61A Lecture 12

Friday, February 20

- 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

(Demo)

- 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.

Example: Strings

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

"Bell" (\a)   ASCII Code Chart   "Line feed" (\n)

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

(Demo)

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

https://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

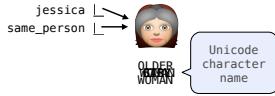U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE

(Demo)

Mutation Operations

Some Objects Can Change

[Demo]

First example in the course of an object changing state

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

jessica ⟶ 👩

same_person ⟶

OLDER WOMAN

Unicode character name

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

Only objects of *mutable* types can change: lists & dictionaries

{Demo}

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 mystery(s):    or   def mystery(s):
>>> len(four)                        s.pop()                  s[2:] = []
4                                    s.pop()
>>> mystery(four)
>>> len(four)
2


>>> four = [1, 2, 3, 4]          def another_mystery():
>>> len(four)                        four.pop()
4                                    four.pop()
>>> another_mystery() # No arguments!
>>> len(four)
2
```

Interactive Diagram

Tuples

(Demo)

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)           >>> turtle = [1, 2, 3]
>>> ooze()                       >>> ooze()
>>> turtle                       >>> turtle
(1, 2, 3)                        ['Anything could be inside!']
```

Next lecture: ooze can change turtle's binding

The value of an expression can change because of changes in names or objects

Name change:
```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:
```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)              >>> s = ([1, 2], 3)
>>> s[0] = 4                     >>> s[0][0] = 4
ERROR                            >>> s
                                 ([4, 2], 3)
```

Mutation

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]                     >>> a = [10]
>>> b = a                        >>> b = [10]
>>> a == b                       >>> a == b
True                             True
>>> a.append(20)                 >>> b.append(20)
>>> a == b                       >>> a
True                             [10]
>>> a                            >>> b
[10, 20]                         [10, 20]
>>> b                            >>> a == b
[10, 20]                         False
```

Identity Operators

Identity

<exp0> **is** <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

<exp0> == <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values

(Demo)

Mutable Default Arguments are Dangerous

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

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```

Global frame

f ⟶ func f(s) [parent=Global]

list
```
0   1   2
3   3   3
```

f1: f [parent=Global]

s

Return value 1

Each time the function is called, s is bound to the same value!

f2: f [parent=Global]

s

Return value 2

f3: f [parent=Global]

s

Return value 3

Interactive Diagram