Nonlocal and Iterators

Abhinav Ashar
CS 61A: Structure and Interpretation of Computer Programs

January 18, 2019

1 Nonlocal

The keyword nonlocal is used to modify variables outside of the current frame. This is often useful when you have a variable before the creation of a function, and you want to modify that variable within a function.

a) You can only declare nonlocal when that variable is not being used in the current frame

b) The variable that is declared nonlocal should exist in a parent frame that is not the global frame

c) When doing environment diagrams, mark which variables are nonlocal. When a variable is referenced, first check whether it exists in the current frame. Then check the parent frame, which is not always the frame physically above the current frame in the environment diagram. Keep doing this until you find the variable.

d) The two ways to modify variables outside of a current frame: nonlocal and having them in a list

e) **With A Grain of Salt**: In the test, if you see a variable declared outside of a function, then the function declaration (Ex. def foo():), then an empty line for you to fill in, think about whether nonlocal works on that line.

2 Iterators and Generators

a) Iterable - anything that can be looped over (Ex. string, list etc.)

b) An iterator remembers the state of the function during its last iteration

c) An iterator is an iterable, but an iterable is not necessarily an iterator.

d) Example: Iterable is a book (pages can be flipped through). The iterator is the actual bookmark that keeps track of where you last were.

e) Generator - regular Python functions that contain the keyword yield

f) When a generator is called, it returns a generator object, not whatever the yielded value is

g) This generator object is an iterator

h) When only get the yield value from a generator object when we call next() on it

i) The keyword yield will return that value, pause the function, and save the local state

j) For loops and lists (Ex. list(foo()) where foo returns a generator object) implicitly call next()