Iterating in Python: while loops & lists

L445/L515
Autumn 2010

The while loop

As we saw before...

A while loop does a condition check (like an if statement), and then runs a block of code as long as that condition evaluates to True.

```python
i = 0
while (i < 10):
    print i
    i += 1
```

Blocks of code

Python treats everything that is indented as a block of code.

• For while loops, this means that as long as the condition is true, the block of code is run
• In other words, the same block of code is run multiple times...
... usually with some of the variable values being different

It helps to trace your code to see this, i.e., observe what happens at every step

Trace output

Starting value of i: 1
1 1
Ending value of i: 2
Starting value of i: 2
2 4
Ending value of i: 3
...
Starting value of i: 10
10 100
Ending value of i: 11

Use #1: iteration

As we've just seen, while loops can be used to iterate over a sequence.

• This is most commonly done by iterating over integers, because integers easily count how many times you do something.
• You can change the way you iterate—e.g., i += 2 or i -= 1 or whatever
Use #2: until

Another, subtly different use is to perform the same actions until a certain condition is reached.

```python
user_input = ""
while len(user_input) < 10:
    user_input = raw_input("Please enter a long string: ")
print "Thank you for entering a long enough string!"
```

Output

```
Please enter a long string: a
Please enter a long string: ab
Please enter a long string: hello
Please enter a long string: supercalifragilistic
Thank you for entering a long enough string!
```

Logical operators

As mentioned last time, Python makes a difference between an assignment equals sign (=) and a logical equivalence sign (==)

- = assigns a value to a variable; == checks to see if two values are identical (used for if and while statements)

Review: we have these operators to compare values: ==, !=, <, >, <=, and >=

- We can also combine (and embed) conditions with and and or
- while (user_input != 'y') and (user_input != 'n'):

Lists

Python has a number of (compound) data types for combining other values, most notably lists

```python
a = ['hotel', 'motel', 100]
```

- Lists are concatenated and sliced in the same way that strings are

```python
>>> aa = ['word']
>>> bb = ['up']
>>> cc = aa + bb  # cc = ['word', 'up']
>>> cc[1]
'up'
```

- len gets the length of the list: len(a) equals 4
- sort and reverse do what you pretty much expect them to

Out of bounds errors

A common mistake is to try to index a part of the list that isn’t there

```python
>>> a = ['hotel', 'motel']
>>> a[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

A new way to iterate

A for loop allows you to iterate directly over the items in a list

```python
>>> a = ['hotel', 'motel', 'inn']
>>> for item in a:
...    print "Are you staying in a(n) " + item + "?"
... Are you staying in a(n) hotel?
Are you staying in a(n) motel?
Are you staying in a(n) inn?
>>> a
['hotel', 'motel', 'inn']
```
Another iteration method

List method: \texttt{pop()} removes the last item

- So, we can \texttt{pop} items from a list until there is no more list:

```python
>>> a = ['hotel', 'motel', 'inn']
>>> while a:
...     print "Are you staying in a(n) " + a.pop() + "?"
...
Are you staying in a(n) inn?
Are you staying in a(n) motel?
Are you staying in a(n) hotel?
>>> a
[]
```

NB: Only do this if you don't need the list contents when you're done

Another iteration method

The built-in function \texttt{range} takes an integer \( i \) and returns a list of integers from 0 to \( i-1 \)

- So, we can use this in combination with \texttt{len}

```python
>>> for i in range(len(a)):
...     print "Are you staying in a(n) " + a[i] + "?"
...
Are you staying in a(n) hotel?
Are you staying in a(n) motel?
Are you staying in a(n) inn?
>>> a
['hotel', 'motel', 'inn']
```

Allows you to access both the index (\( i \)) and the list value at that index (\( a[i] \))

List operations

- \texttt{append}—add list item to back of list
  
  ```python
  a.append('inn') # a = ['hotel','motel',100,'inn']
  ```

- \texttt{insert}—add item at a given position

  ```python
  a.insert(0,'hostel') # a = ['hostel','hotel','motel',100,'inn']
  ```

- \texttt{pop}—remove item at a given position

  ```python
  # a = ['hostel','hotel','motel',100,'inn']
a.pop() # a = ['hostel','hotel','motel',100]
a.pop(0) # a = ['hotel','motel',100]
  ```

- \texttt{index}—get index of element

  ```python
  a.index('motel') # 1
  ```