§§3.4-3.10, 5.4: while and for loops

20 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

- *HW03 due today*
- *Quiz ch2 back*

Announcements

- Class cancelled tomorrow, Thursday 21Sep
- Python 2.5 has been released; we won't use it

Review of last time (§3.1-3.8)

- Selection: if, if..else.., if..elif..else
- Loops: while
- Sentinel variables
- Loop counters
- Using mathematical closed forms instead of loops
- abs(), += etc., string.capitalize()

What's on for today (§3.4-3.10, 5.4)

- String concatenation (+), repetition (*)
- Qualified import
- while loops: continue, break, else
- Common mistakes in loops
- for loops
- range()

String concatenation, repetition

- The plus operator (+) is overloaded to work with strings: concatenation
  - "Hello" + "World!" ---> "HelloWorld!"
  - Overloading is when one operator or function can do different things depending on the type of its arguments:
    - 2 + 3 --> integer addition
    - 2 + 3.0 --> float addition
    - "A" + "B" --> string concatenation
- Python also has string repetition:
  - "Hi!" * 3 --> "Hi!Hi!Hi!"

String concatenation vs. print

- print converts each of its arguments to a string, and puts spaces between them:
  - print “Hello”, “dear”, “World!”
    - ---> Hello dear World!
- String concatenation doesn't insert spaces:
  - print “Hello” + “dear” + “World!”
    - ---> HellodearWorld!

Qualified import

- The usual way to import a library:

```
import string
string.capitalize(“Hello!”)
```

- Import individual functions from a library:

```
from string import capitalize
capitalize(“Hello!”)
```

- Or import an entire library (don't do this):

```
from string import *
capitalize(“Hello!”)
```

- We'll learn later about namespaces

while loops: continue

- You can prematurely go to the next iteration of a while loop by using continue:
    - counter = 0
    - while counter < 5:
        - counter += 1
        - if counter == 3:
            - continue
        - print counter,
  - Output:
    - 1 2 4 5

while loops: break

- You can quit a while loop early by using break:
  - counter = 0
  - while counter < 5:
    - counter += 1
    - if counter == 3:
      - break
    - print counter,
- Output:
  - 1 2

while loops: else

- The optional else clause of a while loop is executed when the loop condition is False:
  - counter = 0
  - while counter < 5:
    - counter += 1
    - print counter,
  - else:
    - print “Loop is done!”
- Output:
  - 1 2 3 4 5 Loop is done!

while loops: break skips else

- If the loop is exited via break, the else clause is not performed:
  - counter = 0
  - while counter < 5:
    - counter += 1
    - if counter == 3:
      - break
    - print counter,
  - else:
    - print "Loop is done!"
- Output:
  - 1 2

Common errors with loops

- Print squares from 1^2 up to 10^2:
    - counter = 0
    - while counter < 10:
        - print counter*counter,
- What's wrong with this loop?

- Always make sure progress is being made in the loop!

Common errors with loops

- Count from 1 up to 10 by twos:
  - counter = 1
  - while counter != 10:
    - print counter,
    - counter += 2
- What's wrong with this loop?
  How would you fix it?
  - counter = 1
  - while counter < 10:
    - print counter,
    - counter += 2

Common errors with loops

- Count from 1.1 up to 2.0 in increments of 0.1:
  - counter = 1.1
  - while counter != 2.0:
    - print counter,
    - counter += 0.1
- Seems like it should work, but it might not due to inaccuracies in floating-point arithmetic
  - counter = 1.1
  - while counter < 2.0:
    - print counter,
    - counter += 0.1

for loops

- Since many while loops are counting loops, the for loop is an easy construct that prevents many of these errors
- Syntax:
  - **for *target* in *expression list* :**
    - ***Statement sequence***
- Example:
  - **for counter in (0, 1, 2, 3, 4):**
    - **print counter,**
  - Output:
    - 0 1 2 3 4
- for loops can also take an else sequence, like while loops

range()

- The built-in function range() produces a list suitable for use in a for loop:
  - range(10)   ----> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  - Note 0-based, and doesn't include end of range
  - Specify starting value:
    - range(1, 10)   ----> [1, 2, 3, 4, 5, 6, 7, 8, 9]
  - Specify increment:
    - range(10, 0, -2)   ----> [10, 8, 6, 4, 2]

- Technically, range() returns a list (mutable), rather than a tuple (immutable). We'll learn about lists and mutability later.

for loop examples

- Print squares from 1^2 up to 10^2:
  - **for counter in range(1, 10):**
    - **print counter * counter,**
- for loops can iterate over other lists:
  - **for appleVariety in ("Fuji", "Braeburn", "Gala"):**
    - **print "I like", appleVariety, "apples!"**

- Technically, the for loop uses an iterator to get the next item to loop over. Iterators are beyond the scope of CMPT140/145.

Review of today (§3.4-3.10, 5.4)

- String concatenation (+), repetition (*)
- Qualified import
- while loops: continue, break, else
- Common mistakes in loops
- for loops
- range()

TODO items

- Quiz: ch3 on Mon
- Lab2 next week: §3.14 # 36 and 45
- Reading: through §4.7 for Fri
- Class cancelled tomorrow