## Discussion 06

#### Mutability, Iterators, Generators

Aditya Balasubramanian aditbala [at] berkeley [dot] edu

## Announcements

Homework 4 is due Thursday 10/6.



# Mutability

#### List Mutation Functions

- append(elem)
  - box elem in list and add to end of lst (can lead to nested lists)
  - only adds one element!
- extend(elem)
  - unbox elem and add to end (have to use an iterable)
- insert(index, elem)
  - insert elem at index (don't replace existing elem)
- remove(elem)
  - remove first appearance of elem in list (error if not found)
- pop(index)
  - o removes and return elem at index (default arg is end of list)

## **Mutating Lists**

- List Mutation Functions modify existing list
- Slicing creates a **new** list
- a = a + b creates a new list
- a += b mutates **existing** list (basically extend)
- Indexing into list and changing values modifies existing list

$$\circ$$
 a = [1, 2, 3]

$$\circ$$
 a[0] = 7

## Indentity vs Equality

- is
  - Check if two objects are the same (point to same reference in memory)
- ==
  - Check to see if content is the same
- Demo

```
>>> a = [7,6,4]
>>> b = [7,6,4]
>>> a is b
False
>>> a == b
True
```

## Shallow Copy and Deep Copy

- Shallow Copy
  - What Python does most of the time
  - Copy top level of list
  - Point to same objects with nested list
- Deep Copy
  - Make completely new copy of list
  - Difficult to do this
- Whenever we copy a sequence, we are using a shallow copy

## Worksheet

#### Iterators

- What are they?
  - An object that iterates over **iterables**, anything you can loop over
- Why use them?
  - Lazy evaluation
  - this

#### **Iterator Functions**

- next([iterator])
  - get next element in iterator
  - When there are no more elements,
     return StopIteration Exception
- iter([iterator])
  - o calling iter on an [iterator] will return the same iterator

## Iterator Example

```
>>> next(lst) # Calling next on an iterable
TypeError: 'list' object is not an iterator
>>> list_iter = iter(lst) # Creates an iterator for the list
>>> next(list_iter) # Calling next on an iterator
>>> next(iter(list_iter)) # Calling iter on an iterator returns itself
>>> for e in list_iter: # Exhausts remainder of list_iter
   print(e)
>>> next(list_iter) # No elements left!
StopIteration
                        # Original iterable is unaffected
>>> lst
[1, 2, 3]
```



#### Generators

- What are they?
  - functions that return a custom iterator
  - When called, they return generator objects
- How do they work?
  - When you call next on the generator object, you evaluate all the code until the yield statement
  - At the yield, you stop and return to that line when next is called again

#### **Generator Function**

```
def countdown(n):
    print("Beginning countdown!")
    while n \ge 0:
       yield n
        n -= 1
    print("Blastoff!")
>>> c1, c2 = countdown(2), countdown(2)
>>> c1 is iter(c1) # a generator is an iterator
True
>>> c1 is c2
False
>>> next(c1)
Beginning countdown!
>>> next(c2)
Beginning countdown!
               Slides by Aditya Balasubramanian
```

## yield from

• yield from iterable

```
for i in iterable: yield i
```

### Worksheet

## Thank you

Mid-Semester Survey -> http://go.cs61a.org/midsem-survey

Anon Feedback -> https://tinyurl.com/adit-anon