1 Motivation for `while`

Consider code to do this: type your password twice; must retry until get it to match!

Q. How would you write a **`for`** loop for this?

A.

There's another kind of loop for our situation: a **`while`** loop.

2 While loop basics

A **`while`** loop allows us to continue looping as long as some ***condition*** is **`True`**.

Form for a `while` loop:

```
while(condition):
    body
```

How it works:

1. Check to see if the condition (a Boolean expression) is **`True`**.
2. If it is, execute the ***entire*** body of the loop (even if the condition becomes **`False`** at some point).
3. Go back to the step 1.

[**`while_termination.py`**]

Q. In general, how many times does a **`while`** loop iterate?

A.

3 While loop examples

Example 1. [**yes_no.py**]

```
def yes_no_answer(message):
    '''(str) -> bool
    Prompt the user for a yes/no answer with message. Continue until
    they give a valid response. Return True if the answer was "yes", and
    return False otherwise.'''
    answer = raw_input(prompt)

    while answer != 'yes' and answer != 'no':
        answer = raw_input(prompt)

    return answer == 'yes'

if __name__ == "__main__":

    if yes_no_answer("Are you having fun? "):
        print "Great! Me too!"
    else:
        print "That's a shame."
```

Example 2. [**mystery.py**]

Trace this function a couple of examples, by hand, to predict what it returns. Run it to see if your predictions are correct. Write a good docstring for it and design some thorough test cases for it.

[**mystery.py**]

```
def mystery(s):
    '''
    (str) -> int
        Return the index of the first digit in s, or the length of s if there are no digits in s.


    '''

    i = 0
    while i < len(s) and s[i] not in "0123456789":
        i += 1
    return i
```

4 Lazy Evaluation

When we give python an **and** or an **or**, it always takes the lazy way:
don't bother evaluating the second operand if it makes no difference.