## 3.6 Break-statements and Continue-statements

### Break-statements

Both **for**-loops and **while**-loops are set up to exit only at the beginning or end of the loop. Once we start executing the body, the entire body is executed. Sometimes that isn't what you need to do. Suppose, for example, that you have a loop that reads names, and follows each name with a greeting. Suppose further that we want the loop to end when we get the empty string for a name. Since this is an indefinite situation (we don't know in advance how many names we will see, we need a **while**-loop for this. It is tempting to write the following code:

```
done = False
while not done:
    name = input( "Enter a name: " )
    print( "Hi %s!" % name )
    if name == "":
        done = True
```

This works, to some extent, but when we enter the empty string to exit it still prints

```
"Hi !"
```

What we really want is to check the exit condition as soon as we read the name, and to leave the loop as soon as this condition becomes True. Since the loop only checks this condition at the top, we need to put the rest of the loop body inside an **if**-statement:

```
done = False
while not done:
    name = input( "Enter a name: " )
    if name == "":
        done = True
    else:
        print( "Hi %s!" % name )
```

For a short loop this is simple and clear. As our programs become more complicated, it will sometimes be useful to exit from a loop immediately without putting the rest of the loop in a conditional statement. The **break**-statement is designed for this. The statement **break** causes execution to immediately leave the innermost loop in which it occurs. For example, the loop above could be written:

```
while True:
    name = input( "Enter a name: " )
    if name == "":
        break
    print( "Hi %s!" % name )
```

The "while True" header of this loop indicates that the loop body will handle the termination of the loop with a **break**-statement. Note that there is no **else**-clause on the **if**-statement. This isn't needed, because if the condition is True we will exit from the loop and never get to the **print**-statement.

Break-statements work with **for**-loops just as they do with **while**-loops. For example, the following code will print a statement about sending money to John and Paul, but not to George or Ringo:

```
for name in ["John", "Paul", "George", "Ringo"]:
    if name == "George":
        break
    print( "Send money to %s" % name )
```

If your code has loops inside loops, **break**-statements only exit from the innermost loop inside which they occur. Consider the following code fragment:

```
while True:
    name = input( "Enter a name: " )
    if name == "":
        break
    firstName = ""
    for letter in name:
        if letter == " ":
            break
        firstName = firstName + letter
    print( firstName )
```

Here there are two loops. The outer loops handles the input of names, and exits when it sees a blank name; the first **break**-statement handles this termination. The second loop, inside the first, builds up a string firstName by walking along the letters of name until it gets to the end or to a blank. When it gets to a blank, the **break**-statement exits it from the inner loop, causing firstName to be printed, but the outer loop is not ended, so the program will again ask for another name.

## Continue-statements

It is very common for a loop to generate values (perhaps by asking the user for them) and then do something with these values. For certain values there might not be anything to do. The **continue**-statement is designed for this situation.

When a **continue**-statement is executed, control goes back to the top of the loop. For example, the following code prints the odd numbers between 1 and 10:

```
for x in range(1, 10):
    if x % 2 == 0:
        continue
    print( x )
```

In practice **continue**-statements are used less frequently than **break**-statements but they are occasionally handy. Of course, we could always get the same effect with a conditional statement:

```
for x in range(1, 10):
    if x % 2 != 0:
        print( x )
```