Lesson 13. Introducing decisions in programs

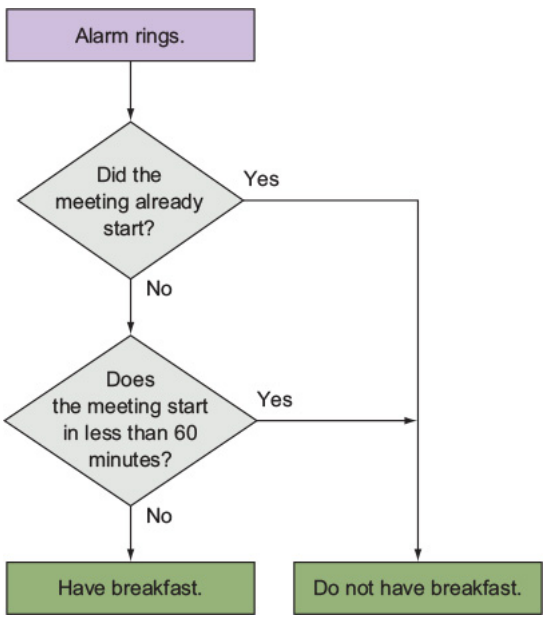
After reading lesson 13, you'll be able to

- Understand how the Python interpreter makes decisions
- Understand which lines of code get executed when a decision is made
- Write code that automatically decides which lines to execute depending on user input

When you write a program, you write lines of code. Each line of code is called a *statement*. You've been writing linear code, which means that when you run your program, every line of code is executed in the order that you wrote it; none of the lines are executed more than once, and none of the lines are skipped. This is equivalent to going through life without being allowed to make any decisions; this would be a constraining way to experience the world. You react to different stimuli in the world to make decisions, which leads to much more interesting experiences.

Consider this

It's Monday morning. Your first meeting is at 8:30 A.M., and your commute takes 45 minutes. Your alarm clock wakes you up promptly at 7:30 A.M. Use the following decision-maker to figure out whether you have time to eat breakfast.



A flowchart to decide whether you have time to eat breakfast after your alarm clock rings and you have a meeting that morning

Answer: You have time for breakfast!

13.1. MAKING DECISIONS WITH CONDITIONALS

You want programs to behave differently when given different stimuli. Stimuli come in the form of inputs to the program. The inputs can be given by a user interacting with the program or could be the result of an internal computation. Regardless, programs become more interesting, interactive, and useful when they're reactive.

13.1.1. Yes/no questions and true/false statements

phone during a break, and many others. A computer is great at doing what it's told, and you can program it to make these decisions for you.

When you make a decision, you ask a question. Questions such as "Is it sunny today?" can be answered with yes or no. All yes/no questions can be converted to statements that are either true or false. The Python interpreter doesn't understand yes/no, but it does understand true/false (Boolean logic). The question "Is it sunny today?" can be converted to the statement "It is sunny today." If you answered yes to the question, the statement is true. If you answered no, the statement is false. All decisions can be simplified to one (or more) yes/no questions, or equivalently, a series of true/false statements.

Quick check 13.1

Answer yes or no to the following questions:

1

Are you afraid of the dark?

2

Does your phone fit in your pocket?

3

Are you going to the movies tonight?

4

Does 5 times 5 equal 10?

5

Is the word *nibble* longer than the word *googol*?

Recall that every line of code in Python is a statement. Also recall that an expression is a specific kind of statement or part of a statement; an expression can be reduced to a value. The value is an object in Python; for example, an integer, float, or Boolean. Just as you make decisions in your day-to-day life, you can write programs that get the computer to make decisions. The true/false decision is a Python expression that evaluates to a Boolean, called a *Boolean expression*. Statements that contain a Boolean expression are called *conditional statements*.

Quick check 13.2

If possible, convert the following questions to Boolean expressions. Are there any that can't be converted?

1

Do you live in a treehouse?

2

What are you eating for dinner?

3

What color is your car?

4

Is the word youniverse in the dictionary?

5

Is the number 7 even?

6

Are variables `a` and `b` equal?

A computer works in terms of true and false, as opposed to yes and no. You should start to think about expressions that contain decisions as Boolean expressions, which evaluate to true or false.

13.1.2. Adding a condition to a statement

The same thought process can be applied to the code that you write. You can have special statements that contain an expression whose value is either true or false. We say that these statements are conditional statements and that they contain an expression that evaluates to the Python values of `True` or `False`. The part of the statement that evaluates to `True` or `False` is the conditional Boolean expression. This part drives the program to make a decision.

13.2. WRITING THE CODE TO MAKE THE DECISION

Python has a set of reserved keywords. They have a special meaning and therefore can't be used as variable names. One word, `if`, is reserved because it's used to write the simplest of all conditional statements, the `if` *conditional statement*.

13.2.1. Coding up a decision—an example

Listing 13.1 shows a simple conditional statement in code. You get an input number from the user. Then you check whether the user input is greater than 0. If that's true, you print an additional message. Lastly, you print a final message to the user that doesn't depend on the result of the conditional check.

Listing 13.1. Example of a simple conditional statement

```
num = int(input("Enter a number: "))              1
if num > 0:                                       2
    print("num is positive")                      3
print("finished comparing num to 0")              4
```

- ***1* Waits for user input and assigns the input from the user to the variable num**
- ***2* if statement checks whether the value stored in num is greater than 0**
- ***3* Go inside this block and execute all lines in this block if num is greater than 0.**
- ***4* Don't enter the indented code block if num isn't greater than 0, and execute this line directly.**

This is a simple way of writing an `if` statement. The code execution stops when it encounters a conditional statement, and performs the requested Boolean check. Depending on the result of the check, it'll either execute statements inside the condition's code block or not. Listing 13.1 can be rewritten as a flowchart, as shown in figure 13.1.

Figure 13.1. The flow of the code in listing 13.1. The result of asking the question in the diamond determines whether you execute another statement of code.

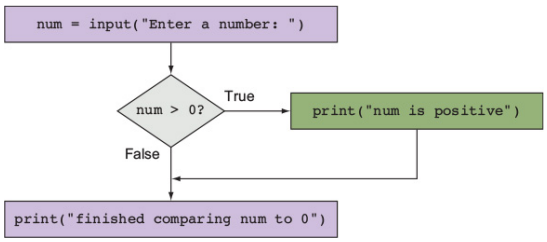


Figure 13.1 is a visual representation of listing 13.1. You can think of the conditional statement as a question you ask to decide whether to bypass executing statements inside its code block. In the visual representation, if the result of the conditional check is false, you take the "false" route and bypass the conditional's code block. If the result of the conditional check is true, you must enter the code block, visually represented by taking a small detour to execute the statements inside the conditional's code block.

Quick check 13.3

Take a look at this code snippet:

```
if num < 10:
    print("num is less than 10")
print("Finished")
```

2

What will the user see on the screen if `num` has a value of `10`?

3

What will the user see on the screen if `num` has a value of `100`?

13.2.2. Coding up a decision—a general way

Conditional statements have a certain look to them, and you must write them in this exact way so Python knows what you want to do (see the following listing). This is part of the *syntax* of the Python language.

Listing 13.2. A general way to write a simple `if` conditional

```
<some code before>              1
if <conditional>:               2
    <do something>              3
<some code after>               1
```

- ***1* <some code before> gets executed before checking the conditional, and <some code after> gets executed after the condition.**
- ***2* The keyword "if" starts the conditional line, followed by a conditional statement, followed by a conditional expression.**
- ***3* Indented to represent code that's executed only if the condition is True**

In listing 13.2, you see that the structure of the programs you're writing starts to change. Some lines of code are indented four spaces.

The conditional breaks up the flow of the program. Before, you were executing every line of code. Now, you're choosing whether to execute a line of code depending on whether a certain condition is met.

Quick check 13.4

Write simple conditional statements to do these tasks:

1

Ask the user for a word. Print the word that the user gave you. If the user gives you input that contains a space, also print that the user didn't follow the directions.

2

Ask the user for two numbers. Print their sum. If the sum is less than zero, also print "Wow, negative sum!"

13.3. STRUCTURING YOUR PROGRAMS

At this point, you can see that the structure of the programs you're writing is starting to change as you design them to make decisions:

- The conditional breaks up the flow of the program, which allows your program to make a decision.
- Some lines of code are indented, which tells Python how they relate to the statements above and below them.
- Before, you were executing every line of code. Now, you're choosing whether to execute a line of code depending on whether a certain condition is met.

13.3.1. Making many decisions

You can combine conditionals one after another to have a series of `if` statements. Every time you encounter the `if` statement, you decide whether to execute the code within that `if` statement's code block. In the following listing, you can see three conditional statements in a series. Each one checks for a different condition: a number is greater than 0, a number is less than 0, and a number is equal to 0.

Listing 13.3. Code with many conditional statements in a series

```
num_a = int(input("Pick a number: "))
if num_a > 0:
```

```
if num_a == 0:
    print("Your number is zero")
print("Finished!")
```

- ***1*** **Input from user**
- ***2*** **Check whether the number is greater than 0.**
- ***3*** **Do only if the preceding is true**
- ***4*** **Check whether the number is less than 0.**
- ***5*** **Do only if the preceding is true**
- ***6*** **Check whether the number equals 0.**
- ***7*** **Do only if the preceding is true**
- ***8*** **Executes no matter what**

Notice that you check for equality by using a double equal sign. This differentiates between equality (==) and variable assignment (=). Also, notice that the conditional `print` statements are all indented by the same amount.

Quick check 13.5

Q1:

Draw a flowchart for the code in listing 13.3 to make sure you understand that decisions are made sequentially. Flowcharts are a great way to organize all the possible paths through the code in a visual way. This is like figuring out all the possible ways to carry out a recipe.

13.3.2. Making decisions based on another decision's outcomes

Sometimes you want to consider a second decision based on the result of a previous decision. For example, you decide which cereal to buy only after you determine that you don't have any more cereal.

One way to do this in a Python program is using *nested conditionals*: a second conditional executes only if the result of the first conditional is `True`. Everything inside a conditional code block is a part of that code block—even a nested conditional. Further, the nested conditional will have its own code block.

Listing 13.4 compares two pieces of code; one code example nests a conditional inside the other, and the other code example leaves the conditionals in a series. In the nested code, the nested conditional statement (`if num_b < 0`) is executed only when the outer conditional (`if num_a < 0`) is `True`. Further, the code block inside the nested conditional (`print("num_b is negative")`) is executed only when both conditionals are `True`. In the unnested code, the nested conditional statement (`if num_b < 0`) is executed every time the program runs. The code block inside the nested conditional (`print("num_b is negative")`) is executed only when `num_b` is less than 0.

Listing 13.4. Combining conditionals by nesting or by putting them in series

```
Nested code                                  Unneste
num_a = int(input("Number? "))               num_a =
num_b = int(input("Number? "))               num_b =
if num_a < 0:                        1       if num_
    print("num_a: is negative")                  pri
    if num_b < 0:                    2       if num_
        print("num_b is negative")               pri
print("Finished")                    3        print(
```

- ***1*** **First conditional**
- ***2*** **Second conditional**
- ***3*** **Statement to be executed**

Quick check 13.6

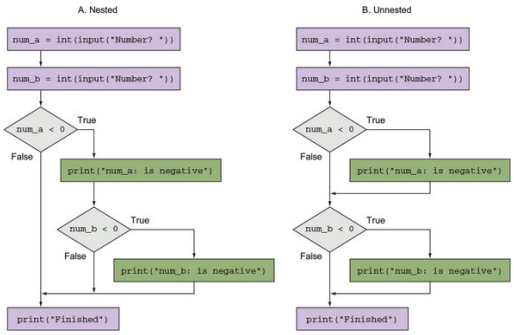
Q1:

What result will you get from the nested and unnested code in listing 13.4 if you input these values for `num_a` and `num_b`? Type up the code to check yourself!

```
num_a   num_b    Nested    Unnested
-------------------------------------------
-9      5
-------------------------------------------
 9      5
```

```
-----------------------------------------
9       -5
```

If you're not sure what happens or why you get a certain result from the code, try to trace through with the same values in the following flowchart.



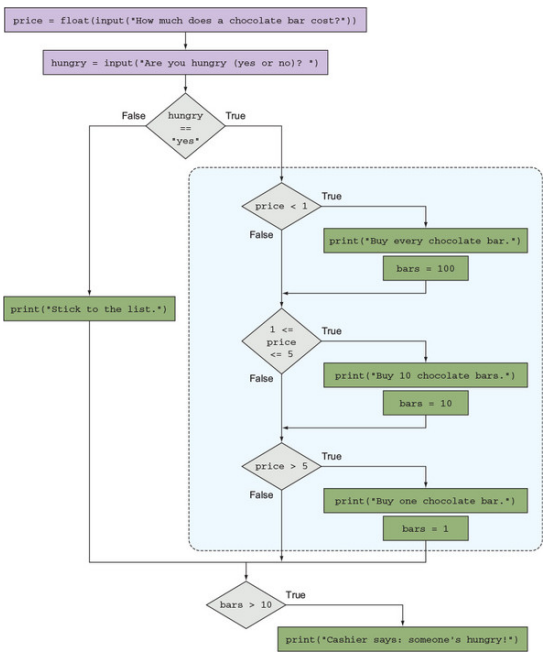
Difference between nested conditionals and conditionals in series. For the nested case, you make the `num_b < 0` decision only if the result of the `num_a < 0` decision was `True`. For the unnested case, you make the `num_b < 0` decision without taking into account the result of the `num_a < 0` decision.

13.3.3. A more complicated example with nested conditionals

As a last exercise, look at this more complicated task. You're going to the grocery store to buy groceries for the week. You notice chocolate bars as you enter the store. The program will help you decide the number of chocolate bars to buy. Start by looking at the flowchart in figure 13.2 for a program with these steps, to help you with this decision:

- It asks whether you're hungry.
- It asks how much a chocolate bar costs.
- If you're hungry and a chocolate bar costs less than a dollar, buy all of them.
- If you're hungry and a chocolate bar costs between 1 and 5 dollars, buy 10.
- If you're hungry and a chocolate bar costs more than 5 dollars, buy only 1.
- If you're not hungry, don't buy any.
- Then, depending on the number of bars you bought, the cashier will make a remark.

Figure 13.2. The dashed-line box represents the code block for when the `hungry == "yes"` conditional is true. Inside the dashed line is another conditional to determine the number of chocolate bars to buy, depending on the price.



You can see that the main flow of the program follows a vertical path from top to bottom. Every decision offers the possibility to deviate from the main path. The program contains three conditionals: one to determine whether

A code block for one conditional can contain other conditionals. The conditional to determine the number of chocolate bars to buy is nested inside the one that determines whether you're hungry.

Quick check 13.7

Q1:

Using the flowchart in figure 13.2 as a guide, try to write a Python program that performs this more complicated task. From the think-write-test-debug-repeat programming cycle, you're given the recipe, so you need to focus on the write-test-debug part. Specifically, the test step tells you how your program behaves. Your program should give the same output for the same input. When you're finished, compare your code to listing 13.5 and keep the following points in mind:

- Variable names can differ.
- Comments should be used to help you understand which parts are where.
- You can reorder some of the conditionals to get the same behavior; the same inputs should produce the same outputs.
- Most important, there's always more than one correct implementation.

Listing 13.5. Conditionals to decide how much chocolate to buy

```
price = float(input("How much does a chocolate bar co
hungry = input("Are you hungry (yes or no)? ")
bars = 0

if hungry == "yes":
    if price < 1:
        print("Buy every chocolate bar they have.")
        bars = 100
    if 1 <= price <= 5:
        print("Buy 10 chocolate bars.")
        bars = 10
    if price > 5:
        print("Buy only one chocolate bar.")
        bars = 1

if hungry == "no":
    print("Stick to the shopping list.")

if bars > 10:
    print("Cashier says: someone's hungry!")
```

- ***1*** **Input from user**
- ***2*** **Conditional check to decide if you're hungry**
- ***3*** **Conditional check to see whether the price of a bar is less than $1**
- ***4*** **Actions to do when the price of a bar is less than $1**
- ***5*** **Conditional check and actions to do when the price of a bar is between $1 and $5**
- ***6*** **Conditional check and actions to do when the price is greater than $5**
- ***7*** **Conditional check and actions to do when you say "no" when prompted if you're hungry**
- ***8*** **Prints a message only when the number of bars is greater than 10**

SUMMARY

In this lesson, my objective was to teach you how to implement decisions in code by using the `if` conditional statement. Conditional statements add a layer of complexity to your programs. They give programs the capability to deviate from the main program flow and to follow detours through other parts of the code. Here are the major takeaways:

- The `if` statement starts a conditional code block.
- A program can have more than one conditional, in a series or nested.
- A nested conditional is one within the code block of another conditional.
- You can visualize a program that includes conditional statements by using flowcharts.

As you're starting to write programs that involve a few concepts, it's impor-

type up your code, and run, test, and debug your program. Don't forget to comment out your code.

Let's see if you got this...

Q13.1

You're given the following two statements: "x is an odd number" and "x + 1 is an even number." Write a conditional statement and outcome using these two statements in the form: `if <condition> then <outcome>`.

Q13.2

Write a program that creates one variable, which can be an integer or a string. If the variable is an integer, print `I'm a numbers person`. If the variable is a string, print `I'm a words person`.

Q13.3

Write a program that reads in a string from the user. If the string contains at least one space, print `This string has spaces`.

Q13.4

Write a program that prints `Guess my number!` and assign a secret number to a variable. Read in an integer from the user. If the user's guess is lower than the secret number, print `Too low`. If the user's guess is higher than the secret number, print `Too high`. Finally, if the user's guess is the same as the secret number, print `You got it!`

Q13.5

Write a program that reads in an integer from the user and prints the absolute value of that number.

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Unit 3. Making decisions in your programs

NEXT
Lesson 14. Making more-complicated decisions