4.2. Modules

A **module** is a file containing Python definitions and statements intended for use in other Python programs. There are many Python modules that come with Python as part of the **standard library**. Providing additional functionality through modules allows you to only use the functionality you need when you need it, and it keeps your code cleaner.

Functions imported as part of a module live in their own **namespace**. A namespace is simply a space within which all names are distinct from each other. The same name can be reused in different namespaces but two objects can’t have the same name within a single namespace. One example of a namespace is the set of street names within a single city. Many cities have a street called “Main Street”, but it’s very confusing if two streets in the same city have that name! Another example example is the folder organization of file systems. You can have a file called to do in your work folder as well as your personal folder, but you know which is which because of the folder it’s in; each folder has its own namespace for files. Note that human names are not part of a namespace that enforces uniqueness; that’s why governments have invented unique identifiers to assign to people, like passport numbers.

The [Python Documentation](https://docs.python.org/3.6/) site for Python version 3.6 is an extremely useful reference for all aspects of Python. The site contains a listing of all the standard modules that are available with Python (see [Global Module Index](https://docs.python.org/3.6/py-modindex.html)). You will also see that there is a [Standard Library Reference](https://docs.python.org/3.6/library/index.html) and a [Tutorial](https://docs.python.org/3.6/tutorial/index.html) as well as installation instructions, how-tos, and frequently asked questions. We encourage you to become familiar with this site and to use it often.

If you have not done so already, take a look at the Global Module Index. Here you will see an alphabetical listing of all the modules that are available as part of the standard library. Find the turtle module.

4.2.1. Importing Modules

In order to use Python modules, you have to **import** them into a Python program. That happens with an import statement: the word import, and then the *name* of the module. The name is case-sensitive. Roughly translated to English, an import statement says “there’s some code in another file; please make its functions and variables available in this file.” More technically, an import statement causes all the code in another file to be executed. Any variables that are bound during that execution (including functions that are defined) may then be referred in some way (to be discussed) in the current file.

By convention, all import commands are put at the very top of your file. They can be put elsewhere, but that can lead to some confusions, so it’s best to follow the convention.

Where do these other files that you can import come from? It could be a code file that you wrote yourself, or it could be code that someone else wrote and you copied on to your computer.

For example, if you have a file myprog.py in directory ~/Desktop/mycode/, and myprog.py contains a line of code import morecode, then the python interpreter will look for a file called morecode.py, execute its code, and make its object bindings available for reference in the rest of the code in myprog.py.

Note that it is import morecode, not import morecode.py, but the other file has to be called morecode.py.

The tests you see in your problem sets are also using a Python module that’s in the standard library, called unittest. Right now, you can’t see the code that causes those tests to run, because we have hidden it from you, but later in the course, you will learn how to write your own Unit Tests for code, and to do so, you will need to write an import statement at the beginning of your programs. Even before you learn how to write your own tests, you will see code for Unit Tests in your problem set files.

**Don’t overwrite standard library modules!**

Given the order of search for external Python modules that is described in the list above, it is possible to overwrite a standard library. For example, if you create a file random.py in the same directory where myprog.py lives, and then myprog.py invokes import random, it will import *your* file rather than the standard library module. That’s not usually what you want, so be careful about how you name your python files!

4.2.2. Syntax for Importing Modules and Functionality

When you see imported modules in a Python program, there are a few variations that have slightly different consequences.

1. The most common is import morecode. That imports everything in morecode.py. To invoke a function f1 that is defined in morecode.py, you would write morecode.f1(). Note that you have to explicitly mention morecode again, to specify that you want the f1 function from the morecode namespace. If you just write f1(), python will look for an f1 that was defined in the current file, rather than in morecode.py.
2. You can also give the imported module an alias (a different name, just for when you use it in your program). For example, after executing import morecode as mc, you would invoke f1 as mc.f1(). You have now given the morecode module the alias mc. Programmers often do this to make code easier to type.
3. A third possibility for importing occurs when you only want to import SOME of the functionality from a module, and you want to make those objects be part of the current module’s namespace. For example, you could write from morecode import f1. Then you could invoke f1 without referencing morecode again: f1().

**Note: Python modules and limitations with activecode**

Throughout the chapters of this book, activecode windows allow you to practice the Python that you are learning. We mentioned in the first chapter that programming is normally done using some type of development environment and that the activecode used here was strictly to help us learn. It is not the way we write production programs.

To that end, it is necessary to mention that many of the modules available in standard Python will **not** work in the activecode environment. In fact, only math, random, and a couple others have been ported at this point. If you wish to explore any additional modules, you will need to run from the native python interpreter on your computer.

**Check your understanding**

modules-1-1: In Python a module is:

Top of Form

A. A file containing Python definitions and statements intended for use in other Python programs.  
B. A separate block of code within a program.  
C. One line of code in a program.  
D. A file that contains documentation about functions in Python.

Bottom of Form

modules-1-2: To find out information on the standard modules available with Python you should:

Top of Form

A. Go to the Python Documentation site.  
B. Look at the import statements of the program you are working with or writing.  
C. Ask the professor.  
D. Look in this textbook.

Bottom of Form

modules-1-3: True / False: All standard Python modules will work in activecode.

Top of Form

A. True  
B. False

4.3. The random module

We often want to use **random numbers** in programs. Here are a few typical uses:

* To play a game of chance where the computer needs to throw some dice, pick a number, or flip a coin,
* To shuffle a deck of playing cards randomly,
* To randomly allow a new enemy spaceship to appear and shoot at you,
* To simulate possible rainfall when we make a computerized model for estimating the environmental impact of building a dam,
* For encrypting your banking session on the Internet.

Python provides a module random that helps with tasks like this. You can take a look at it in the documentation. Here are the key things we can do with it.

import random

​

prob = random.random()

print(prob)

​

diceThrow = random.randrange(1,7) # return an int, one of 1,2,3,4,5,6

print(diceThrow)

​

Press the run button a number of times. Note that the values change each time. These are random numbers.

The randrange function generates an integer between its lower and upper argument where the lower bound is included, but the upper bound is excluded. So, randrange(1,7) will include numbers from 1-6. If you omit the first parameter it is assumed to be 0 so randrange(10) will give you numbers from 0-9. All the values have an equal probability of occurring (i.e. the results are *uniformly* distributed).

The random() function returns a floating point number in the range [0.0, 1.0) — the square bracket means “closed interval on the left” and the round parenthesis means “open interval on the right”. In other words, 0.0 is possible, but all returned numbers will be strictly less than 1.0. It is usual to *scale* the results after calling this method, to get them into a range suitable for your application.

In the case shown below, we’ve converted the result of the method call to a number in the range [0.0, 5.0). Once more, these are uniformly distributed numbers — numbers close to 0 are just as likely to occur as numbers close to 3, or numbers close to 5. If you continue to press the run button you will see random values between 0.0 and up to but not including 5.0.

import random

​

prob = random.random()

result = prob \* 5

print(result)

​

It is important to note that random number generators are based on a **deterministic** algorithm — repeatable and predictable. So they’re called **pseudo-random** generators — they are not genuinely random. They start with a *seed* value. Each time you ask for another random number, you’ll get one based on the current seed attribute, and the state of the seed (which is one of the attributes of the generator) will be updated. The good news is that each time you run your program, the seed value is likely to be different meaning that even though the random numbers are being created algorithmically, you will likely get random behavior each time you execute.

**Check your understanding**

modules-2-1: The correct code to generate a random number between 1 and 100 (inclusive) is:

Top of Form

A. prob = random.randrange(1, 101)  
B. prob = random.randrange(1, 100)  
C. prob = random.randrange(0, 101)  
D. prob = random.randrange(0, 100)  
Check MeCompare me

Bottom of Form

modules-2-2: One reason that lotteries don’t use computers to generate random numbers is:

Top of Form

A. There is no computer on the stage for the drawing.  
B. Because computers don’t really generate random numbers, they generate pseudo-random numbers.  
C. They would just generate the same numbers over and over again.  
D. The computer can’t tell what values were already selected, so it might generate all 5’s instead of 5 unique numbers.

# 4.4. Glossary

**deterministic**

A process that is repeatable and predictable.

**documentation**

A place where you can go to get detailed information about aspects of your programming language.

**module**

A file containing Python definitions and statements intended for use in other Python programs. The contents of a module are made available to the other program by using the import statement.

**namespace**

A naming system for making names unique, to avoid duplication and confusion. Within a namespace, no two names can be the same.

**pseudo-random number**

A number that is not genuinely random but is instead created algorithmically.

**random number**

A number that is generated in such a way as to exhibit statistical randomness.

**random number generator**

A function that will provide you with random numbers, usually between 0 and 1.

**standard library**

A collection of modules that are part of the normal installation of Python.

Bottom of Form

Bottom of Form