

Return values, multiple assignment

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 9, Spring 2021

Monday March 1

More on boolean operators

Compound boolean operations

Allow us to check if multiple conditions are true. Assume `x = 1`, `y = 2`, `z = False`

Operation	Result	Example
<code>not x</code>	True if <code>x</code> is false, False if <code>x</code> is true	<pre>if not z: # Do stuff</pre>
<code>x and y</code>	True only if <i>both</i> <code>x</code> and <code>y</code> is true, otherwise False	<pre>if x > 0 and y < 3: # Do stuff</pre>
<code>x or y</code>	True if <i>at least one of</i> <code>x</code> or <code>y</code> is true False if they are <i>both</i> false	<pre>if x > 5 or y < 3: # Do stuff</pre>

Precedence Rules: Higher in the table get evaluated earlier

Operation	Result
math operators	* / + - etc
relational operators	< > == != etc
not x	True if x is false, False if x is true
x and y	True only if <i>both</i> x and y is true, otherwise False
x or y	True if <i>at least one of</i> x or y is true False if they are <i>both</i> false

Lazy evaluation (1/2)

If there is more than one of the same compound boolean operator in a conditional statement, **Python evaluates left-to-right**.

In a long string of `and` conditions, Python stops checking and considers the entire expression `False` if it encounters one `False`:

```
if a and b and c and d and e and f and g:  
    print('yes')
```

If `a` is `False`, Python does not even bother checking `b`, `c`, etc.

Lazy evaluation (2/2)

In a long string of `or` conditions, Python stops checking and considers the entire expression `True` if it encounters one `True`:

```
if a or b or c or d or e or f or g:  
    print('yes')
```

If `a` is `True`, Python does not even bother checking `b`, `c`, etc.

Returning values from a function

Returning values from a function

```
import math
def distance(x, y):
    dist = math.sqrt(x**2 + y**2)

dist = 0
x = 3
y = 4
distance(x, y)
print(dist)
```

- What is printed at the end of the program?
- `dist` is not changed in the main symbol table and its value computed in the function "disappears"! How can we get the value to "live on"???

The return keyword

```
import math
def distance(x, y):
    dist = math.sqrt(x**2 + y**2)
    return dist
```

```
dist = 0
x = 3
y = 4
dist = distance(x, y)
print(dist)
```

- Let's make the changes and run the code.

None (1/3)

A **very** common error: attempting to assign a value from a function that DOES NOT return a value:

```
def distance(x, y):  
    dist = math.sqrt(x**2 + y**2)
```

```
# main code  
z = distance(x, y)
```

There is no return statement, so `distance` has nothing to assign to the variable `z` in the main symbol table.

- Python tries its best to avoid crashing and assigns `z` a special value called `None`.
- Python keeps executing the rest of the program.

None (2/3)

- You can assign variables of your own to None: `myVar = None`.
- You can checking if a variable is set to None:
 - `if z is None: ...`
 - or see it printed `print(z)`

None (3/3)

Same problem happens if you tried to assign a value to a function call that has an "empty" return statement, but doesn't return anything:

```
def distance(x, y):  
    return
```

```
# main code  
z = distance(x, y)
```

Example: Returning values to other functions

Let's write a program that prints the happy birthday song.

Symbol table and return values

Example: Calculating car's miles-per-gallon (MPG).