Lecture 12: Mutable Sequences

Announcements

Roadmap

Introduction
Functions
Data
Mutability
Objects
Interpretation
Paradigms
Applications

Mutability

- 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

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

```python
>>> four = [1, 2, 3, 4]  
def mystery(s): or def mystery(s):
    s.pop()  s[2] = []
>>> mystery(four)  s.pop()
>>> len(four)  2
```

A function’s scope also includes parent frames

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

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

- Identical objects are always equal values

Mutable Default Arguments

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

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

>>> f()  1
>>> f()  2
>>> f()  3
```

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!