The kid spoke. Very squeakily.
"I charge you ... to ... to..." Get on with it! "T-t-tell me your n-name."

That's usually how they start, the young ones. Meaningless waffle. \textit{He knew, and I knew that he knew, my name already; otherwise how could he have summoned me in the first place? You need the right words, the right actions, and most of all the right name.} I mean, it's not like hailing a cab – you don't get just any\underline{body}, when you call.

... "I am Bartimaeus! I am Sakhr al-Jinni, N'gorso the Mighty, and the Serpent of Silver Plumes! I have rebuilt the walls of Uruk, Karnak, and Prague. I have spoken with Solomon....I am Bartimaeus! I recognize no master!"

\textit{– The Amulet of Samarkand, Jonathan Stroud}
**CS1110**

Lecture 7: More on function calls

---

**Announcements**

Use the opportunity the one-on-one email offers!

Mac installers revised: see Piazza announcements. (This requires activating your Piazza account ...)

---

**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](http://www.cs.cornell.edu/courses/cs1110/2014sp/lectures)


---

Slides by D. Gries, L. Lee, S. Marschner, W. White
How evaluate a function call expression, reformatted slightly:

**Uno**: Create a frame for the call

**Dos**: Assign arguments to parameters

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

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

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

[The potentially hard/new concept embedded here: 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 \texttt{lt\_speed} is a variable, can you write a function \texttt{violate\_physics(...) that changes the value of lt\_speed?}

*Given the Python we know at this point, where \textit{all assignments to a "plain variable" (not expressions with a "dot" in them) within a function are treated as referring to a local variable.}
import lec07
lt_speed = 3e8
lec07.v_p_try1()

def v_p_try1():
    """Change lt_speed to 42.0"""
    lt_speed = 42.0
```python
import lec07
lt_speed = 3e8
lec07.v_p_try1()

def v_p_try1():
    """Change lt_speed to 42.0"""
    lt_speed = 42.0
```

```plaintext
no parameters
```
The frame is the "world" of the function; so it is, in fact, "boxed in" to its frame, like a genie in a bottle.

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

```python
def v_p_try1():
    """Change lt_speed to 42.0""
    lt_speed = 42.0
```

Python creates the variable assigned to if it's not already accessible.
import lec07
lt_speed = 3e8
lec07.v_p_try1()

def v_p_try1():
    """Change lt_speed to 42.0"""
    lt_speed = 42.0
import lec07
lt_speed = 3e8
lec07.v_p_try2(42.0)

def v_p_try2(new):
    """Change lt_speed to new"""
    lt_speed = new
import lec07
lt_speed = 3e8
lec07.v_p_try2(42.0)

def v_p_try2(new):
    """Change lt_speed to new""
    lt_speed = new
import lec07
lt_speed = 3e8
lec07.v_p_try2(42.0)

def v_p_try2(new):
    """Change lt_speed to new"""
    lt_speed = new

lt_speed 3 × 10^8
v_p_try2: 1
    new 42.0
import lec07
lt_speed = 3e8
lec07.v_p_try2(42.0)

def v_p_try2(new):
    """Change lt_speed to new"""
    lt_speed = new

lt_speed  \( \times 10^8 \)

v_p_try2: 2

new 42.0

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

def v_p_try2(new):
    """Change lt_speed to new"""
    lt_speed = new

lt_speed 3 × 10^8

v_p_try2(42.0)

new 42.0

lt_speed 42.0
import lec07
lt_speed = 3e8
lec07.v_p_try3(lt_speed)

def v_p_try3(lt_speed):
    """Change lt_speed to 42.0""
    lt_speed = 42.0
```python
import lec07
lt_speed = 3e8
lec07.v_p_try3(lt_speed)
```

```python
def v_p_try3(lt_speed):
    """Change lt_speed to 42.0"""
    lt_speed = 42.0
```

```python
lt_speed 3 \times 10^8
```

```python
v_p_try3: 1
```

```python
lt_speed 3 \times 10^8
```
import lec07
lt_speed = 3e8
lec07.v_p_try3(lt_speed)

def v_p_try3(lt_speed):
    """Change lt_speed to 42.0"""
    lt_speed = 42.0

note: only one lt_speed in the frame
import lec07
lt_speed = 3e8
lec07.v_p_try3(lt_speed)

def v_p_try3(lt_speed):
    """Change lt_speed to 42.0""
    lt_speed = 42.0

note: only one lt_speed in the frame
import lec07
lt_speed = 3e8
lt_speed = lec07.boring(-lt_speed)

def boring(new):
    """Returns: new""
    return new
```python
import lec07
lt_speed = 3e8
lt_speed = lec07.boring(-lt_speed)
```

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

```
lt_speed  3 \times 10^8
boring:1
  new  -3 \times 10^8
```
import lec07
lt_speed = 3e8
lt_speed = lec07.boring(-lt_speed)

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

boring:2
new -3 \times 10^8

lt_speed \quad 3 \times 10^8
import lec07
lt_speed = 3e8
lt_speed = lec07.boring(-lt_speed)

def boring(new):
    """Returns: new"""
    return new
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.*
How many things are wrong with this picture?

(A) 0-1  (B) 1-2  (C) 3-5  (D) more than 5

(E) You mean besides the fact that you think I can answer this?

[note: the ...distanceFrom... value will be 5.0, and "point" -> "tuple3d"
Executing yellow-code line "p=tuple3d.Point(0,3,4)"

<table>
<thead>
<tr>
<th>id1</th>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>x</td>
</tr>
<tr>
<td>3.0</td>
<td>y</td>
</tr>
<tr>
<td>4.0</td>
<td>z</td>
</tr>
</tbody>
</table>
Step uno of in evaluating the right-hand side of yellow-code line "p_new = lec07.rescale(p)"
Step dos of evaluating the RHS of "p_new = lec07.rescale(p)"
Step tres of evaluating the RHS of "p_new = lec07.rescale(p)": after executing line 1 of rescale
Step tres of evaluating the RHS of "p_new = lec07.rescale(p)"; after executing line 2 of rescale
Step tres of evaluating the RHS of "p_new = lec07.rescale(p)": after executing line 3 of rescale
Step tres of evaluating the RHS of "p_new = lec07.rescale(p)": after executing line 4 of rescale
Step quatro of evaluating the RHS of "p_new = lec07.rescale(p)"
lec07.rescale(p)
Note the lack of any returned value
After the completion of assignment statement `p_new = lec07.rescale(p)`

Note: in fact, the line "p_new = lec07.rescale()" isn't a good idea, given the rescale functions spec. Better would be just `lec07.rescale(p)` with no assignment to a variable.
Wrong things below: id1's coords should have changed. There should only be one object around, id1. There should be a variable named p_new with value None. Parameter value for pt should be id1. Local variables pt.x, pt.y, pt.z should not exist. norm should have value 5.0

(We purposely included mistakes students often make.)