Designing Types

**Type:** set of values and the operations on them
- int: set; integers; ops: +, -, *, /, ...
- Time: set; 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 are the attributes? What values can they have?
2. Think about what operations you want
   - This often influences the previous question
3. To make (1) precise: write a class invariant
   - Statement we promise to keep true after every method call
4. To make (2) precise: write method specifications
   - Statement of what method does and what it expects (preconditions)
5. Write your code to make these statements true!

Planning out a Class

**Instance Attributes:**
- 2.
- 1.

**Clas s Invariant**
- States what attributes are present and what values they can have.
- A statement that will always be true of any Time instance.

**Method Specification**
- States what the method does.
- Gives preconditions stating what is assumed true of the arguments.

Implementing an Initializer

```python
def __init__(self, hour: int, minute: int):
    """The time is noon or later."""
    # Put code here
    self.hour = hour
    self.minute = minute
```

Implementing a Method

```python
def increment(self, hours: int, minutes: int):
    """Move this time <hours> hours and <minutes> minutes into the future.
    Pre: hour in 0..23; min in 0..59""
    # Put code here
    self.hour = self.hour + hours + minutes / 60
    self.minute = self.minute + minutes
```

Rectangle

```python
class Rectangle(object):
    """Instances represent rectangular regions of the plane.
    Instance Attributes:
    b: y coordinate of top edge (float)
    l: x coordinate of left edge (float)
    r: x coordinate of right edge (float)
    t: y coordinate of bottom edge (float)
    ""
    # Put code here
    self.l = l
    self.r = r
    self.t = t
    self.b = b
```
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 parts of an app
- Important concept for making large software systems
- Will return to this idea later
  - ...who might well be you!

Instance attributes:

- How do we prevent this?

* These are just comments!

```python
>>> p = Fraction(1, 2)
>>> p.numerator = 'Hello'
```

Enforce Method Preconditions with `assert`

```python
class Time(object):
    """Instances represent times of day"""
    hour = int
    min = int

    def __init__(self, hour, min):
        """The time constructor. Pre: hour in 0..23; min in 0..59"""
        assert hour >= 0 and hour < 24
        assert type(hour) == int
        assert 0 <= min and min < 60
        assert type(min) == int
        self.hour = hour
        self.min = min

    def increment(self, hours, mins):
        """Move this time <hours> hours and <mins> minutes into the future. Pre: hour in 0..23; min in 0..59""
        assert type(hours) == int
        assert type(mins) == int
        assert hour >= 0 and hour < 24
        assert type(hour) == int
        assert hour >= 0 and self.min + mins < 60
        self.hour += hours
        self.min += mins
```

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
- Example: `t.color = 'red'`

**Getter Method**

- Used to access an attribute
- Replaces all usage of attribute in an expression
- Example: `t.color`

Mutable vs. Immutable Attributes

<table>
<thead>
<tr>
<th>Mutable</th>
<th>Immutable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value can change directly</td>
<td>Value can’t change directly</td>
</tr>
<tr>
<td>Change must meet invariant</td>
<td>May change “behind scenes”</td>
</tr>
<tr>
<td>Example: <code>t.color = 'red'</code></td>
<td>Example: <code>t.x = 5</code></td>
</tr>
<tr>
<td>To implement</td>
<td>To implement</td>
</tr>
<tr>
<td>* Hide the attribute with <code>_</code></td>
<td>* Hide the attribute with <code>_</code></td>
</tr>
<tr>
<td>* Implement getter</td>
<td>* Implement getter</td>
</tr>
<tr>
<td>* Implement setter w/ asserts</td>
<td>* DO NOT implement a setter</td>
</tr>
</tbody>
</table>