Announcements

• A3 due tonight at 11:59pm.
• Spring break next week:
  - No office hours
  - No consulting hours
  - Limited piazza

Designing Types

• 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

Case Study: Fractions

• Want to add a new type
  - Values are fractions: ½, ¾
  - Operations are standard multiply, divide, etc.
  - Example: ½ * ¾ = ⅜
• Can do this with a class
  - Values are fraction objects
  - Operations are methods
• Example: simplefrac.py

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!

Planning out a Class

class Time(object):
    """Instances represent times of day."
    """Instance Attributes:
hour: hour of day [int in 0..23]
min: minute of hour [int in 0..59]"
    def __init__(self, hour, min):
        self.hour = hour
        self.min = min
    def increment(self, hours, mins):
        self.hour += hours
        self.min += mins
        if self.min >= 60:
            self.min -= 60
            self.hour += 1
        if self.hour >= 24:
            self.hour -= 24
    def isPM(self):
        """Returns: this time is noon or later."""

Class 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.

Planning out a Class

class Rectangle(object):
    """Instances represent rectangular regions of the plane."
    """Instance Attributes:
t: y coordinate of top edge [float]
l: x coordinate of left edge [float]
b: y coordinate of bottom edge [float]
r: x coordinate of right edge [float]
For all Rectangles, l <= r and b <= t"
    def __init__(self, t, l, b, r):
        self.t = t
        self.l = l
        self.b = b
        self.r = r
    def area(self):
        """Return: area of the rectangle."""
        return (self.r - self.l) * (self.t - self.b)
    def intersection(self, other):
        """Return: new Rectangle describing intersection of self with other."""

Class Invariant

States what attributes are present and what values they can have.
A statement that will always be true of any Rectangle 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, min):
    """The time hour:min.
    Pre: hour in 0..23; min in 0..59"
    self.hour = hour
    self.min = min
```

This is true to start

You put code here

This should be true at the end

You put code here

Implementing a Method

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

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

Data Encapsulation

```python
class Fraction(object):
    """Instance attributes:
    numerator: top [int]
    denominator: bottom [int > 0]"
    def __init__(self,n=0,d=1):
        assert self._is_denominator(d)
        self.numerator = n
        self.denominator = d
```

Hiding Methods From Access

- Put underscore in front of a method will make it hidden
  - Will not show up in help()
  - But it is still there…
  - Hidden methods
    - Can be used as helpers inside of the same class
      - But it is bad style to use them outside of this class
    - Can do same for attributes
      - Underscore makes it hidden
      - Do not use outside of class

```python
class Fraction(object):
    """Instance attributes:
    num: top       [int]
    den: bottom [int > 0]"
    def is_denominator(self,d):
        """Return: True if d valid denom"
        return type(d) == int and d > 0
    def __init__(self,n=0,d=1):
        assert self._is_denominator(d)
        self.numerator = n
        self.denominator = d
```

Example: Converting Values to Strings

```python
class Point3(object):
    """Instances are points in 3d space"
    ...
    def __str__(self):
        """Returns: string with contents"
        return "('Point3 object at 0x1007a90')"
    def __repr__(self):
        """Returns: unambiguous string"
        return str(self.__class__) + str(self)
```

```
"""Example: Converting Values to Strings"

<table>
<thead>
<tr>
<th>str() Function</th>
<th>Backquotes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usage:</strong> str(expression)</td>
<td><strong>Usage:</strong> <code>expression</code></td>
</tr>
<tr>
<td>* Evaluates the expression</td>
<td>* Evaluates the expression</td>
</tr>
<tr>
<td>* Converts it into a string</td>
<td>* Converts it into a string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How does it convert?</th>
<th>How does it convert?</th>
</tr>
</thead>
<tbody>
<tr>
<td>str(2) → '2'</td>
<td>'2' → '2'</td>
</tr>
<tr>
<td>str(True) → 'True'</td>
<td>'True' → 'True'</td>
</tr>
<tr>
<td>str('True') → 'True'</td>
<td>'True' → 'True'</td>
</tr>
<tr>
<td>str(Point3()) → '(0.0,0.0,0.0)'</td>
<td>'Point3()' → 'class 'Point3'&gt; (0.0,0.0,0.0)'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What Does str() Do On Objects?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Does NOT</strong> display contents</td>
</tr>
</tbody>
</table>
| >>> p = Point3(1,2,3) | >>> str(p)
| 'Point3 object at 0x1007a90' |

<table>
<thead>
<tr>
<th>Must add a special method</th>
</tr>
</thead>
<tbody>
<tr>
<td>* <em>str</em> for str()</td>
</tr>
<tr>
<td>* <em>repr</em> for backquotes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Could get away with just one</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Backquotes require <em>repr</em></td>
</tr>
<tr>
<td>* str() can use <em>repr</em> if <em>str</em> is not there</td>
</tr>
</tbody>
</table>
```

```python
class Point3(object):
    """Instances are points in 3d space"
    ...
    def __str__(self):
        """Returns: string with contents"
        return '<Point3 object at 0x1007a90>'
    def __repr__(self):
        """Returns: unambiguous string"
        return str(self.__class__) + str(self)
```