Lesson 35. Useful libraries

After reading lesson 35, you'll be able to

- Bring libraries from outside the standard Python package into your code
- Use the `math` library to do mathematical operations
- Use the `random` library to generate random numbers
- Use the `time` library to time programs

Programming is an activity that's usually most efficient and enjoyable when you build upon work that others have done. Some problems have already been solved, and code has likely been written to solve similar tasks to the ones that you're trying to solve. It's highly unlikely that you'll have to start a task by implementing code to do everything from scratch. In any language, libraries exist that you can use to help you code tasks in a modular way: by building upon code that's already written, tested, and debugged for correctness and efficiency.

To some extent, you've already been doing this! You've been using objects and operations built into the Python language. Imagine how tough learning to program would've been if you had to learn how to work with memory locations in the computer and to build up everything from scratch.

Consider this

Much of programming is building upon objects and ideas already there. Think about what you've learned so far. What are some examples of ways you've built upon things?

Answer:

You can use code that others have written (or even that you've written previously). You start with simple object types, and you make ones that are more complex. You build layers of abstractions by using functions and reuse functions with different inputs.

35.1. IMPORTING LIBRARIES

Arguably, you've learned two important things so far:

- How to create your own functions
- How to create your own object types, which package together a set of properties and behaviors for an object type

More complex code requires incorporating many functions and object types whose definitions you have to include. One way to do this is to copy and paste the definitions into your code. But there's another way, which is more common and less error-prone. When you have functions and classes defined in other files, you can use an `import` statement at the top of your code. The reason you might have different functions or classes defined in different files is to keep your code organized, keeping in line with the idea of abstraction.

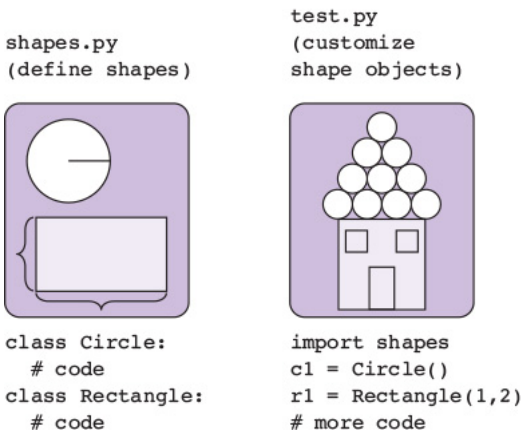
Suppose you have a file in which you define two classes you've already seen: `Circle` and `Rectangle`. In another file, you'd like to use these classes. You put one line in your file to bring in classes defined in another.

In a file named shapes.py, you define the `Circle` and `Rectangle` classes. In another file, you can bring in the classes by using the following:

```
import shapes
```

This process is called *importing* and tells Python to bring in all the classes and it omits the .py after the name of the file. For this line to work, both of these files must be in the same folder on your computer. Figure 35.1 shows the organization of code.

Figure 35.1. Two files: shapes.py and test.py in the same folder. One file defines classes for `Circle` and `Rectangle`. Another file imports the classes defined in shapes.py and uses them by creating different objects of those types and changing their data attributes.



```
shapes.py
(define shapes)

class Circle:
  # code
class Rectangle:
  # code
```

```
test.py
(customize
shape objects)

import shapes
c1 = Circle()
r1 = Rectangle(1,2)
# more code
```

Importing is a common practice that promotes code organization and decluttering. Typically, you'll want to import libraries into your code. Libraries are one or more modules, and a module is a file that contains definitions. Libraries often bundle modules of related uses together. Libraries can be built into the language (included with your installation of the language) or third-party (you download them from other online resources). In this unit, you'll use only built-in libraries.

Think of a Python library like a shop. Some shops are large, like department stores. These help you get all your items in one place, but they might not have some specialized merchandise. Other shops are small, like mall kiosks. These focus on one type of merchandise (for example, phones or perfume) and have a wider selection related to that one type of item.

When using a library, your first action is to go to its documentation to see the classes and functions defined in the library. For libraries that are built-in (a part of the language), this documentation can be found at the Python website: https://docs.python.org. This site links to the documentation for the latest version of Python, but you can view the documentation for any of the previous versions. This book uses Python version 3.5. If you don't have an internet connection to see the documentation online, you can also view the documentation through the Python console. You'll see how to do this in the next section.

Quick check 35.1

Q1:

Assume you have three files:

- fruits.py contains the definitions for classes of fruits.
- activities.py contains functions for activities that you'd do throughout your day.
- life.py contains the gameplay for a game of Life.

You want to use the classes and functions defined in fruits.py and activities.py in life.py. What lines do you need to write?

35.2. DOING MATHEMATICAL OPERATIONS WITH THE MATH LIBRARY

One of the most useful libraries is the math library. The math library for Python 3.5 is documented at https://docs.python.org/3.5/library/math.html. It deals with mathematical operations on numbers, where the operations aren't built into the language. To view the math library documentation without going online, you can type the following into the IPython console:

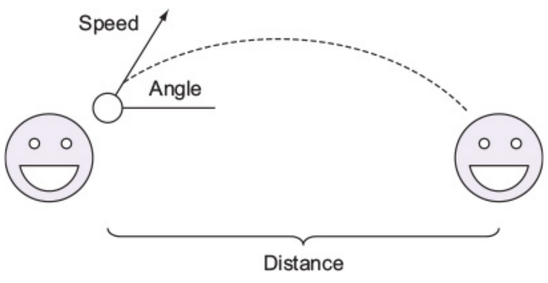
```
import math
help(math)
```

The console shows all the classes and functions defined in the math library, along with their docstrings. You can browse the docstrings and see whether any of them would be useful in your code.

The math library consists of functions organized by type: number-theoretic and representation functions, power and logarithmic functions, trigonomet-

Suppose you'd like to simulate throwing a ball at your friend in a field. You want to see whether the throw reaches your friend, allowing for a bit of leniency because your friend can jump up to catch, if need be. Let's see how to write a program that does this simulation for you. You'll ask the user for the distance your friend is standing away from you, the speed at which you can throw the ball, and an angle at which to throw the ball. The program will tell you whether the ball will make it far enough to be caught. Figure 35.2 shows the setup.

Figure 35.2. Setup for throwing a ball at a certain speed and angle so that it goes a certain distance



The formula that you can use to calculate how far a ball goes when it's thrown with a certain speed at a certain angle is as follows:

```
                        2
reach = 2 * speed  * sin(angle) * cos(angle) / 9.8
```

Listing 35.1 shows the code for this simple program. It first asks the user for the distance to the user's friend, the speed at which to throw the ball, and the angle at which to throw it. Then it uses the formula to calculate how far the ball will go. Allowing some tolerance (to account for the receiver being able to reach for it), it'll display one of three messages: it was caught, it fell short, or it went too long. To calculate the sin and cos values, you'll use functions in the math library, so you have to bring in the library using an `import` statement. Aside from implementing the formula, only one detail is left. An angle can be measured in degrees or radians. The functions in the math library assume that angles are given in radians, so you need to convert the angle from degrees to radians by using a function in the math library.

Listing 35.1. Using the math library to throw a ball at an angle

```
import math

distance = float(input("How far away is your friend?
speed = float(input("How fast can you throw? (m/s) ")
angle_d = float(input("What angle do you want to thro
tolerance = 2

angle_r = math.radians(angle_d)

reach = 2*speed**2*math.sin(angle_r)*math.cos(angle_r

if reach > distance - tolerance and reach < distance
    print("Nice throw!")
elif reach < distance - tolerance:
    print("You didn't throw far enough.")
else:
    print("You threw too far.")
```

- **1 Imports the math library functions**
- **2 Implements the formula, using the math library functions**
- **3 The library math.sin and math.cos take radians, not degrees, as the input, so convert the user input.**

Quick check 35.2

Q1:

Modify the program so that it asks the user only for how far away the friend is and how fast to throw the ball. Then, it loops through all the angles from 0 to 90 and prints whether the ball made it.

35.3. RANDOM NUMBERS WITH THE RANDOM LIBRARY

The random library provides numerous operations you can do to add unpredictability to programs. This library is documented at https://docs.python.org/3.5/library/random.html.

35.3.1. Randomizing lists

dictability comes from a pseudo-number generator, which can help you do things like pick a random number within a certain range, pick an item at random in a list or a dictionary, or rearrange a list for you at random, among others.

For example, say you have a list of people and want to pick one at random. Try typing this code in a file and running it:

```
import random
people = ["Ana","Bob","Carl","Doug","Elle","Finn"]
print(random.choice(people))
```

It prints one of the elements, at random, in the list of named people. If you run the program more than once, you'll notice that you'll get a different output each time it's run.

You can even pick a certain number of people from the list:

```
import random
people = ["Ana","Bob","Carl","Doug","Elle","Finn"]
print(random.sample(people, 3))
```

This code ensures that the same person isn't picked more than once, and prints a list of however many elements are specified (three, in this case).

35.3.2. Simulating games of chance

Another common use of the random library is to play games of chance. You can simulate probabilities of certain events happening by using the `random.random()` function: the first `random` is the library name, and the second `random` is the function name, which happens to be the same as the library name. This function returns a random floating-point number between 0 (inclusive) and 1 (not inclusive).

Listing 35.2 shows a program that plays rock-paper-scissors with the user. The program first asks the user to make their choice. Then, it gets a random number by using `random.random()`. To simulate the 1/3 probability of the computer picking one of rock, paper, or scissors, you can check that the random number generated falls within one of three ranges: 0 to 1/3, 1/3 to 2/3, and 2/3 to 1.

Listing 35.2. Using the random library to play rock-paper-scissors

```
import random

choice = input("Choose rock, paper, or scissors: ")
r = random.random()
if r < 1/3:
    print("Computer chose rock.")
    if choice == "paper":
        print("You win!")
    elif choice == "scissors":
        print("You lose.")
    else:
        print("Tie.")
elif 1/3 <= r < 2/3:
    print("Computer chose paper.")
    if choice == "scissors":
        print("You win!")
    elif choice == "rock":
        print("You lose.")
    else:
        print("Tie.")
else:
    print("Computer chose scissors.")
    if choice == "rock":
        print("You win!")
    elif choice == "paper":
        print("You lose.")
    else:
        print("Tie.")
```

- ***1*** **Brings in functions defined in the random library**
- ***2*** **Chooses a random float between 0 (inclusive) and 1 (not inclusive)**
- ***3*** **Case when computer chose rock, with 1/3 probability**
- ***4*** **Case when computer chose paper, with 1/3 probability**
- ***5*** **Case when computer chose scissors, with 1/3 probability**

35.3.3. Replicating results by using a seed

When you have programs that don't produce the results you want, you need to test them to figure out where the problem is. Programs that deal with random numbers add a layer of complexity; a program involving a random

Random numbers generated by the random library aren't truly random. They're pseudo-random. They appear random, but are determined by the result of applying a function on something that changes frequently or is unpredictable, such as the time in milliseconds since a specific date. The date generates the first number in the pseudo-random sequence, and each number in that sequence is generated from the previous number. The random library allows you to *seed* the random numbers generated by using `random.seed(N)`, where `N` is any integer. Setting the seed allows you to start with a known number. The sequence of random numbers generated within your program will be the same every time you run the program, as long as the seed is set to the same value. The following lines generate a random integer between 2 and 17 and then between 30 and 88:

```
import random
print(random.randint(2,17))
print(random.randint(30,88))
```

If you run this program many times, the numbers printed will likely change. But you can seed the results so that the two numbers are going to be the same every time the program is run, by setting the seed:

```
import random
random.seed(0)
print(random.randint(2,17))
print(random.randint(30,88))
```

The program now prints 14 and 78 every time it's run. By changing the integer inside the seed function, you can generate a different sequence. For example, if you change `random.seed(0)` to `random.seed(5)`, this program will now print 10 and 77 every time it's run. Note that these numbers may change if you're using a Python version other than 3.5.

Quick check 35.3

Q1:

Write a program that simulates flipping a coin 100 times. Then it prints how many heads and how many tails came up.

35.4. TIMING PROGRAMS WITH THE TIME LIBRARY

When you start dealing with programs that might take a long time to run, it'd be nice to know how long they've been running. The time library has functions that can help you do this, and its documentation is available at https://docs.python.org/3.5/library/time.html.

35.4.1. Using the clock

Computers are pretty fast, but how quickly can they do simple calculations? You can answer this question by timing a program that gets the computer to count up to one million. Listing 35.3 shows code that does this. Just before a loop that increments a counter, you save the current time on the computer clock. Then you run the loop. After the loop, you get the current time on the computer again. The difference between the start and end times tells you how long the program took to run.

Listing 35.3. Using the time library to show how long a program takes to run

```
import time                          1

start = time.clock()                 2

count = 0                            3
for i in range(1000000):             3
    count += 1                       3

end = time.clock()                   4
print(end-start)                     5
```

- ***1*** **Brings in functions defined in the time library**
- ***2*** **Gets the current time on the clock, in ms**
- ***3*** **Code that has the computer count to one million**
- ***4*** **Gets the current time on the clock, in ms**
- ***5*** **Prints the difference in start and end times**

This program took about 0.2 seconds to run on my computer. This time will vary, depending on how new and fast your computer is and how many other applications are running. If you have video streaming in the background, the computer may dedicate resources to doing that rather than running your

The time library also allows you to pause your program by using a sleep function. This stops it from executing the next line until that amount of time has passed. One use for this is to show the user a loading screen. Listing 35.4 shows how to print a progress bar that shows 10% increments every half-second. The code prints the following, with a half-second pause between each line. Can you tell why the code prints multiple stars by looking at the code? You'll have to think back to the lesson on strings and string manipulations:

```
Loading...
[              ] 0 % complete
[ *            ] 10 % complete
[ **           ] 20 % complete
[ ***          ] 30 % complete
[ ****         ] 40 % complete
[ *****        ] 50 % complete
[ ******       ] 60 % complete
[ *******      ] 70 % complete
[ ********     ] 80 % complete
[ *********    ] 90 % complete
```

Listing 35.4. Using the time library to show a progress bar

```
import time

print("Loading...")
for i in range(10):
    print("[",i*"*",(10-i)*" ","]",i*10,"% complete")
    time.sleep(0.5)
```

- ***1*** **Brings in functions defined in the time library**
- ***2*** **A loop representing 10% increments**
- ***3*** **Prints progress represented by multiple * characters**
- ***4*** **Pauses the program for half a second**

Quick check 35.4

Q1:

Write a program that generates 10 million random numbers and then prints how long it takes to do this.

SUMMARY

In this lesson, my goal was to teach you how to use libraries that other programmers have created to enhance your own programs. The libraries shown here are simple, but using them can lead to a more interesting user experience. Here are the major takeaways:

- Organizing code that deals with similar functionality in a separate file leads to code that's easier to read.
- Libraries store functions and classes related to one group of actions in one place.

Let's see if you got this...

Q35.1

Write a program that gets the user to roll a die against the computer. First, simulate the user rolling a six-sided die and show the result to the user. Then, simulate the computer rolling a six-sided die, add a 2-second delay, and show the result. After each roll, ask the user whether they want to roll again. When the user is done playing, show them how long they've been playing the game in seconds.

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

PREV
Unit 8. Using libraries to enhance your programs

NEXT
Lesson 36. Testing and debugging your programs