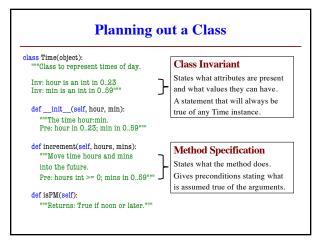
#### What Does **str()** Do On Objects? Does **NOT** display contents class Point3(object): """Class for points in 3d space""" >>> p = Point3(1,2,3)>>> str(p) def \_\_str\_\_(self): '<Point3 object at 0x1007a90>' """Returns: string with contents"" Must add a special method return '('+str(self.x) + ',' + \_\_str\_\_ for str() str(self.y) + ',' + str(self.z) + ')' \_\_repr\_\_ for repr() · Could get away with just one def \_\_repr\_\_(self): repr() requires \_\_repr\_\_ ""Returns: unambiguous string"" return str(self.\_\_class\_\_)+ str() can use \_\_repr\_ (if \_\_str\_\_ is not there) str(self)

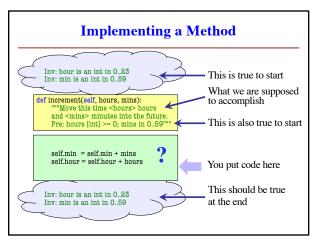
# Making a Class into a Type

- 1. Think about what values you want in the set
  - What are the attributes? What values can they have?
- 2. Think about what operations you want
  - This often influences the previous question
- To make (1) precise: write a *class invariant*
- Statement we promise to keep true after every method call
- To make (2) precise: write *method specifications* 
  - Statement of what method does/what it expects (preconditions)
- Write your code to make these statements true!

1 2



3



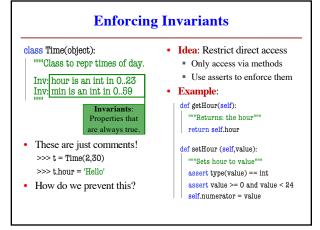
**Enforce Method Preconditions with assert** class Time(object): Inv: hour is an int in 0..23 ""Class to represent times of day.""" def \_\_init\_\_(self, hour, min): """The time hour:min. Pre: hour in 0..23; min in 0..59" Initializer creates/initializes all assert type(hour) == int of the instance attributes assert 0 <= hour and hour < 24 assert type(min) == int assert  $0 \le \min$  and  $\min \le 60$ Asserts in initializer guarantee the initial values satisfy the invariant. def increment(self, hours, mins): """Move this time <hours> hours and <mins> minutes into the future Pre: hours is int >= 0; mins in 0..59 assert type(hour) == int Asserts in other methods enforce assert type (min) == int the method preconditions. assert hour >= 0 assert 0 <= min and min < 60

5 6

1

4

#### **Hiding Methods From Access** class Time(object): Hidden methods "Class to represent times of day. start with an underscore Inv: hour is an int in 0..23 Inv: min is an int in 0..59"" do not show up in help() are meant to be internal (e.g. helper methods) def is minute(self.m): But they are not restricted """Return: True if m valid minute"" ■ You can still access them return (type(m) == int and m >= 0 and m < 60) But this is bad practice! Like a precond violation \_init\_\_(self, hour, min): · Can do same for attributes "The time hour:min. Pre: hour in 0..23; min in 0..59"" Underscore makes it hidden assert self.\_is\_minute(m) Only used inside of methods Helper



8

# **Data Encapsulation** • **Idea**: Force the user to only use methods

- Do not allow direct access of attributes

# **Setter Method**

## • Used to change an attribute

- Replaces all assignment statements to the attribute
- Bad:

9

- >>> t.hour = 5
- Good: >>> f.setHour(5)

## Getter Method

- · Used to access an attribute
- Replaces all usage of attribute in an expression
- Bad:
- >>> x = 3\*t.hour
- Good:
- >>> x = 3\*t.getHour()

class Time(object): """Class to repr times of day. Getter def getHour (self): """Returns: hour attribute" return self.\_hour Setter def setHour(self, h): " Sets hour to h Pre: h is an int in 0..23""" assert type(h) == intassert  $0 \le h$  and  $h \le 24$ self.\_hour = d

**Data Encapsulation NO ATTRIBUTES** in class specification Method specifications describe the attributes Setter precondition is same as the **invariant** 

10

### **Encapsulation and Specifications** class Time(object): No attributes """Class to represent times of day. """ in class spec ### Hidden attributes These comments # Att \_hour: hour of the day make it part of the # Inv: \_hour is an int in 0..23 class invariant but not part of the # Att \_min: minute of the hour (public) interface # Inv: \_min is an int in 0..59 These comments do not go in help()

**Mutable vs. Immutable Attributes** Mutable **Immutable** · Can change value directly · Can't change value directly If class invariant met May change "behind scenes" ■ Example: turtle.color **Example:** turtle.x Has both getters and setters · Has only a getter Setters allow you to change No setter means no change Enforce invariants w/ asserts Getter allows limited access May ask you to differentiate on the exam

11 12