CS1 Lecture 6 Jan. 30, 2017

- HW2 available 1pm today,

  due Mon., Feb. 6, 9:00am
- Discussion sections tomorrow and Wednesday will practice things covered today and have small problem to submit.
- Survey number 2 (specific to this week's discussion section work) will be made available tomorrow. Best to complete them right away, but by Friday night at the latest.
- People who were paired with partners last week will keep the same partner for two more weeks.

Last time

Finished Ch 3

Variables and parameters are local

Stack diagrams

Fruitful functions

- incl return vs. print

Why Functions?

First part of Ch 5: Conditionals

Floor division and modulus (useful for HW1)

Logical/Boolean expressions

Conditional execution – if/elif/else

Today

- Finish conditional execution part of Chapter 5 (nested conditionals and if-elif-else)
- Chapter 7, Iteration (the last of what I said were five key programming components).

Note: the second “half” of Chapter 5 covers the topic of recursion. It is important and we will return to it a bit later.

Ch5: Condition execution – **if** (last time)

The most basic conditional statement is **if**

if (Boolean expression):

    …

    … lines of code that execute when Boolean

    … expression evaluates to True

…

… more lines of code. These execute whether or

… not Boolean expression evaluated to True

…

Ch5: **if-else** (last time)

if (Boolean expression):

    …
    … code that executes when Boolean expr
    … evaluates to True

else:

    …
    … code that executes when Boolean not true
    …

…

… code that executes after if-else statement, whether

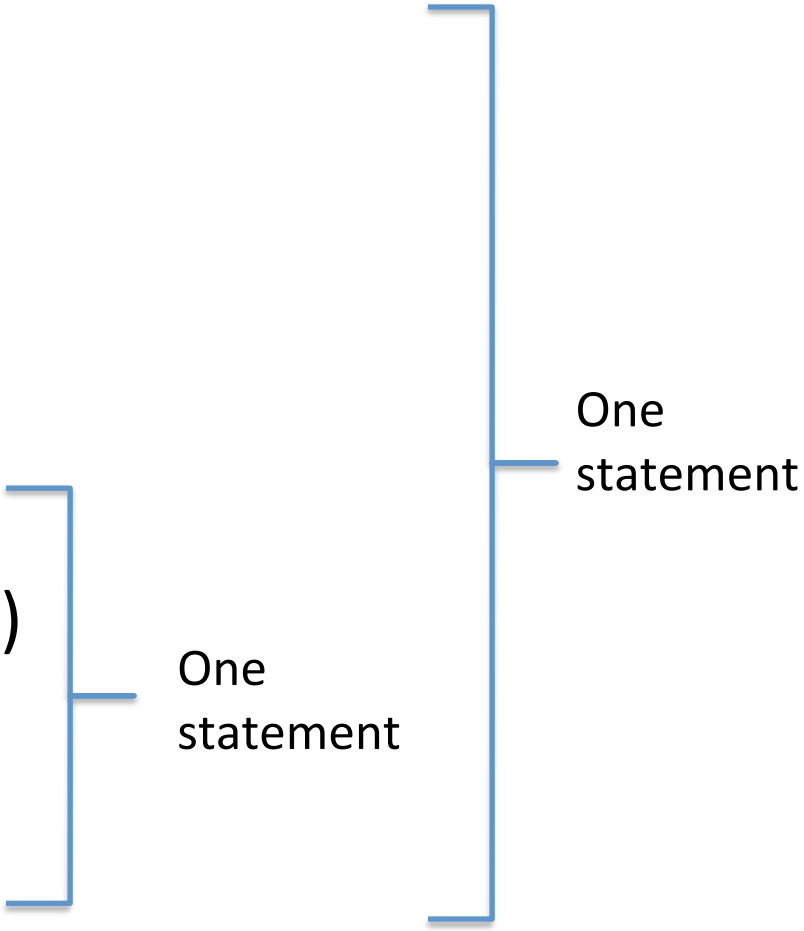
… Boolean expression was True or not

…

demo: testifs.py

Ch5: nested conditionals

```
if (a < b):
    print('a is smaller')
else:
    if (a == b):
        print('a and b are equal')
    else:
        print('a is larger')
```

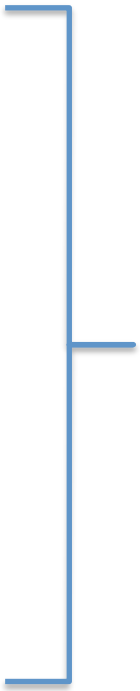


But there's an alternative in this particular situation …

Ch5: if-elif-else

(book calls this “chained conditionals)

```
if (a < b):
    print(‘a is smaller’)
elif (a == b):
    print(‘a and b are equal’)
else:
    print(‘a is larger’)
```



One statement

demo: lec6ifs.py

Ch 7: Iteration

- Recall again, five key components of programming, independent of particular programming language.
  - Expressions
  - Variables and assignment
  - Functions
  - Conditional execution (if-else)
  - Iteration

We've covered the first four. Chapter 7 introduces the last one – iteration/repetition.

Ch 7: Reassignment and Updating Variables

But before covering iteration, Chapter 7 has two small sections on assignment.

Reassignment: Sometimes programs reassign variables to new values. E.g.

```
>>> a = 3
>>> b = a + 2
>>> a = 2
```

As mentioned in earlier lectures, this should not cause confusion. Remember, assignment statements are not algebraic constraints. They have an immediate, perhaps temporary, effect. When evaluating an assignment statement, think in two steps:

1) [Ignore left hand side for the moment] Evaluate expression on right hand side of '=' using current values of variables.
2) associate variable name of left hand side with value obtained in step 1

Thus, in b = a + 2, associates b *with the value* 5, *not with the variable* a

And then a = 2 changes the associate of a from 3 to 2, and does not affect b's association (with 5) at all

Ch 7: Reassignment and Updating Variables

Updating: Often programs *reassign* a variable to a new value in terms of its own current value.

```
>>> x = 3
>>> y = 4
>>> x = x + y
```

If you think via the two-step process, this is not mysterious/confusing.

1) [Ignore left hand side for the moment] Evaluate expression on right hand side of '=' using current values of variables.
2) associate variable name of left hand side with value obtained in 1

Thus, after the first two lines, x has value 3, y has value 4.

To evaluate the third line,

1) evaluate x + y, yielding 7
2) associate x with 7

It's only mysterious/confusing when thought of as an algebraic equality/constraint.

Ch 7: Iteration – the **while** statement

- Many computations involve repetition, doing the same (or nearly the same) things repeatedly (perhaps a few times, perhaps billions of times)
- You can already write a program to, say, print out the first 1000 integers

```
def printFirstThousand():
    print(1)
    print(2)
    …
    print(1000)
```

- But Python (and other languages) provide statements to conveniently describe and control repetitive computations.

Ch 7 – the while statement

The **while** statement provides a very general mechanism for describing repetitive tasks.

```
…
… (B1: code before while)
…
while boolean expression:
    …
    …      (B2: code in while body)
    …
…
… (B3: code after while)
…
```

What happens?

1. Execute B1 code
2. Evaluate boolean expr
3. If True, do
   - 3a. eval B2 code.
   - 3b. jump to step 2 again

   If False, ignore B2 code and simply continue with step 4
4. Execute B3 code

Ch 7: the **while** statement

Using **while**, how can we write concise printFirstThousand()?

and

sumFirstThousand()

sumFirstN(number)

demo: lec6while.py

Next time

- Several more while examples
- Ch 8: Strings (and another iteration construct: **for)**

Read Ch 8!