Repetition

(continued)

Review: ***while*** and ***for***

- The ***while*** loop repeats a set of statements while some condition is true.
  - Often called a ***sentinel*** controlled loop
  - while some *condition x* is true: execute block of statements inside { }
    - *condition x* is the sentinel

- The ***for*** statement is useful for iteration, moving through a sequence, one step at a time
  - Often called a counter-controlled loop
  - for a sequence of *n* steps: execute block of statements

Review: the *while* loop

```
x = 0
while x < 10:
    print("x = " + str(x))  //do something
    x += 1  # increment x; equivalent to this: x = x + 1
```

1. Test the condition. If the condition is true:
    a. Execute the statements inside the while block
    b. Repeat (i.e., go back to 1.)
2. Otherwise: exit the loop

What will this code print out?

Review: the *while* loop

```
x = 0
while x < 10:
    print("x = " + x)  # do something
    x += 1 # increment x; equivalent to this: x = x + 1
```

Implementing a while loop:

1. Initialize the sentinel ***outside the loop***
2. Inside the loop, ***change something***: either the value of the sentinel variable or something else that will eventually lead to the condition being false and exiting the loop

What can go wrong?

What's the problem in this code?

```
x = 0
while x < 10:
    print("x = " + str(x))
```

What about this code?

```
x = 0
while x < 10:
    print("x = " + str(x))
    x = x - 1
```

These are infinite loops: they go on forever (and will crash your browser)

A Useful Infinite Loop

- Our first attempt:

```
x = prompt("print more? type '0' for no")# initialize sentinel
while x > 0: # test sentinel
      print("more!")
      x = prompt("print more?") #change sentinel
```

- A very different approach:

```
while True:
      x = prompt("print more? type '0' for no")# initialize sentinel
      if x == 0:
            break //breaks out of the loop at this point
      print("more!")
```

- Use the break statement to end the loop prematurely

Review: the *for* loop

```
for item in collection:
    print("item = " + item)  //do something
    # we DO NOT increment a counter: the for loop does it for us internally
    # we DO NOT initialize it either: the for loop does it for us internally

for varName in iterable-Data-Structure:
    # next thing in iterable-Data-Structure put in varName
    suite of code
```

Range function

- Returns a "range" object
  (which is an iterable data structure)
- 3 versions
  - range(r, s) – means range of r to s-1
  - range(x) – means range(0, x), which is 0 to (x-1)
  - range(a, b, c) – like range(a, b), but c is the “step” value

Range function

- How many numbers are generated in range(0,8)?
- How many numbers are generated in range(1,8)?
- How many numbers are generated in range(x,y)?

Range function

- How many numbers are generated in range(0, 8)?
  8 - 0 = 8 numbers
- How many numbers are generated in range(1, 8)?
  8 - 1 = 7 numbers
- How many numbers are generated in range(x, y)?
  y - x numbers

When to use which?

- ***Use a for loop*** when we know in advance the number of iterations
  - Doing something n times (e.g., build a 8 x 8 table)
  - Iterate over a collection of HTML elements (e.g. , modify all links on a page)

- ***Use a while loop*** when we do not know in advance the number of iterations
  - Keep asking a user for valid input

Practice...