String Representations

An object value should behave like the kind of data it is meant to represent. For instance, by producing a string representation of itself:

Strings are important: they represent language and programs. In Python, all objects produce two string representations:
- The str is legible to humans
- The repr is legible to the Python interpreter

The str and repr strings are often the same, but not always.

Some objects do not have a simple Python-readable string:
```
>>> repr(min)
'<built-in function min>'
```

**Implementing repr and str**

The behavior of repr is slightly more complicated than invoking __repr__ on its argument:
- An instance attribute called __repr__ is ignored if class attributes are found
- Questions: How would we implement this behavior?

The behavior of str is also complicated:
- An instance attribute called __str__ is ignored
- If no __str__ attribute is found, uses repr string
- (By the way, str is a class, not a function)

**Polymorphic Functions**

Polymorphic functions: A function that applies to many (poly) different forms (morph) of data
str and repr are both polymorphic; they apply to any object

The repr function returns a Python expression (a string) that evaluates to an equal object.
```
>>> repr(min)
'<built-in function min>'
```

The str function invokes a zero-argument method __str__ on its argument.
```
>>> str(min)
'1/2'
```

**Message passing**

Objects interact by looking up attributes on each other (passing messages)

The attribute lookup rules allow different data types to respond to the same message.

A shared message (attribute name) that elicits similar behavior from different object classes is a powerful method of abstraction.

An interface is a set of shared messages, along with a specification of what they mean.

Examples:
- Classes that implement __repr__ and __str__ methods that return human-readable and computer-readable strings implement an interface for producing string representations.
Special Method Names in Python

Certain names are special because they have built-in behavior:

These names always start and end with two underscores:

- __init__ - Method invoked automatically when an object is constructed
- __repr__ - Method invoked to display an object as a Python expression
- __add__ - Method invoked to add one object to another
- __bool__ - Method invoked to convert an object to True or False
- __float__ - Method invoked to convert an object to a float (real number)

Example:
```python
>>> zero, one, two = 0, 1, 2
>>> one + two
3
>>> bool(zero), bool(one)
(False, True)
```

Adding instances of user-defined classes invokes either the __add__ or __radd__ method:

```python
>>> Ratio(1, 3) + Ratio(1, 6)
Ratio(1, 2)
```

Generic Functions

A polymorphic function might take two or more arguments of different types:

- Type Dispatching: Inspect the type of an argument in order to select behavior
- Type Coercion: Convert one value to match the type of another

Example:
```python
>>> Ratio(1, 3) + 1
Ratio(4, 3)
>>> from math import pi
>>> Ratio(1, 3) + pi
3.4749259869231266
```

Separation of Concerns

A design principle: Isolate different parts of a program that address different concerns:

- A modular component can be developed and tested independently

<table>
<thead>
<tr>
<th>Bag</th>
<th>Bug Game Simulator</th>
<th>Player Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bug Game Simulator</td>
<td>Player Strategies</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>Action</td>
</tr>
<tr>
<td></td>
<td>Movement</td>
<td>Movement</td>
</tr>
<tr>
<td></td>
<td>User Interface</td>
<td>User Interface</td>
</tr>
</tbody>
</table>

Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

```json
[{
  "business_id": "gclB3ED6uk6viWlolSb_uA",
  "name": "Cafe 3",
  "stars": 2.0,
  "price": 1,
  ...}
{
  "business_id": "WXKx2I2SEzBpeUGtDMCS8A",
  "name": "La Cascada Taqueria",
  "stars": 3.0,
  "price": 2
  ...}
...
{
  "business_id": "gclB3ED6uk6viWlolSb_uA",
  "user_id": "xVocUszkZtAqCxgWak3xVQ",
  "stars": 1,
  "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years...",
  "date": "2012-01-19"
  ...
}
{
  "business_id": "WXKx2I2SEzBpeUGtDMCS8A",
  "user_id": "84dCHkhWG8IDtk30VvaY5A",
  "stars": 2,
  "text": "-Excuse me for being a snob but if I wanted a room temperature burrito I would take one home, stick it in the fridge for a day, throw it in the microwave for 45 seconds, then eat it. NOT go to a restaurant and pay like seven dollars for one...",
  "date": "2009-04-30"
  ...
}
...]
```

Example: Similar Restaurants
Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files

```
Example: Reading Files
```

Set Intersection

```
Set Intersection
```

Sets

Sets are built-in Python container type
- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Elements may be added or removed from a set

```
Sets
```

Linear-Time Intersection of Sorted Lists

```
Linear-Time Intersection of Sorted Lists
```

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

```
Sets
```

Set

```
Set
```

Example: Reading Files

```
Example: Reading Files
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files

```
Example: Reading Files
```

Set Intersection

```
Set Intersection
```

Sets

Sets are built-in Python container type
- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Elements may be added or removed from a set

```
Sets
```

Linear-Time Intersection of Sorted Lists

```
Linear-Time Intersection of Sorted Lists
```

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

```
Sets
```

Set

```
Set
```

Example: Reading Files

```
Example: Reading Files
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files

```
Example: Reading Files
```

Set Intersection

```
Set Intersection
```

Sets

Sets are built-in Python container type
- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Elements may be added or removed from a set

```
Sets
```

Linear-Time Intersection of Sorted Lists

```
Linear-Time Intersection of Sorted Lists
```

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

```
Sets
```

Set

```
Set
```

Example: Reading Files

```
Example: Reading Files
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files

```
Example: Reading Files
```

Set Intersection

```
Set Intersection
```

Sets

Sets are built-in Python container type
- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Elements may be added or removed from a set

```
Sets
```

Linear-Time Intersection of Sorted Lists

```
Linear-Time Intersection of Sorted Lists
```

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

```
Sets
```

Set

```
Set
```

Example: Reading Files

```
Example: Reading Files
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files

```
Example: Reading Files
```

Set Intersection

```
Set Intersection
```

Sets

Sets are built-in Python container type
- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Elements may be added or removed from a set

```
Sets
```

Linear-Time Intersection of Sorted Lists

```
Linear-Time Intersection of Sorted Lists
```

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.""
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

```
Sets
```

Set

```
Set
```

Example: Reading Files

```
Example: Reading Files
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher similarity values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the similarity function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison.""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda x: -similarity(self, x))[:k]
```

Example: Reading Files