61A Lecture 12

Friday, February 20

---

**Announcements**

- Homework 4 due Monday 2/23 @ 11:59pm (small)
- Project 2 due Thursday 2/26 @ 11:59pm (BIG!)
- Project party Tuesday 2/24 5pm-6:30pm in 2050 VLSB
  - Bonus point for early submission by Wednesday 2/25 @ 11:59pm!

---

**Objects**

- Objects represent information.
- They consist of data and behavior, bundled together to create abstractions.
- Objects can represent things, but also properties, interactions, & processes.
- A type of object is called a class; classes are first-class values in Python.
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
  - In Python, every value is an object.
- All objects have attributes.
- A lot of data manipulation happens through object methods.
- Functions do one thing; objects do many related things.

---

**Representing Strings: the ASCII Standard**

- American Standard Code for Information Interchange
- 8 rows: 3 bits
- 16 columns: 4 bits
- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

---

**Representing Strings: the Unicode Standard**

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character


---

**Mutation Operations**

---

(U+0058 LATIN CAPITAL LETTER X)
(U+263a WHITE SMILING FACE)
(U+2639 WHITE FROWNING FACE)
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

All names that refer to the same object are affected by a mutation

Only objects of mutable types can change: lists & dictionaries

Identity Operators

Identity

is exp1

evaluates to True if both exp1 and exp2 evaluate to the same object

Equality

== exp1

evaluates to True if both exp1 and exp2 evaluate to equal values

Identical objects are always equal values

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
def mystery1(s):
    s.pop()
def mystery2(s):
    s[2] = []

>>> len(four)
4
>>> mystery1(four)
>>> len(four)
3
>>> mystery2(four)
>>> len(four)
2
```

The same object can change in value throughout the course of computation

First example in the course of an object changing state

```
evaluates to True

evaluates to True
```

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(x=1):
    ... s.append(x)
    ... return s

>>> f()
[1]
>>> f()
[1, 2]
```

Identity Operators

Identity

is exp1

evaluates to True if both exp1 and exp2 evaluate to the same object

Equality

== exp1

evaluates to True if both exp1 and exp2 evaluate to equal values

Identical objects are always equal values

Interactive Diagram