String Representations

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

The repr String for an Object

The **repr** function returns a Python expression (a string) that evaluates to an equal object

```
repr(object) -> string

Return the canonical string representation of the object.
For most object types, eval(repr(object)) == object.
```

The result of calling **repr** on a value is what Python prints in an interactive session

```
>>> 12e12
12000000000000.0
>>> print(repr(12e12))
12000000000000.0
```

Some objects do not have a simple Python-readable string

```
>>> repr(min)
'<built-in function min>'
```

The str String for an Object

Human interpretable strings are useful as well:

```
>>> from fractions import Fraction
>>> half = Fraction(1, 2)
>>> repr(half)
'Fraction(1, 2)'
>>> str(half)
'1/2'
```

The result of calling **str** on the value of an expression is what Python prints using the **print** function:

```
>>> print(half)
1/2
```

(Demo)

Polymorphic Functions

Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data

str and **repr** are both polymorphic; they apply to any object

repr invokes a zero-argument method `__repr__` on its argument

```
>>> half.__repr__()
'Fraction(1, 2)'
```

str invokes a zero-argument method `__str__` on its argument

```
>>> half.__str__()
'1/2'
```

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! Only class attributes are found
- *Question*: 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)
- *Question*: How would we implement this behavior?

```
def repr(x):
    return x.__repr__(x)
```

```
def repr(x):
    return x.__repr__()
```

```
def repr(x):
    return type(x).__repr__(x)
```

```
def repr(x):
    return type(x).__repr__()
```

```
def repr(x):
    return super(x).__repr__()
```

(Demo)

Interfaces

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

The attribute look-up 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

Example:

Classes that implement `__repr__` and `__str__` methods that return Python-interpretable and human-readable strings implement an interface for producing string representations

(Demo)

Special Method Names

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)

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

Same behavior using methods

```
>>> zero, one, two = 0, 1, 2
>>> one.__add__(two)
3
>>> zero.__bool__(), one.__bool__()
(False, True)
```

Special Methods

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

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

>>> Ratio(1, 3).__add__(Ratio(1, 6))
Ratio(1, 2)

>>> Ratio(1, 6).__radd__(Ratio(1, 3))
Ratio(1, 2)
```

http://getpython3.com/diveintopython3/special-method-names.html

http://docs.python.org/py3k/reference/datamodel.html#special-method-names

(Demo)

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

```
>>> Ratio(1, 3) + 1
Ratio(4, 3)

>>> 1 + Ratio(1, 3)
Ratio(4, 3)

>>> from math import pi
>>> Ratio(1, 3) + pi
3.4749259869231266
```

(Demo)

Announcements

Modular Design

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

Hog

Hog Game Simulator
- Game rules
- Ordering of events
- State tracking to determine the winner

Game Commentary
- Event descriptions
- State tracking to generate commentary

Player Strategies
- Decision rules
- Strategy parameters (e.g., margins & number of dice)

Ants

Ants Game Simulator
- Order of actions
- Food tracking
- Game ending conditions

Actions
- Characteristics of different ants & bees

Tunnel Structure
- Entrances & exits
- Locations of insects

Example: Restaurant Search

Restaurant Search Data

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


```
{"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 resturant and pay like seven dollars for one...", "date": "2009-04-30", ...}

...
```


(Demo)

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**.

```
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 r: -similarity(self, r)_____)______[:k]_______
```

sorted(iterable, /, *, key=None, reverse=False)

Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.

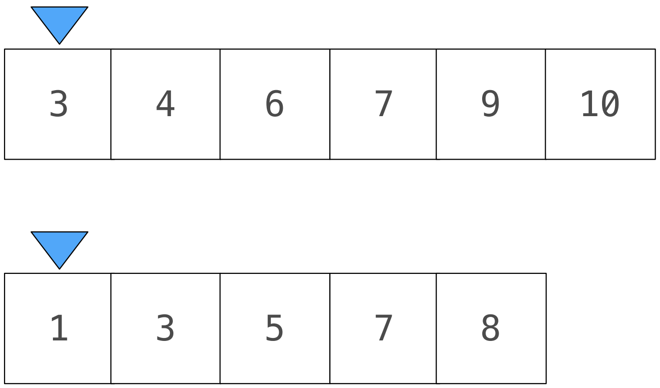
Example: Reading Files

(Demo)

Set Intersection

Linear-Time Intersection of Sorted Lists

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




```
def fast_overlap(s, t):
    """Return the overlap between sorted S and sorted T.

    >>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
    2
    """
    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
```

(Demo)

Sets

Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Sets have arbitrary order

```
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```