Designing Types

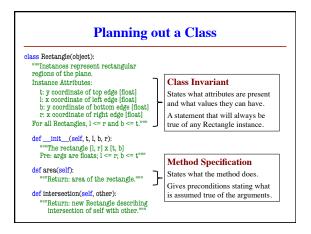
From first day of class!

- Type: set of values and the operations on them
 - int: (set: integers; ops: +, -, *, /, ...)
 - Time (set: times of day; ops: time span, before/after, ...)
 - Worker (set: all possible workers; ops: hire,pay,promote,...)
 - Rectangle (set: all axis-aligned rectangles in 2D;
 ops: contains, intersect, ...)
- To define a class, think of a *real type* you want to make
 - Python gives you the tools, but does not do it for you
 - Physically, any object can take on any value
 - Discipline is required to get what you want

Making a Class into a Type

- 1. Think about what values you want in the set
 - What attributes? What values can they have?
- 2. Think about what operations you want
 - 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!

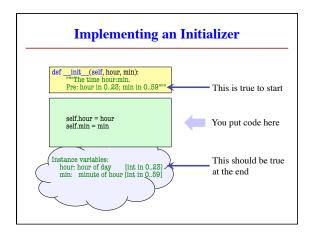
Planning out a Class class Time(object): Class Invariant ""Instances represent times of day Instance Attributes: States what attributes are present hour: hour of day [int in 0..23] and what values they can have. min: minute of hour (int in 0..59) A statement that will always be true of any Time instance def __init__(self, hour, min): """The time hour:min. Pre: hour in 0..23; min in 0..59"" def increment(self, hours, mins): Method Specification """Move this time <hours> hours and <mins> minutes into the future. States what the method does Pre: hours is int >= 0; mins in 0..59""" Gives preconditions stating what is assumed true of the arguments. def isPM(self): """Returns: this time is noon or later."""

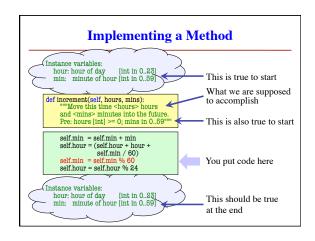


Planning out a Class class Hand(object): Class Invariant ""Instances represent a hand in cards. Instance Attributes: States what attributes are present cards: cards in the hand [list of card] and what values they can have. This list is sorted according to the ordering defined by the Card class. A statement that will always be true of any Rectangle instance def init (self, deck, n): """Draw a hand of n cards. Pre: deck is a list of >= n cards""" Method Specification def isFullHouse(self): """Return: True if this hand is a full house: False otherwise""" States what the method does Gives preconditions stating what def discard(self, k): is assumed true of the arguments. ""Discard the k-th card.""

Implementing a Class

- All that remains is to fill in the methods. (All?!)
- When implementing methods:
 - 1. Assume preconditions are true
 - 2. Assume class invariant is true to start
 - 3. Ensure method specification is fulfilled
 - 4. Ensure class invariant is true when done
- Later, when using the class:
 - When calling methods, ensure preconditions are true
 - If attributes are altered, ensure class invariant is true





Role of Invariants and Preconditions

- · They both serve two purposes
 - Help you think through your plans in a disciplined way
 - Communicate to the user* how they are allowed to use the class
- Provide the interface of the class
 - interface btw two programmers
 - interface btw parts of an app
- Important concept for making large software systems
 - Will return to this idea in a week
- * ...who might well be you!

n•ter•face |lintərifās| noun

- a point where two systems, subjects, organizations, etc., meet and interact: the interface between accountancy and the law
- chiefly Physics a surface forming a common boundary between two portions of matter or space, e.g., between two immiscible liquids: the surface tension of a liquid at its air/ liquid interface.
- Computing a device or program enabling a user to communicate with a computer.
- a device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other.
 - -The Oxford American Dictionary

Implementing a Class

- All that remains is to fill in the methods. (All?!)
- When implementing methods:
 - 1. Assume precondit
 - 2. Assume class inv Easy(ish) if we are the user.
 - 3. Ensure method s But what if we aren't?
 - 4. Ensure class invariant when done
- Later, when using the class:
 - When calling methods, ensure preconditions are true
 - If attributes are altered, ensure class invariant is true

Enforce Method Preconditions with assert class Time(object): Instance Attributes: "Instances represent times of day."" hour: hour of day [int in 0..23] min: minute of hour [int in 0..59] def init (self, hour, min): """The time hour:min. Pre: hour in 0..23; min in 0..59" assert type(hour) == int assert 0 <= hour and hour < 24 Initializer creates/initializes all of the instance attributes. ${\tt assert\ type(min) == int}$ Asserts in initializer guarantee the assert $0 \le \min$ and $\min < 60$ 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 assert type (min) == int Asserts in other methods enforce assert hour >= 0 and assert 0 <= min and min < 60 the method preconditions

