More on Python Class and Inheritance

Professor Andrew Chapkowski

Recall Implementing vs Using Classes

- Implementing a new object type with a class
  - Define the class
  - Define the attributes
  - Define the methods

- Using the new object
  - Create the instance
  - Do the operations

Why use OOP and Classes?

- Mimic real life
- Group different object parts of the same type

Getter/Setter Methods

- **getters and setters** should be used outside of class to access data attributes
- Getters retrieve information
- Setter set an attribute

Getter/Setter Continued

```
class Human(object):
	def __init__(self, age):
		self.age = age
	@property
	def age(self):                          Getter
		return self.age
	@age.setter
	def age(self, value:int):               Setter
		self.age = value
```

Usage of Getter/Setter

- Like methods and attributes, use the “.” notation

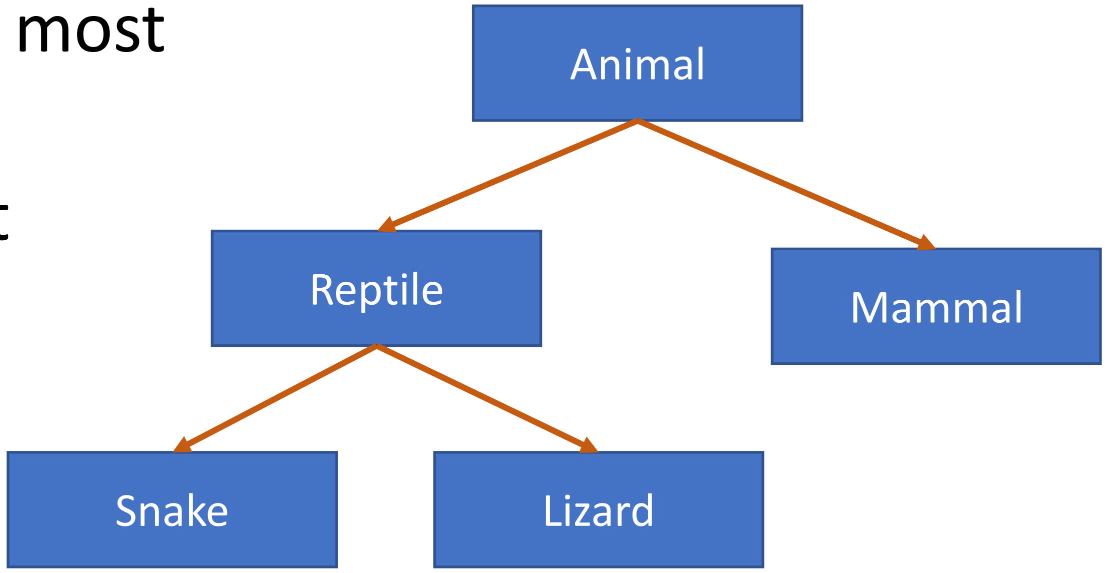
```
>>> h = Human()
>>> print(h.age)
24
```

Hiding Information

- When defining a class you might want to prevent data from being changed
- **Python does not have the concept of PRIVATE**
- Use a “_” in front of variable to say this is private
- Use two underscores “__” to say never touch this

Hierarchies

- **Parent classes** – these are the top most classes
- **Child classes** – inherit from parent
  - Add functionality
  - Have different behavior
  - Override behavior



Parent Class Example

```
class OperaSinger(Human):
	def speak(self):
		print(‘hello opera’)
	def sing(self):
		print(“LA La LA”)
```

Which Methods to Use?

- Subclasses can have **methods** with the same name
- If a method is not found it moves up the **hierarchy**
- If we call **age** on OperaSinger, it looks up the **inheritance hierarchy** and calls the **getter** from **Human**

OOP Overview

- create your own **collections of data**
- **organize** information
- **division of work**
- access information in a **consistent manner**
- add **layers** of complexity
- like functions, classes are a mechanism for **decomposition and abstraction** in programming