Lecture 12: Mutable Sequences

Marvin Zhang
07/11/2016

Announcements

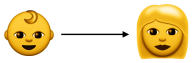
Roadmap

Introduction
Functions
Data
Mutability
Objects
Interpretation
Paradigms
Applications

- This short week (Mutability), the goals are:
  - To explore the power of values that can *mutate*, or change

Mutability (demo)

- Data abstraction allows us to think about compound values as units, or *objects*
- But many compound values have *state* that can change over time, i.e., they are *mutable*
- So far, we have treated all of our values as *immutable* — we can't change a value, we can only create a new one
  - This is not a good analogy for objects in the real world, e.g., people



  - This can also make code less elegant and less efficient
- To solve these problems, we introduce mutability

Lists, Dictionaries, and Sets

(demo)

Dictionary and Set Details

- Dictionaries and sets are *unordered* collections
- Keys in dictionaries and elements in sets:
  - Can't be mutable values, such as lists and dictionaries
  - Must be unique, i.e., no duplicates
- If you want to associate multiple values with a key, store them all in a sequence value, e.g.:

```
parity = {'odds': [1, 3, 5], 'evens': [2, 4, 6]}
```

Mutation through Function Calls

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

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s):
    s.pop()
    s.pop()
```

or

```
def mystery(s):
    s[2:] = []
```

A function's scope also includes *parent frames*

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

```
def another_mystery():
    four.pop()
    four.pop()
```

Interactive Diagram

Tuples and Strings are Immutable

(demo)

Identity vs Equality

- Because mutable values can change, the notion of *equality* is not as strong anymore
  - Two immutable values are always equal or always unequal to each other
  - Two mutable values can be sometimes equal and sometimes unequal to each other
- Each value also has an *identity*, which cannot change
- A list still has the same identity even if we change its contents
  - Conversely, two lists, even if they contain the same elements, never have the same identity

Identity vs Equality (demo)

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**

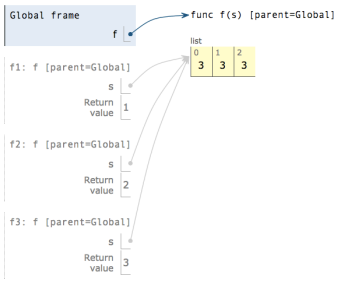
Identical objects are always equal values

Interactive Diagram

Mutable Default Arguments

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

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



Interactive Diagram

The Dictionary ADT, revisited

Now with the power of mutation! (demo)

Summary

- *Mutable values* such as lists and dictionaries have state and can be changed
  - This can be useful in writing programs that are more efficient and more understandable
- *Immutable values* cannot be changed after they are created
  - This is simpler and safer: immutable values that are equal (or unequal) will always be equal (or unequal)
- Knowing when and where to use both types of values is an important part of being a good programmer!