CS1110

Lecture 7: More on function calls

Sign up for one-on-one session on CMS if you didn't finish lab2 or lab3 during the lab session, even if you go checked off later. These 30-min. individual sessions do wonders at getting you up to speed.

CS 1110, LAB 3: TESTING
http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab03.pdf

Important References. This lab exercises concepts from lectures 3-5, and especially lecture 4. See the lectures page for the relevant slides and sample code: http://www.cs.cornell.edu/courses/cs1110/2014sp/lectures

For guidance on command-line navigation and using text-based commands, see http://www.cs.cornell.edu/courses/cs1110/2014sp/materials/command.php

Readings: *Today*: 3.9-3.10 (note that our notation differs slightly) *Next time*: 5.1-5.7; 10.0-10.2; 10.4-10.6
Watch http://www.cs.cornell.edu/courses/cs1110/2014sp/assignment/assignment1 **for updates.**

Slides by D. Gries, L. Lee, S. Marschner, W. White

Q: Why is it important to understand the notation for and mechanics of variables, objects, and frames?
A: You get a clear model of what names are accessible and what objects they refer to. Bonus: you'll understand error messages better.

So, to review: what is a *variable* (in Python)? A name for referring to a value/object. Two names can refer to the same thing; example: "that person talking in front of the room" and "the CS1110 prof with black hair".

What a name refers to can change (hence the name "variable"): "that person talking in front of the room" could refer to the person Prof. Lee at one time, and the person Prof. Marschner at another).

What is an *object?* An actual thing that can be referred to.

What is an *ID?* The unique identifier --- "one true name" --- for an object. Each object has a distinct id.

What is a *frame?* The function's "local view of the world": the names it defines and uses locally. These names disappear when the function call finishes.

How evaluate a function call expression, reformatted slightly:

Uno: Create a frame for the call

Dos: Assign arguments to parameters

(a) For each *parameter* ("the *names* in parentheses in the *function header*"), put a variable with that name in the frame

(b) Evaluate the *arguments* ("the *values* of the stuff in parentheses in the *function call*")

(c) Put the argument values in the corresponding parameter variables in the frame.

[The potentially hard/new concept embedded here: again, it's important to distinguish *names for things* from *the things that are named*.]

Tres: Execute function body, updating the frame's program counter (line number) as you go

Quatro: Cross out the frame

The value of the function call expression is the returned value (if there is one)

Ex: Can a Python function* change the speed of light?

That is, if lt_speed is a variable, can you write a function violate_physics(...) that changes the value of lt_speed?

code with function call

```
import lec07
lt_speed = 3e8
lec07.violate_physics(...)
```

lt_speed 3×10^8

... :...

function "definition" (in lec07.py)

```
def violate_physics(...?):
    """Changes lt_speed"""
1   ...?
```

*Given the Python we know at this point, where ***all assignments to a "plain variable" (not expressions with a "dot" in them) within a function are treated as referring to a local variable.***

---

code with function call

```
import lec07
lt_speed = 3e8
lec07.v_p_try1()
```

lt_speed 3×10^8

:

function definition

```
def v_p_try1():
    """Changes lt_speed"""
1   lt_speed = 42.0
```

---

code with function call

```
import lec07
lt_speed = 3e8
lec07.v_p_try2(42.0)
```

lt_speed 3×10^8

v_p_try2

function definition

```
def v_p_try2(new):
    """Changes lt_speed to new"""
1   lt_speed = new
```

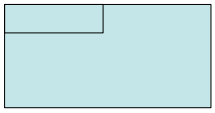
code with function call

```
import lec07
lt_speed = 3e8
lec07.v_p_try3(lt_speed)
```

function definition

```
def v_p_try3(lt_speed):
    """Changes lt_speed to 42.0"""
1   lt_speed = 42.0
```

lt_speed 3×10^8



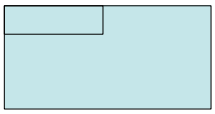
code with function call

```
import lec07
lt_speed = 3e8
lt_speed = lec07.boring(-3e8)
```

function definition

```
def boring(new):
    """Returns new"""
1   return new
```

lt_speed 3×10^8 -3×10^8



If functions are passed the *IDs* of objects as arguments,

...then they can "reach out" beyond the frame because they have a "handle" on the object: they can "summon" the object by its "true name".

With that in mind, now let's do the exercise.