A Closer Look at MeetMe

Classes and Objects and Methods, Oh My

Recall ...

<table>
<thead>
<tr>
<th>Potential meeting times</th>
<th>Kevin's available times</th>
<th>Mary's available times</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00 to 9:00 am</td>
<td>8:30 to 9:30 am</td>
<td>8:30 to 9:30 am</td>
</tr>
<tr>
<td>9:00 to 10:00 am</td>
<td>9:30 to 10:30 am</td>
<td>9:30 to 10:30 am</td>
</tr>
<tr>
<td>10:00 to 11:00 am</td>
<td>10:30 to 11:30 am</td>
<td>10:30 to 11:30 am</td>
</tr>
<tr>
<td>11:00 to 12:00 noon</td>
<td>11:30 to 12:30 noon</td>
<td>11:30 to 12:30 noon</td>
</tr>
<tr>
<td>12:00 to 1:00 pm</td>
<td>12:30 to 1:30 pm</td>
<td>12:30 to 1:30 pm</td>
</tr>
<tr>
<td>1:00 to 2:00 pm</td>
<td>1:30 to 2:30 pm</td>
<td>1:30 to 2:30 pm</td>
</tr>
<tr>
<td>2:00 to 3:00 pm</td>
<td>2:30 to 3:30 pm</td>
<td>2:30 to 3:30 pm</td>
</tr>
<tr>
<td>3:00 to 4:00 pm</td>
<td>3:30 to 4:30 pm</td>
<td>3:30 to 4:30 pm</td>
</tr>
<tr>
<td>4:00 to 5:00 pm</td>
<td>4:30 to 5:30 pm</td>
<td>4:30 to 5:30 pm</td>
</tr>
<tr>
<td>5:00 to 6:00 pm</td>
<td>5:30 to 6:30 pm</td>
<td>5:30 to 6:30 pm</td>
</tr>
</tbody>
</table>

Creating an Appt: Constructor

```python
import datetime
class Appt:
    def __init__(self, day, begin, end, desc):
        """Create an appointment on date from begin time to end time."

        Arguments:
        day: A datetime.date object
        begin: A datetime.time object
        end: A datetime.time object,
             after begin
        desc: A string describing the appointment

        ...""
        self.day = day
        self.begin = begin
        self.end = end
        self.desc = desc
        return
```

Modules

```
meetme.py
    main
    program logic

agenda.py
    class Appt:
    class Agenda:
    unit tests
    if __name__ == "__main__":
```

Apt object

Agenda objects
The constructor is a bit of a pain ...

```python
sample = Appt(datetime.date(2012, 10, 31),
            datetime.time(14, 30),
            datetime.time(15, 45),
            "Sample appointment")
```

We can make it easier with a “factory” to create Appt objects
from text like "2012.10.31 14:30 15:45 | Sample meeting"

```python
Exceptions

@classmethod
def from_string(cls, txt):
    """Factory parses a string to create an Appt""
    fields = txt.split("|")
    if len(fields) != 2:
        raise ValueError("Appt literal requires exactly one '|' before description")
    fields = appt_date_text.split(".")
    try:
        year = int(fields[0].strip())
        month = int(fields[1].strip())
        day = int(fields[2].strip())
    except:
        raise ValueError("Date in Appt literal should be 9999.99.99 (Year.Month.Day)")
```

Creating an Appt from text: Factory method

```python
Class method is called on Appt class, not on Appt object.
Factory: Class method to indirectly invoke constructor
Example:
an_appt = Appt.from_string(
    "2012.10.31 14:30 15:45 | Team mtg")
```

Special methods

```
Operation Implemented by
appt1 < appt2 def __lt__(self, other):
appt1 > appt2 def __gt__(self, other):
str(appt) def __str__(self):
```

Keyword (default) arguments

```python
def intersect(self, other, desc=""):  
    """Return an appointment representing the period in common between this appointment and another. Requires self.overlaps(other)."

    Arguments:
        other: Another Appt
desc: (optional) description text for this appointment.

    Returns:
        An appointment representing the time period in common between self and other. Description of returned Appt is copied from this (self), unless a non-null string is provided as desc.

    """
    if desc="":
        desc = self.desc
```

The Agenda class “wraps” a list of Appt

```python
class Agenda:
    """An Agenda is essentially a list of appointments, with some agenda-specific methods.
    """
    def __init__(self):
        """An empty agenda.""
        self.appts = [
    def append(self, appt):
        """Add an Appt to the agenda.""
        self.appts.append(appt)
    def __len__(self):
        """Number of appointments, callable as built-in len() function""
        return len(self.appts)
```

More special methods

<table>
<thead>
<tr>
<th>Operation</th>
<th>Implemented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(agenda)</td>
<td>def <strong>len</strong> (self):</td>
</tr>
<tr>
<td>for appt in agenda:</td>
<td>def <strong>iter</strong>(self):</td>
</tr>
</tbody>
</table>

In Appt class, these just wrap corresponding list methods

Agenda adds new Appt-specific methods

```python
class Agenda:
    ...
@classmethod
def from_file(cls, f):
    """Factory: Read an agenda from a file. ..."
    ...
```

```python
def intersect(self, other, desc=""):  
    """Return a new agenda containing appointments that are overlaps between appointments in this agenda and appointments in the other agenda."
    """
```
Testing __name__

```python
from test_harness import *
import io

def selftest_appt():
    """Simple smoke test for Appt class."""
    sample = Appt(datetime.date(2012, 10, 31),
                   datetime.time(14, 30), datetime.time(15, 45),
                   "Sample appointment")
    testEQ("Create and format", str(sample),
            "2012.10.31 14:30 15:45 | Sample appointment")
...

if __name__ == "__main__":
    selftest_appt()
    selftest_agenda()
```

What does the main program do?

- Read and interpret the command line
- Open each agenda file
- Create Agenda objects to do the main work
  - Create starting “available” block
  - Intersect with each personal agenda file
- Print results