**4.2 Conditional Loops**

**4.2.1 While and while-else**

A ***loop*** is a statement that repeats other statement(s), which are called the action of the loop. A ***conditional loop*** repeats the action as long as an expression is **True**, or until something happens. In Python, the **while** statement is used as the conditional loop. The general form is:

while expression:

action

# rest of code

The **while** loop begins by evaluating the expression (which is also sometimes called the condition). If the expression is **True**, then the action is executed. So far, that is exactly like an **if** statement! But, after the action has been executed in its entirety, control goes back to the top of the loop, and the expression is evaluated again. If the expression is **True** this time, the action is executed again. Then, the expression is evaluated, and if it is **True**, the action is executed again. This continues as long as the expression is **True**. When the expression becomes **False**, the **while** loop ends and control goes to the rest of the code after the **while** loop. Just like selection statements and the **for** loop, the action can consist of any number of statements, which must be indented to the same level (typically 4 spaces).

For example, the **while** loop

| *x = 2*  *while x < 5:*  *x = x + 1*  *print(x)*  *print('!')* |
| --- |

would result in this output:

3

4

5

!

To begin with, the value of the variable *x* is 2, which is less than 5. So, the action is executed which increments *x* to get the value of 3, and this (3) is printed. 3 is less than 5, so the action is executed again, incrementing *x* to get the value of 4, and printing 4. 4 is still less than 5, so the action is executed again, incrementing *x* to get the value of 5, and printing 5. Then, when the expression is evaluated, 5 is not less than 5 so the expression is **False** and the **while** loop ends. The statement that prints '!' is after the loop, so it only executes once. Note that in this example the two statements in the action are indented 4 spaces.

Since the expression is always evaluated before the action in a **while** loop, it is possible that the action will not be executed at all. This could occur if the expression is **False** the first time that it is evaluated. The code

| *x = 20*  *while x < 5:*  *x = x + 1*  *print(x)*  *print('!')* |
| --- |

would result in this output:

!

Since 20 is not less than 5, the action of the loop is skipped and the code just prints '!'*.*

It is important that the action must change something in the expression so that eventually it becomes **False**. If this never happens, an ***infinite loop*** occurs. To exit from an infinite loop, hit Control-C in Windows or Linux, or Command-C on Macs for most Python environments. Others may be different; for example, in Jupyter notebooks it is necessary to choose Kernel and then Interrupt, or hit the square “interrupt the kernel” icon.

An optional **else** clause can be added to a **while** loop, which specifies an action to be executed when the condition becomes **False**. The general form is

while expression:

whileaction

else:

elseaction

# rest of code

For example, we could print the value of *x* that ends the action of the **while** loop:

| *x = 2*  *while x < 5:*  *x = x + 1*  *print(x)*  *else:*  *print('We are done at ', x) print('!')* |
| --- |

3

4

5

We are done at 5

!

The **else** clause only works if the condition becomes **False**. Any other method of ending the loop would not result in the execution of the **else** clause.

As we have seen, the **random** function returns a random real number in the range from 0 to 1, not including 1. As another example, we could use a **while** loop to print random real numbers as long as they are less than 0.5.

| *from random import random rf = random()*  *while rf < 0.5:*  *print(f'{rf:.2f}')*  *rf = random()*  *print('And that is it!')* |
| --- |

0.37

0.22

And that is it!

The **else** clause could be used to print the random number that ended the loop. **4.2.2 Error Checking**

When there is user input into a program, there is almost always a valid range of values. For example, if the user is prompted to enter the length of the sides of a square, the user should enter a positive number. There might be a tighter range than that; for example, it may be specified that the sides should be in the range from 5 to 7 meters. ***Error-checking*** means checking the user’s entry for errors. More specifically, error-checking generally involves continuing to prompt the user and read in the user’s entry until a valid value is entered. In Python, this can be accomplished using a **while** loop.

For example, the following code prompts the user for a positive number, and loops to continue prompting the user until the user does enter a positive number. For now, we will assume that the user enters a number, although we will see functions in Chapter 5 that will allow us to check to make sure that the user in fact entered a number.

| *number = input('Enter a positive number: ')*  *number = float(number)*  *while (number <= 0):*  *number = input('Seriously! Enter a positive number: ') number = float(number)*  *print('Thanks for entering', number)* |
| --- |

Running the code might result in the following.

Enter a positive number: -5

Seriously! Enter a positive number: -11.1

Seriously! Enter a positive number: 3

Thanks for entering 3.0

The user was prompted for a positive number, but entered -5. Since the **input** function returns a string, the user’s input is cast to the type **float**. Since that was a negative number (less than or equal to 0), the action of the loop was executed. In the action, the user was again prompted for a positive number (‘Seriously!’), but again the user entered a negative number. So, the action

was executed again but this time the user entered 3. Since 3 is not less than or equal to 0, the **False** condition causes the loop to cease executing. The code after the loop then printed the positive number that the user finally entered. Notice that prompting the user is repeated before the loop, and also in the action of the loop. This is so that there will be a new value of *number* every time the condition is evaluated.

Of course, it is possible that the user will follow instructions the first time and enter a positive number. In that case, running the code might result in the following.

Enter a positive number: 5.2

Thanks for entering 5.2

Since the condition is evaluated at the top of the loop, and it was already **False**, the action of the loop was skipped entirely.

**4.2.3 Counting in a while loop**

With conditional loops, it is not known ahead of time how many times the action of the loop will be executed. However, it is often useful to count how many times the action ended up being executed.

This can be accomplished by creating a counter variable, initializing it before the loop, and incrementing it by one in the action of the loop (so it is incremented every time the action is executed). For example, when error-checking we may want to know how many tries it took before the user entered a correct value.

| counter = 0;  number = input('Enter a positive number: ')  number = float(number)  counter = counter + 1  while (number <= 0):  number = input('Seriously! Enter a positive number: ') number = float(number)  counter = counter + 1  print('Thanks for entering', number)  print('It took you', counter, 'tries.') |
| --- |

Enter a positive number: -9

Seriously! Enter a positive number: 33 Thanks for entering 33.0

It took you 2 tries.