CS1110

Lecture 7: More on function calls; if-then-else

Announcements

No office hours this week:
They've been replaced by the scheduled one-on-ones.

Readings:
Today: 3.9-3.10 (note that our notation differs slightly) and 5.1-5.7
Next time: 10.0-10.2 and 10.4-10.6

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's going on with Python, and, often, what's gone wrong with your code when you have a bug.

So, what is a *variable?*

A name for referring to a value/object ("that person talking in front of the room") .

What is an *object?*

The actual thing being referred to ("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 a *frame?*

The function's "local view of the world": the names and values it keeps around locally, and which 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*.]

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

Quatro: ~~Erase~~ Cross out the frame

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

Stack of frames: When functions call functions

function definitions

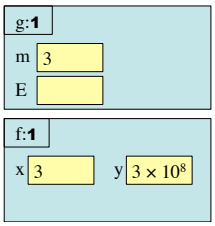
```
def g(m):
    """Returns: energy equivalent of mass m"""
1   E = f(m, lt_speed)
2   return E

def f(x, y):
    """Returns: x times square of y"""
1   return x * (y**2)
```

lt_speed 3×10^8

function call

```
lt_speed = 3e8
print g(3)
```

Two "live" frames



Conditionals: If-Statements

Format	Example

if <*boolean-expression*>:
  <*statement*>
  ...
  <*statement*>
	# Put x's value in z if it is positive
	if x > 0:
	  z = x

Execution:
if <*boolean-expression*> is true, then execute all of the statements indented directly underneath (until first non-indented statement)

Conditionals: If-Else-Statements

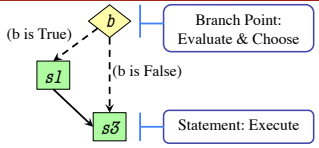
Format	Example

if <*boolean-expression*>:
  <*statement*>
  ...
else:
  <*statement*>
  ...
	# Put max of x, y in z
	if x > y:
	  z = x
	else:
	  z = y

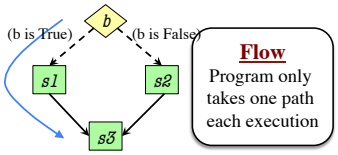
Execution:
if <*boolean-expression*> is true, then execute statements indented under if; otherwise execute the statements indented under else

Conditionals: "Control Flow" Statements

```
if b :
    s1 # statement
s3
```




```
if b :
    s1
else:
    s2
s3
```



Flow
Program only takes one path each execution

Conditional Expressions

Format	Example
`e1 if bexp else e2`	`# Put max of x, y in z` `z = x if x > y else y`

- `e1` and `e2` are any expression
- `bexp` is a boolean expression

expression, not statement

Conditionals: If-Elif-Statements

Format

```
if <boolean-expression>:
    <statement>
    ...
elif <boolean-expression>:
    <statement>
    ...
...
else:
    <statement>
    ...
```

Optional; executed if **all** are false

Example

```
# Put max of x, y, z in w
if x > y and x > z:
    w = x
elif y > z:
    w = y
else:
    w = z
```

Program Flow and Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return temp
```

- Value of `max(3,0)`?

A: 3
B: 0
C: **Error!**
D: I do not know

- What about if we said `max(0,3)` instead?