Pass by Object Reference in Python

Tyler Moore

CS 2123, The University of Tulsa

Variables in Python

- Better thought of as names or identifiers attached to an object.
- A nice explanation:
  `http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#other-languages-have-variables`

Key distinction: mutable vs. immutable objects

- Immutable: objects whose value cannot change
  1. Tuples (makes sense)
  2. Booleans (surprise?)
  3. Numbers (surprise?)
  4. Strings (surprise?)
- Mutable: objects whose value can change
  1. Dictionaries
  2. Lists
  3. User-defined objects (unless defined as immutable)
- This distinction matters because it explains seemingly contradictory behavior

Variable assignment in action

```
>>> #variables are really names
... c = 4
>>> d = c
>>> c+=1
>>> c
5
>>> d     #d does not change because numbers are immutable
4
>>> #lists are mutable
... a = [1,4,2]
>>> b = a       #so this assigns the name b to the object attached to name a
>>> a.append(3)
>>> a
[1, 4, 2, 3]
>>> b  #b still points to the same object, its contents have just changed.
[1, 4, 2, 3]
```

Pass by object reference

- In Python, variables are not passed by reference or by value
- Instead, the name (aka object reference) is passed
- If the underlying object is mutable, then modifications to the object will persist
- If the underlying object is immutable, then changes to the variable do not persist
- For more info, see: `https://jeffknupp.com/blog/2012/11/13/is-python-callbyvalue-or-callbyreference-neither/`

Im/mutablility and function calls

```
>>> #let's try this in a function
... def increment(n): #n is a name assigned to the function argument when called
...     #because numbers are immutable, the following
...     #reassigns n to the number represented by n+1
...     n+=1
...     return n
...
>>> a = 3
>>> increment(a)
4
>>> #a does not change
... a
3
```

Im/mutablility and function calls

```
>>> def sortfun(s):
...     s.sort()
...     return s
...
>>> def sortfun2(s):
...     l = list(s)
...     l.sort()
...     return l
...
>>> a = [1,4,2]
>>> sortfun(a)
[1, 2, 4]
>>> a
[1, 2, 4]
>>> b = [3,9,1]
>>> sortfun2(b)
[1, 3, 9]
>>> b
[3, 9, 1]
```

Im/mutablility and function calls

```
def selection_sort(s):
    """
    Input: list s to be sorted
    Output: sorted list
    """
    for i in range(len(s)):
        #don't name min since reserved word
        minidx=i
        for j in range(i+1,len(s)):
            if s[j]<s[minidx]:
                minidx=j
        s[i],s[minidx]=s[minidx],s[i]
    return s

>>> b
[3, 9, 1]
>>> selection_sort(b)
[1, 3, 9]
>>> b
[1, 3, 9]
```