protected

- It *is* possible to declare certain data attributes to be either private or protected
  - Really more a warning or recommendation that enforcement
  - Still possible to access and change those attributes, just a bit harder
  - Nothing is truly private in Python

Private and Protected

- It *is* possible to declare certain data attributes to be either private or protected
  - Really more a warning or recommendation that enforcement
  - Still possible to access and change those attributes, just a bit harder
  - Nothing is truly private in Python
- Private data attribute names prefixed with 2 underscores
  - `self.__myval = 5`
  - Only accessible (easily) within that specific class

Private and Protected

- It *is* possible to declare certain data attributes to be either private or protected
  - Really more a warning or recommendation that enforcement
  - Still possible to access and change those attributes, just a bit harder
  - Nothing is truly private in Python
- Private data attribute names prefixed with 2 underscores
  - `self.__myval = 5`
  - Only accessible (easily) within that specific class
- Protected data attribute names prefixed with a single underscore
  - `self._myval = 5`
  - Accessible within that class and future sub-classes

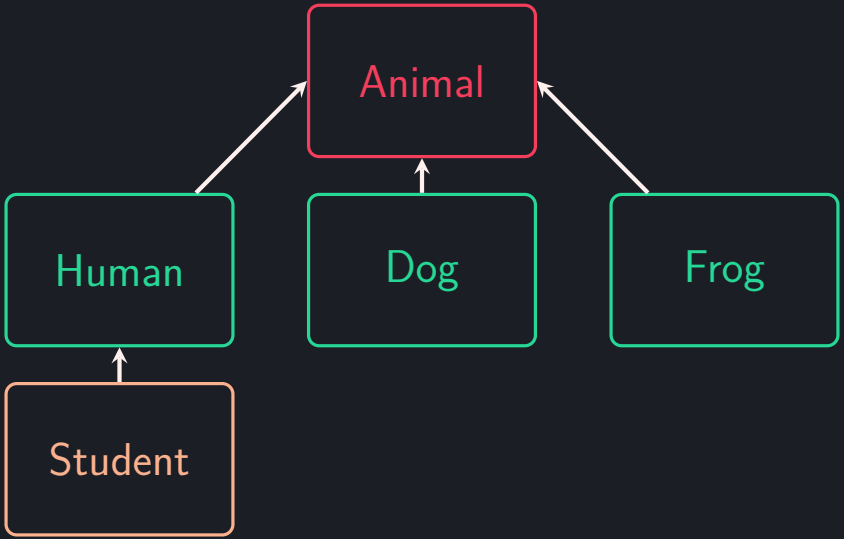
More a set of guidelines really...

- Setting attributes to be private or protected isn't a guarantee that they can't be viewed, accessed, or modified!
- Think of it more as a reminder to future you (or other coders) if certain values should be allowed to be accessed outside the class definition.

Inheritance

- Classes are already about grouping similar types of objects together
- Groups of objects are frequently related to other groups of objects
  - Either as a subset or a superset
- We can mimic and take advantage of these relationships in our classes!

Hierarchy Example



Hierarchy Terms

- Arrows point toward the parent class or superclass
- Arrows point from a child class or subclass
  - Inherit all data attributes and methods from parent class
  - Can have new attributes added
  - Can have new methods added
  - Can have parents methods overridden
- Children specify who their parents are (not vice versa)

Name your parent (in code)

- We specify what a class's parent is immediately following the class's name
  - Surround parent's name is parentheses
  - `class Human(Animal):`
  - The default parent is type `object`