5
ASKING QUESTIONS WITH IF AND ELSE

In programming, we often ask yes or no questions, and decide to do something based on the answer. For example, we might ask, "Are you older than 20?" and if the answer is yes, respond with "You are too old!"

These sorts of questions are called *conditions*, and we combine these conditions and the responses into *if statements*. Conditions can be more complicated than a single question, and `if` statements can also be combined with multiple questions and different responses based on the answer to each question.

In this chapter, you'll learn how to use `if` statements to build programs.

IF STATEMENTS

An `if` statement might be written in Python like this:

```
>>> age = 13
>>> if age > 20:
        print('You are too old!')
```

An `if` statement is made up of the `if` keyword, followed by a condition and a colon (:), as in `if age > 20:`. The lines following the colon must be in a block, and if the answer to the question is yes (or *true*, as we say in Python programming), the commands in the block will be run. Now, let's explore how to write blocks and conditions.

A BLOCK IS A GROUP OF PROGRAMMING STATEMENTS

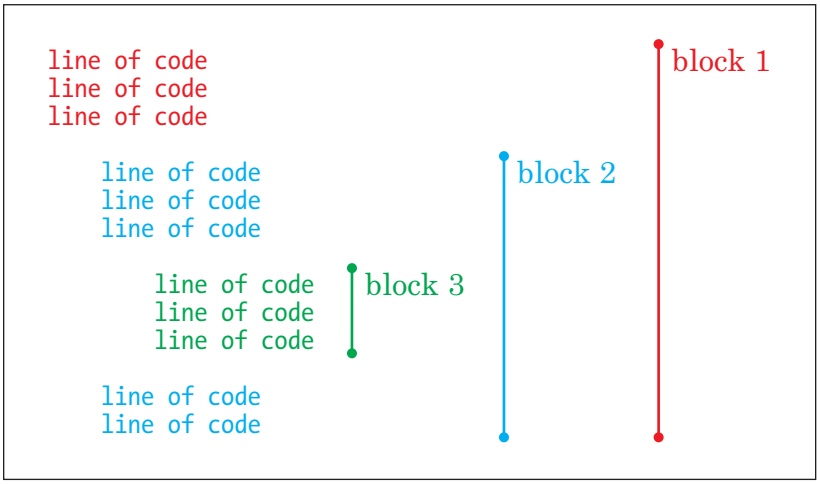
A *block* of code is a grouped set of programming statements. For example, when `if age > 20:` is true, you might want to do more than just print "You are too old!" Perhaps you want to print out a few other choice sentences, like this:

```
>>> age = 25
>>> if age > 20:
        print('You are too old!')
        print('Why are you here?')
        print('Why aren\'t you mowing a lawn or sorting papers?')
```

This block of code is made up of three `print` statements that are run only if the condition `age > 20` is found to be true. Each line in the block has four spaces at the beginning, when you compare it with the `if` statement above it. Let's look at that code again, with visible spaces:

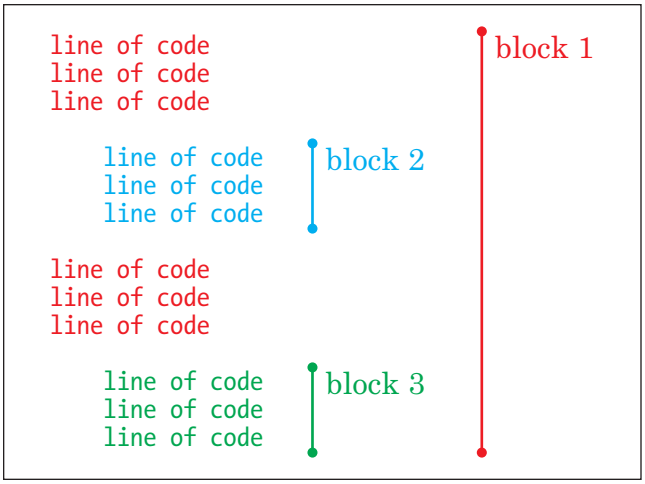
```
>>> age = 25
>>> if age > 20:
    ····print('You are too old!')
    ····print('Why are you here?')
    ····print('Why aren\'t you mowing a lawn or sorting papers?')
```

In Python, *whitespace*, such as a tab (inserted when you press the TAB key) or a space (inserted when you press the spacebar), is meaningful. Code that is at the same position (indented the same number of spaces from the left margin) is grouped into a block, and whenever you start a new line with more spaces than the previous one, you are starting a new block that is part of the previous one, like this:



We group statements together into blocks because they are related. The statements need to be run together.

When you change the indentation, you're generally creating new blocks. The following example shows three separate blocks that are created just by changing the indentation.



Here, even though blocks 2 and 3 have the same indentation, they are considered different blocks because there is a block with less indentation (fewer spaces) between them.

For that matter, a block with four spaces on one line and six spaces on the next will produce an *indentation error* when you run it, because Python expects you to use the same number of spaces for all the lines in a block. So if you start a block with four spaces, you should consistently use four spaces for that block. Here's an example:

```
>>> if age > 20:
    □□□□print('You are too old!')
    □□□□□□print('Why are you here?')
```

I've made the spaces visible so that you can see the differences. Notice that the third line has six spaces instead of four.

When we try to run this code, IDLE highlights the line where it sees a problem with a red block and displays an explanatory `SyntaxError` message:

```
>>> age = 25
>>> if age > 20:
        print('You are too old!')
        ■print('Why are you here?')
SyntaxError: unexpected indent
```

Python didn't expect to see two extra spaces at the beginning of the second `print` line.

NOTE *Use consistent spacing to make your code easier to read. If you start writing a program and put four spaces at the beginning of a block, keep using four spaces at the beginning of the other blocks in your program. Also, be sure to indent each line in the same block with the same number of spaces.*

CONDITIONS HELP US COMPARE THINGS

A *condition* is a programming statement that compares things and tells us whether the criteria set by the comparison are either `True` (yes) or `False` (no). For example, `age > 10` is a condition, and is another way of saying, "Is the value of the `age` variable greater than 10?" This is also a condition: `hair_color == 'mauve'`, which is another way of saying, "Is the value of the `hair_color` variable mauve?"

We use symbols in Python (called *operators*) to create our conditions, such as equal to, greater than, and less than. Table 5-1 lists some symbols for conditions.

Table 5-1: Symbols for Conditions

Symbol	Definition
`==`	Equal to
`!=`	Not equal to
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to

For example, if you are 10 years old, the condition `your_age == 10` would return `True`; otherwise, it would return `False`. If you are 12 years old, the condition `your_age > 10` would return `True`.

WARNING *Be sure to use a double equal sign* (`==`) *when defining an equal-to condition.*

Let's try a few more examples. Here, we set our age as equal to 10 and then write a conditional statement that will print "You are too old for my jokes!" if `age` is greater than 10:

```
>>> age = 10
>>> if age > 10:
        print('You are too old for my jokes!')
```

What happens when we type this into IDLE and press ENTER?

Nothing.

Because the value returned by `age` is not greater than 10, Python does not execute (run) the `print` block. However, if we had set the variable `age` to 20, the message would be printed.

Now let's change the previous example to use a greater-than-or-equal-to (`>=`) condition:

```
>>> age = 10
>>> if age >= 10:
        print('You are too old for my jokes!')
```

You should see "You are too old for my jokes!" printed to the screen because the value of `age` is equal to 10.

Next, let's try using an equal-to (`==`) condition:

```
>>> age = 10
>>> if age == 10:
        print('What\'s brown and sticky? A stick!!')
```

You should see the message "What's brown and sticky? A stick!!" printed to the screen.

IF-THEN-ELSE STATEMENTS

In addition to using `if` statements to do something when a condition is met (`True`), we can also use `if` statements to do something when a condition is not true. For example, we might print one message to the screen if your age is 12 and another if it's not 12 (`False`).

The trick here is to use an `if-then-else` statement, which essentially says "*If* something is true, then do this; or *else*, do that."

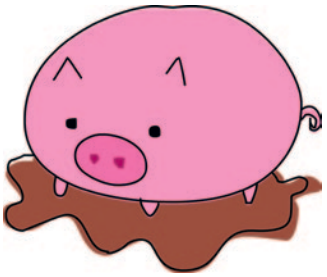
Let's create an `if-then-else` statement. Enter the following into the shell:

```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
```

```
>>> age = 12
>>> if age == 12:
        print("A pig fell in the mud!")
else:
        print("Shh. It's a secret.")

A pig fell in the mud!
```

Because we've set the `age` variable to 12, and the condition is asking whether `age` is equal to 12, you should see the first `print` message on the screen. Now try changing the value of `age` to a number other than 12, like this:



```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
>>> age = 8
>>> if age == 12:
        print("A pig fell in the mud!")
else:
        print("Shh. It's a secret.")

Shh. It's a secret.
```

This time, you should see the second `print` message.

IF AND ELIF STATEMENTS

We can extend an `if` statement even further with `elif` (which is short for `else-if`). For example, we can check if a person's age is 10, 11, or 12 (and so on) and have our program do something different based on the answer. These statements are different from `if-then-else` statements in that there can be more than one `elif` in the same statement:

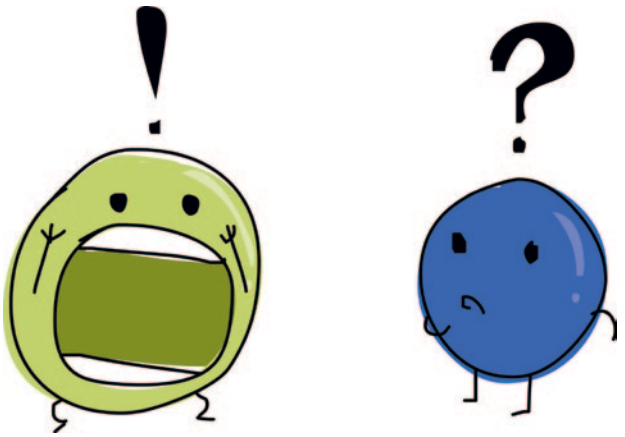
```
>>> age = 12
❶ >>> if age == 10:
❷         print("What do you call an unhappy cranberry?")
          print("A blueberry!")
```

```
❸ elif age == 11:
        print("What did the green grape say to the blue grape?")
        print("Breathe! Breathe!")
❹ elif age == 12:
❺       print("What did 0 say to 8?")
        print("Hi guys!")
  elif age == 13:
        print("Why wasn't 10 afraid of 7?")
        print("Because rather than eating 9, 7 8 pi.")
  else:
        print("Huh?")

What did 0 say to 8? Hi guys!
```

In this example, the `if` statement on the second line checks to see if the value of the `age` variable is equal to 10 at ❶. The `print` statement that follows at ❷ is run if `age` is equal to 10. However, since we've set `age` equal to 12, the computer jumps to the next `if` statement at ❸ and checks if the value of `age` is equal to 11. It isn't, so the computer jumps to the next `if` statement at ❹ to see if `age` is equal to 12. It is, so this time, the computer executes the `print` command at ❺.

When you enter this code in the IDLE, it will automatically indent, so be sure to press the BACKSPACE or DELETE key once you've typed each `print` statement, so that your `if`, `elif`, and `else` statements will start at the far-left margin. This is the same position the `if` statement would be in if the prompt (`>>>`) were absent.



COMBINING CONDITIONS

You can combine conditions by using the keywords `and` and `or`, which produces shorter and simpler code. Here's an example of using `or`:

```
>>> if age == 10 or age == 11 or age == 12 or age == 13:
        print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
        print('Huh?')
```

In this code, if any of the conditions on the first line are true (in other words, if `age` is 10, 11, 12, *or* 13), the block of code on the next line beginning with `print` will run.

If the conditions in the first line are not true (`else`), Python moves to the block in the last line, displaying `Huh?` on the screen.

To shrink this example even further, we could use the `and` keyword, along with the greater than or equal-to operator (`>=`) and less-than-or-equal-to operator (`<=`), as follows:

```
>>> if age >= 10 and age <= 13:
        print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
        print('Huh?')
```

Here, if `age` is greater than or equal to 10 *and* less than or equal to 13, as defined on the first line with `if age >= 10 and age <= 13:`, the block of code beginning with `print` on the following line will run. For example, if the value of `age` is 12, then `What is 13 + 49 + 84 + 155 + 97? A headache!` will be printed to the screen, because 12 is more than 10 and less than 13.

VARIABLES WITH NO VALUE—NONE

Just as we can assign numbers, strings, and lists to a variable, we can also assign nothing, or an empty value, to a variable. In Python, an empty value is referred to as `None`, and it is the absence of value. And it's important to note that the value `None` is different

from the value 0 because it is the absence of a value, rather than a number with a value of 0. The only value that a variable has when we give it the empty value `None` is nothing. Here's an example:

```
>>> myval = None
>>> print(myval)
None
```

Assigning a value of `None` to a variable is one way to reset it to its original, empty state. Setting a variable to `None` is also a way to define a variable without setting its value. You might do this when you know you're going to need a variable later in your program, but you want to define all your variables at the beginning. Programmers often define their variables at the beginning of a program because placing them there makes it easy to see the names of all the variables used by a chunk of code.

You can check for `None` in an `if` statement as well, as in the following example:

```
>>> myval = None
>>> if myval == None:
        print("The variable myval doesn't have a value")

The variable myval doesn't have a value
```

This is useful when you only want to calculate a value for a variable if it hasn't already been calculated.

THE DIFFERENCE BETWEEN STRINGS AND NUMBERS

User input is what a person enters on the keyboard—whether that's a character, a pressed arrow or ENTER key, or anything else. User input comes into Python as a string, which means that when you type the number 10 on your keyboard, Python saves the number 10 into a variable as a string, not a number.

What's the difference between the number 10 and the string `'10'`? Both look the same to us, with the only difference being that one is surrounded by quotes. But to a computer, the two are very different.

For example, suppose that we compare the value of the variable age to a number in an if statement, like this:

```
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
```

Then we set the variable age to the number 10:

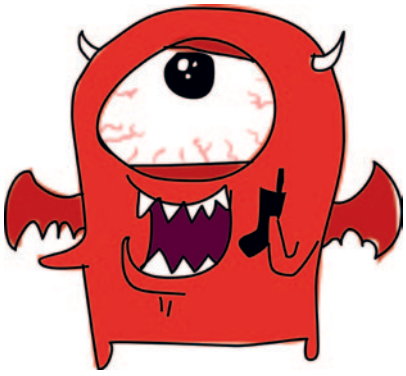
```
>>> age = 10
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
What's the best way to speak to a monster?
From as far away as possible!
```

As you can see, the print statement executes.

Next, we set age to the string '10' (with quotes), like this:

```
>>> age = '10'
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
```

Here, the code in the print statement doesn't run because Python doesn't see the number in quotes (a string) as a number.



Fortunately, Python has magic functions that can turn strings into numbers and numbers into strings. For example, you can convert the string '10' into a number with int:

```
>>> age = '10'
>>> converted_age = int(age)
```

The variable converted_age would now hold the number 10.

To convert a number into a string, use `str`:

```
>>> age = 10
>>> converted_age = str(age)
```

In this case, `converted_age` would hold the string `10` instead of the number 10.

Remember that `if age == 10` statement that didn't print anything when the variable was set to a string (`age = '10'`)? If we convert the variable first, we get an entirely different result:

```
>>> age = '10'
>>> converted_age = int(age)
>>> if converted_age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
What's the best way to speak to a monster?
From as far away as possible!
```

But hear this: If you try to convert a number with a decimal point, you'll get an error because the `int` function expects an integer.

```
>>> age = '10.5'
>>> converted_age = int(age)
Traceback (most recent call last):
    File "<pyshell#35>", line 1, in <module>
    converted_age = int(age)
ValueError: invalid literal for int() with base 10: '10.5'
```

A `ValueError` is what Python uses to tell you that the value you have tried to use isn't appropriate. To fix this, use the function `float` instead of `int`. The `float` function can handle numbers that aren't integers.

```
>>> age = '10.5'
>>> converted_age = float(age)
>>> print(converted_age)
10.5
```

You will also get a `ValueError` if you try to convert a string that doesn't contain a number in digits:

```
>>> age = 'ten'
>>> converted_age = int(age)
Traceback (most recent call last):
    File "<pyshell#1>", line 1, in <module>
```

```
    converted_age = int(age)
ValueError: invalid literal for int() with base 10: 'ten'
```

WHAT YOU LEARNED

In this chapter, you learned how to work with `if` statements to create blocks of code that are executed only when particular conditions are true. You saw how to extend `if` statements using `elif` so that different sections of code will execute as a result of different conditions, and how to use the `else` keyword to execute code if none of the conditions turn out to be true. You also learned how to combine conditions using the `and` and `or` keywords so that you can see if numbers fall in a range, and how to change strings into numbers with `int`, `str`, and `float`. And you discovered that nothing (`None`) has meaning in Python and can be used to reset variables to their initial, empty state.

PROGRAMMING PUZZLES

Try the following puzzles using `if` statement and conditions. The answers can be found at *http://python-for-kids.com/*.

#1: ARE YOU RICH?

What do you think the following code will do? Try to figure out the answer without typing it into the shell, and then check your answer.

```
>>> money = 2000
>>> if money > 1000:
        print("I'm rich!!")
else:
        print("I'm not rich!!")
          print("But I might be later...")
```

#2: TWINKIES!

Create an `if` statement that checks whether a number of Twinkies (in the variable `twinkies`) is less than 100 or greater than 500. Your program should print the message "Too few or too many" if the condition is true.

#3: JUST THE RIGHT NUMBER

Create an `if` statement that checks whether the amount of money contained in the variable `money` is between 100 and 500 *or* between 1,000 and 5,000.

#4: I CAN FIGHT THOSE NINJAS

Create an `if` statement that prints the string "That's too many" if the variable `ninjas` contains a number that's less than 50, prints "It'll be a struggle, but I can take 'em" if it's less than 30, and prints "I can fight those ninjas!" if it's less than 10. You might try out your code with:

```
>>> ninjas = 5
```