1 Compound statements

The **for** statement and the **if** statement are examples of **compound statements**. The first line of a compound statement ends in a colon and the remaining lines are indented and called the **body**.

The point is this: compound statements are just statements. We can put them anywhere. We have already seen an **if** statement inside the body of another **if** statement and we have seen an **if** statement inside the body of a **for** statement.

Today, we look at placing a **for** statement inside the body of another **for** statement. This is known as **nested loops**.

2 Nested loops

Here is an example of a **nested loop**.

```
scale = 5
for i in range(scale):       # outer loop
    print "->",
    for j in range(4):  # inner loop
        print j+1,
    print "<-"
```

The **outer loop** repeats fives times. For each iteration of the outer loop, *all* of the statements in its body are executed. The body includes the **inner loop**. The inner loop simply prints the numbers `1 2 3 4`, all on one line with spaces in between.

So, each iteration of the outer loop does three things: prints a right-pointing arrow (`'->'`), executes the inner loop which prints `1 2 3 4`, and finally prints a left-pointing arrow (`'<-'`). The end result looks like this:

```
-> 1 2 3 4 <-
-> 1 2 3 4 <-
-> 1 2 3 4 <-
-> 1 2 3 4 <-
-> 1 2 3 4 <-
```

Notice that the body of the inner loop, the statement **print** `j+1`, is executed $5 \times 4 = 20$ times!

1. How would we modify the nested loops so that the range of numbers printed increases with each line? Example of pattern at scale 5:

   ```
   -> 1 <-
   -> 1 2 <-
   -> 1 2 3 <-
   -> 1 2 3 4 <-
   -> 1 2 3 4 5 <-
   ```

Solution:

```
scale = 5
for i in range(scale):       # outer loop
    print "->",
    for j in range(i+1):  # inner loop
        print j+1,
    print "<-"
```

2. How would we modify the nested loops so that the range of numbers printed decreases with each line? Example of pattern at scale 5:

```
-> 1 2 3 4 5 <-
-> 1 2 3 4 <-
-> 1 2 3 <-
-> 1 2 <-
-> 1 <-
```

Solution:

```
scale = 5
for i in range(scale):       # outer loop
    print "->",
    for j in range(scale-i):  # inner loop
        print j+1,
    print "<-"
```

3. How would we modify the nested loops so that the range of numbers counts down on each line? Example of pattern at scale 5:

```
-> 5 <-
-> 5 4 <-
-> 5 4 3 <-
-> 5 4 3 2 <-
-> 5 4 3 2 1 <-
```

Solution:

```
scale = 5
for i in range(scale):       # outer loop
    print "->",
    for j in range(i+1):  # inner loop
        print scale-j,
```

```
    print "<-"
```

4. How would we modify the nested loops so that the range of numbers counts down on each line? Example of pattern at scale 5:

```
-> 1 2 3 4 5 <-
-> 2 3 4 5 <-
-> 3 4 5 <-
-> 4 5 <-
-> 5 <-
```

Solution:

```
scale = 5
for i in range(scale):
        print '->',
        for j in range(scale-i):
                print (j+i+1),
        print '<-'
```

3 Clock analogy

It might help to make an analogy between nested loops and the hands on a clock. The hour hand acts like an *outer loop*; the minute hand acts like an *inner loop*. The minute hand must loop entirely around the clock face before the hour hand can advance to the next hour. In fact, we can use nested loops to write a small program that cycles through the times of a digital clock.

5. Write a program that prints military time starting at `00:00` and going to `23:59`. It should be nicely formatted so that leading zeros are added as appropriate so there are always two digits for the hour and two digits for the minute. Example...

```
00:00
00:01
00:02
...
00:59
01:00
01:01
...
```

Solution:

```
for hour in range(24):
    for minute in range(60):
        # make single digit numbers look better
        if hour < 10:
            hour = '0' + str(hour)
        if minute < 10:
            minute = '0' + str(minute)

        # print the time
        print str(hour) + ':' + str(minute)
```