Chapter 9

While loops

We have already learned about for loops, which allow us to repeat things a specified number of times. Sometimes, though, we need to repeat something, but we don't know ahead of time exactly how many times it has to be repeated. For instance, a game of Tic-tac-toe keeps going until someone wins or there are no more moves to be made, so the number of turns will vary from game to game. This is a situation that would call for a while loop.

9.1 Examples

Example 1 Let's go back to the first program we wrote back in Section 1.3, the temperature converter. One annoying thing about it is that the user has to restart the program for every new temperature. A while loop will allow the user to repeatedly enter temperatures. A simple way for the user to indicate that they are done is to have them enter a nonsense temperature like −1000 (which is below absolute 0). This is done below:

```
temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    print('In Fahrenheit that is', 9/5*temp+32)
```

Look at the `while` statement first. It says that we will keep looping, that is, keep getting and converting temperatures, as long as the temperature entered is not −1000. As soon as −1000 is entered, the while loop stops. Tracing through, the program first compares `temp` to −1000. If `temp` is not −1000, then the program asks for a temperature and converts it. The program then loops back up and again compares `temp` to −1000. If `temp` is not −1000, the program will ask for another temperature, convert it, and then loop back up again and do another comparison. It continues this process until the user enters −1000.

We need the line `temp=0` at the start, as without it, we would get a name error. The program would

get to the **while** statement, try to see if `temp` is not equal to −1000 and run into a problem because `temp` doesn't yet exist. To take care of this, we just declare `temp` equal to 0. There is nothing special about the value 0 here. We could set it to anything except −1000. (Setting it to −1000 would cause the condition on the while loop to be false right from the start and the loop would never run.)

Note that is natural to think of the while loop as continuing looping until the user enters -1000. However, when we construct the condition, instead of thinking about when to stop looping, we instead need to think in terms of what has to be true in order to keep going.

A while loop is a lot like an if statement. The difference is that the indented statements in an if block will only be executed once, whereas the indented statements in a while loop are repeatedly executed.

Example 2 One problem with the previous program is that when the user enters in −1000 to quit, the program still converts the value −1000 and doesn't give any message to indicate that the program has ended. A nicer way to do the program is shown below.

```
temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    if temp!=-1000:
        print('In Fahrenheit that is', 9/5*temp+32)
    else:
        print('Bye!')
```

Example 3 When first met if statements in Section 4.1, we wrote a program that played a simple random number guessing game. The problem with that program is that the player only gets one guess. We can, in a sense, replace the if statement in that program with a while loop to create a program that allows the user to keep guessing until they get it right.

```
from random import randint
secret_num = randint(1,10)
guess = 0
while guess != secret_num:
    guess = eval(input('Guess the secret number: '))
print('You finally got it!')
```

The condition `guess!=secret_num` says that as long as the current guess is not correct, we will keep looping. In this case, the loop consists of one statement, the input statement, and so the program will keep asking the user for a guess until their guess is correct. We require the line `guess=0` prior to the while loop so that the first time the program reaches the loop, there is something in `guess` for the program to use in the comparison. The exact value of `guess` doesn't really matter at this point. We just want something that is guaranteed to be different than `secret_num`. When the user finally guesses the right answer, the loop ends and program control moves to the **print** statement after the loop, which prints a congratulatory message to the player.

Example 4 We can use a while loop to mimic a for loop, as shown below. Both loops have the exact same effect.

```
for i in range(10):
    print(i)
```

```
i=0
while i<10:
    print(i)
    i=i+1
```

Remember that the for loop starts with the loop variable `i` equal to 0 and ends with it equal to 9. To use a while loop to mimic the for loop, we have to manually create our own loop variable `i`. We start by setting it to 0. In the while loop we have the same **`print`** statement as in the for loop, but we have another statement, `i=i+1`, to manually increase the loop variable, something that the for loop does automatically.

Example 5 Below is our old friend that converts from Fahrenheit to Celsius.

```
temp = eval(input('Enter a temperature in Celsius: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

A program that gets input from a user may want to check to see that the user has entered valid data. The smallest possible temperature is absolute zero, -273.15 °C. The program below takes absolute zero into account:

```
temp = eval(input('Enter a temperature in Celsius: '))
if temp<-273.15:
    print('That temperature is not possible.'))
else:
    print('In Fahrenheit, that is', 9/5*temp+32)
```

One way to improve this is to allow the user to keep reentering the temperature until they enter a valid one. You may have experienced something similar using an online form to enter a phone number or a credit card number. If you enter an invalid number, you are told to reenter it. In the code below, the while loop acts very similarly to the if statement in the previous example.

```
temp = eval(input('Enter a temperature in Celsius: '))
while temp<-273.15:
    temp = eval(input('Impossible.  Enter a valid temperature: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

Note that we do not need an **`else`** statement here, like we had with the if statement.. The condition on the while loop guarantees that we will only get to the **`print`** statement once the user enters a valid temperature. Until that point, the program will be stuck in the loop, continually asking the user for a new temperature.

Example 6 As mentioned before, it is a valuable skill is to be able to read code. One way to do so is to pretend to be the Python interpreter and go through the code line by line. Let's try it with the

code below.

```
i = 0
while i<50:
    print(i)
    i=i+2
print('Bye!')
```

The variable `i` gets set to 0 to start. Next, the program tests the condition on the while loop. Because `i` is 0, which is less than 50, the code indented under the **`while`** statement will get executed. This code prints the current value of `i` and then executes the statement `i=i+2` which adds 2 to `i`.

The variable `i` is now 2 and the program loops back to the **`while`** statement. It checks to see if `i` is less than 50, and since `i` is 2, which is less than 50, the indented code should be executed again. So we print `i` again, add 2 to it, and then loop back to check the while loop condition again. We keep doing this until finally `i` gets to 50.

At this point, the **`while`** condition will finally not be true anymore and the program jumps down to the first statement after the **`while`**, which prints `Bye!`. The end result of the program is the numbers 0, 2, 4, . . . , 48 followed by the message, `Bye!`.

9.2 Infinite loops

When working with while loops, sooner or later you will accidentally send Python into a never-ending loop. Here is an example:

```
i=0
while i<10:
    print(i)
```

In this program, the value of `i` never changes and so the condition `i<10` is always true. Python will continuously print zeroes. To stop a program caught in a never-ending loop, use `Restart Shell` under the `Shell` menu. You can use this to stop a Python program before it is finished executing.

Sometimes a never-ending loop is what you want. A simple way to create one is shown below:

```
while True:
    # statements to be repeated go here
```

The value **`True`** is called a boolean value and is discussed further in Section 10.2.

9.3 The `break` statement

The **`break`** statement can be used to break out of a for or while loop before the loop is finished.

Example 1 Here is a program that allows the user to enter up to 10 numbers. The user can stop early by entering a negative number.

```
for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        break
```

This could also be accomplished with a while loop.

```
i=0
num=1
while i<10 and num>0:
    num = eval(input('Enter a number: '))
```

Either method is ok. In many cases the `break` statement can help make your code easier to understand and less clumsy.

Example 2 Earlier in the chapter, we used a while loop to allow the user to repeatedly enter temperatures to be converted. Here is, more or less, the original version on the left compared with a different approach using the `break` statement.

```
temp = 0
while temp!=-1000:
    temp = eval(input(': '))
    if temp!=-1000:
        print(9/5*temp+32)
    else:
        print('Bye!')
```

```
while True:
    temp = eval(input(': '))
    if temp==-1000:
        print('Bye')
        break
    print(9/5*temp+32)
```

9.4 The else statement

There is an optional `else` that you can use with `break` statements. The code indented under the `else` gets executed only if the loop completes without a `break` happening.

Example 1 This is a simple example based off of Example 1 of the previous section.

```
for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        print('Stopped early')
        break
else:
    print('User entered all ten values')
```

The program allows the user to enter up to 10 numbers. If they enter a negative, then the program prints `Stopped early` and asks for no more numbers. If the user enters no negatives, then the program prints `User entered all ten values`.

Example 2 Here are two ways to check if an integer `num` is prime. A prime number is a number whose only divisors are 1 and itself. The approach on the left uses a while loop, while the approach on the right uses a for/break loop:

```
i=2
while i<num and num%i!=0:
    i=i+1
if i==num:
    print('Prime')
else:
    print('Not prime')
```

```
for i in range(2, num):
    if num%i==0:
        print('Not prime')
        break
else:
    print('Prime')
```

The idea behind both approaches is to scan through all the integers between 2 and `num`-1, and if any of them is a divisor, then we know `num` is not prime. To see if a value `i` is a divisor of `num`, we just have to check to see if `num%i` is 0.

The idea of the while loop version is we continue looping as long as we haven't found a divisor. If we get all the way through the loop without finding a divisor, then `i` will equal `num`, and in that case the number must be prime.

The idea of the for/break version is we loop through all the potential divisors, and as soon as we find one, we know the number is not prime and we print `Not prime` and stop looping. If we get all the way through the loop without breaking, then we have not found a divisor. In that case the **`else`** block will execute and print that the number is prime.

9.5 The guessing game, more nicely done

It is worth going through step-by-step how to develop a program. We will modify the guessing game program from Section 9.1 to do the following:

- The player only gets five turns.
- The program tells the player after each guess if the number is higher or lower.
- The program prints appropriate messages for when the player wins and loses.

Below is what we want the program to look like:

```
Enter your guess (1-100): 50
LOWER. 4 guesses left.

Enter your guess (1-100): 25
```

```
LOWER. 3 guesses left.

Enter your guess (1-100): 12
LOWER. 2 guesses left.

Enter your guess (1-100): 6
HIGHER. 1 guesses left.

Enter your guess (1-100): 9
LOWER. 0 guesses left.

You lose.  The correct number is 8
```

First, think about what we will need in the program:

- We need random numbers, so there will be an import statement at the beginning of the program and a `randint` function somewhere else.
- To allow the user to guess until they either guess right or run out of turns, one solution is to use while loop with a condition that takes care of both of these possibilities.
- There will be an input statement to get the user's guess. As this is something that is repeatedly done, it will go inside the loop.
- There will be an if statement to take care of the higher/lower thing. As this comparison will be done repeatedly and will depend on the user's guesses, it will go in the loop after the input statement.
- There will be a counting variable to keep track of how many turns the player has taken. Each time the user makes a guess, the count will go up by one, so this statement will also go inside the loop.

Next start coding those things that are easy to do:

```
from random import randint

secret_num = randint(1,100)
num_guesses = 0

while #some condition goes here#
    guess = eval(input('Enter your guess (1-100): '))
    num_guesses = num_guesses + 1
    # higher/lower if statement goes here
```

For the while loop, we want to continue looping as long as the user has not guessed the secret number and as long as the player has not used up all of their guesses:

```
while guess != secret_num and num_guesses <= 4:
```

The higher/lower if statement can be done like this:

```
    if guess < secret_num:
        print('HIGHER.', 5-num_guesses, 'guesses left.\n')
    elif guess > secret_num:
        print('LOWER.', 5-num_guesses, 'guesses left.\n')
    else:
        print('You got it!')
```

Finally, it would be nice to have a message for the player if they run out of turns. When they run out of turns, the while loop will stop looping and program control will shift to whatever comes outside of the loop. At this point we can print the message, but we only want to do so if the reason that the loop stopped is because of the player running out of turns and not because they guessed correctly. We can accomplish this with an if statement after the loop. This is shown below along with the rest of the completed program.

```
from random import randint

secret_num = randint(1,100)
num_guesses = 0
guess = 0

while guess != secret_num and num_guesses <= 4:
    guess = eval(input('Enter your guess (1-100): '))
    num_guesses = num_guesses + 1
    if guess < secret_num:
        print('HIGHER.', 5-num_guesses, 'guesses left.\n')
    elif guess > secret_num:
        print('LOWER.', 5-num_guesses, 'guesses left.\n')
    else:
        print('You got it!')

if num_guesses==5 and guess != secret_num:
    print('You lose.  The correct number is', secret_num)
```

Here is an alternative solution using a for/break loop:

```
from random import randint

secret_num = randint(1,100)

for num_guesses in range(5):
    guess = eval(input('Enter your guess (1-100): '))
    if guess < secret_num:
        print('HIGHER.', 5-num_guesses, 'guesses left.\n')
    elif guess > secret_num:
        print('LOWER.', 5-num_guesses, 'guesses left.\n')
    else:
        print('You got it!')
        break
else:
    print('You lose.  The correct number is', secret_num)
```