1 While loop

In addition to the **for** loop, there is something called a **while** loop. The **while** loop is very simple and flexible (but can be trickier to use!). Here is an example.

```
num = 2
while num < 50:
    print num,
    num = num * 2
```

The **while** condition, `num < 50`, is evaluated, and if it is `True` the statements in the loop body are executed. The loop condition is rechecked and if found to be `True`, the body executes again. This continues until the loop condition is checked and is `False`. In the example above, there are five *iterations*: the loop body executes five times. It prints `2 4 8 16 32`.

In general, the **while** loop has this form:

while boolean_expression:

statements (called the body)

It is similar in form to an **if** statement, but the behavior is very different. In an **if** statement, when the boolean expression is `True` the body is executed *once*; in a **while** statement, when the boolean expression is `True` the body is executed *repeatedly* until the expression becomes `False`.

2 While loops are more general

The **while** loop is more general, meaning that you can do more with a **while** loop than with a **for** loop. Any **for** loop can be translated into a **while** loop. For example, this **for** loop:

```
s = "abcxyz"
for ch in s:
    print ch,
```

can be translated into a while loop:

```
s = "abcxyz"
i = 0               # initialize before loop
while i < len(s):   # check for end
    ch = s[i]
    print ch,
    i += 1          # increment
```

On the other hand, there are while loops that cannot be easily translated to for loops. Consider the loop shown below: it is hard to predict how many times it loops. A mathematician, Lothar Collatz, conjectured this program terminates – that is, stops looping – for every positive n, but *no one knows for sure*!

```
n = int(raw_input("Enter a positive number: "))
while n != 1:
    print n,
    if n % 2 == 0:
        n = n / 2
    else:
        n = n * 3 + 1
```

3 While loops vs. for loops

When should you use a for loop and when should you use a while loop? Use a for loop if you can easily determine, before you start looping, the maximum number of times that you'll need to execute the body. Use a while loop when you don't know. Here are some general cases when a while loop is appropriate:

- You ask the user for a particular kind of input (say, a positive number) and keep asking until you receive an acceptable input.
- You have a random process (say flipping a coin) that you want to repeat until a certain event happens (say 10 heads in a row).
- You want to loop over a sequence (say a list or string) but stop when a certain item is found.
- You want to step through a sequence but take different size steps in each iteration. (For example, suppose you want to skip ahead three characters every time you see the letter "x.")

4 Exercises

Solutions are presented in class and also included in the moodle version of this handout.

1. Ask the user for a lowercase string and print each letter up to but not including the first vowel.

Solution:

```
s = raw_input("Gimme a string: ")
i = 0
while i < len(s) and s[i] not in 'aeiou':
    print s[i],
    i += 1
```

2. Translate this for loop into a while loop.

```
for i in range(5):
    print 2*(i + 1),
```

Solution:

```
i = 2
while i <= 10:
    print i,
    i += 2
```

3. Suppose n refers to some **int**. Print out each digit of n from least to most significant. Example: if n = 4982, the program should print 2 8 9 4. You cannot use the **str** function.

Solution:

```
n = int(raw_input("Enter a number: "))
while n > 0:
    last_digit = n % 10
    print last_digit,
    n = n/10
```

4. Translate this for loop into a while loop.

```
L = ['a', 'b', 'c', 'd', 'e']
for i in range(len(L)):
    print L[-(i+1)],
```

Solution:

```
i = len(L)-1
while i >= 0:
    print L[i],
    i -= 1
```

Adapted from materials by Gries and Campbell.