Designing types

- One definition of a type: a set of objects with the operations on those objects.
  - 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, …

- When you define a class, you are (should be) thinking of a “real type” you want to create.
  - Python gives you the tools to do this, but doesn’t do it for you.
  - Physically, any object can take on any value.
  - Discipline is required to get what you want.

Making a class a “real 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.
   - Sometimes influences the previous question.
   - To make (1) precise: write a class invariant.
     - A statement we promise ourselves to keep true after every method call.
   - To make (2) precise: write specifications of methods.
     - A statement of what the method does and what it expects (preconditions).
   - Write your code to make these statements true!

Planning out a class

time = Time(hour, min)

def increment(time, hours, mins):
  return time + Time(hours, mins)

def is_pm(time):
  return time.hour % 12 == 0

class Time:
  pass

Planning out a class

def is_full_house(hand):
  return len(set(card.rank) for card in hand)) == 3

def discard(hand, k):
  return hand[:k] + hand[k+1:]

class Hand:
  pass

Planning out a class

def area(rectangle):
  return (rectangle.right - rectangle.left) * (rectangle.top - rectangle.bottom)

def intersection(rectangle1, rectangle2):
  top = max(rectangle1.top, rectangle2.top)
  bottom = min(rectangle1.bottom, rectangle2.bottom)
  left = max(rectangle1.left, rectangle2.left)
  right = min(rectangle1.right, rectangle2.right)
  return Rectangle(left, top, right, bottom)

class Rectangle:
  pass

Implementing a class

- All that remains is to fill in the methods. (All?!)"
Implementing the Time initializer

```
def __init__(self, hour, min):
    self.hour = hour
    self.min = min
```

This is true to start

You put code here

This should be true at the end

Implementing the Rectangle initializer

```
def __init__(self, l, r, b, t):
    self.l = l
    self.r = r
    self.b = b
    self.t = t
```

This is true to start

You put code here

This should be true at the end

Implementing a Card initializer

```
rank = ' A23456789TJQK'.index(code[0])
suit = ' CDHS'.index(code[1])
```

This is true to start

You put code here

This should be true at the end

Implementing a Time method

```
self.min += min
self.hour = (self.hour + hours).min % 24
```

This is true to start

What we are supposed to accomplish

This is also true to start

You put code here

This should be true at the end

The view from outside

- Invariants and preconditions serve two purposes
  - They are tools for you, as the author, to think through your plans in a disciplined way
  - They communicate to the user* of the class how they are allowed to use it
  - Together they are the interface of the class
    - interface between two programmers
    - interface between two parts of the program
  
*...who might well be you!

Interface 

1. a point where two systems, subjects, organizations, etc., meet and interact; the interface between accountancy and the law.
2. 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

Acquiring more “real type” behavior

- Want two different objects to be ==?
  - define __eq__(self, other)
    - true if and only if self == other
  - define __ne__(self, other)
    - true if and only if self != other

- Want your type to be ordered?
  - define __gt__(self, other)
    - positive if self > other
  - define __lt__(self, other)
    - negative if self < other
  - define __ge__(self, other)
    - true if and only if self >= other
  - define __le__(self, other)
    - true if and only if self <= other

*a is 5" means that a and b are the same object (same ID).
*Cannot be customized.
*Opposite: “a is not b” Use for “is None”.
*a == b" means that a and b have the same value.
*Can be customized by defining special methods.
*Opposite: “a != b"