Using Modules

by Sophia

WHAT'S COVERED

In this lesson, we'll be looking at the random module and utilizing some of its built-in functionality. Specifically, this lesson covers:

1. Recap

If we are writing a program and wish to make use of methods and properties that are contained in a module, we need to import the module into our program first.

In the prior lesson, we saw that there are several ways to import a module or part of a module. As a recap, let's revisit the import practices.

Importing the Entire Module

One common way to gain access to the methods and properties in a module is to import the module using the `import` keyword and importing the entire module.

EXAMPLE

```
import math
```

Once we import the module, to call a method or property from the module, we include the module name, a period (.), and the name of the function that we're calling.

```
import math

print(math.pi)
```

The output is as expected.

```
3.141592653589793
```

Import Parts of a Module

Another way to gain access to the methods and properties in a module is to import parts of a module. This still uses the `import` reserved keyword but also uses the `from` reserved keyword.

EXAMPLE

```
from math import pi
```

When we import part of a module using this syntax, we can call the imported item (pi, in this case) by its own name.

EXAMPLE

```
from math import pi

print(pi)
```

Finally, we discussed using the * (asterisk) character when importing modules; this imports everything in.

EXAMPLE

```
from math import *

print(pi)
```

We talked about the reason to avoid using the * (asterisk) character and importing the entire module that way. Typically, it is better to import the module (like “import math”), than use the method or property we wish from that module.

THINK ABOUT IT

You may be thinking, “But if I import an entire module using the * (asterisk) character, I can take advantage of not needing the prefix when calling a function. So, I can state a call like this:”

```
print(pi)
```

Instead of like this:

```
print(math.pi)
```

That is true, and maybe if you are only importing one module, that may work for you, but issues can start to appear when importing more than one module into a program. More than one module can use the same function name. Then, if a program calls a function two or more times, we may have an issue with name collisions. If you remember, name collisions can happen when similar named objects are used in multiple scopes that mean different things. When a name collision occurs, that’s when we need to debug the problem since it may not be clear which function was called.

It's not a best practice to import an entire module using the * (asterisk) character. To reduce the possibility of name collisions, either import the module without the from keyword (and use the prefixes) or just import the function(s) that we're going to use.

TRY IT

Directions: Before moving on, try importing a module using both the `import` and `from` keywords.

2. Module Example: Random

Python provides many existing functions within modules that we can use in our programs. To use an existing module, we first import it, and then we can call its functions, just as if they were defined in our existing code. One module that we imported and used in a previous example was the `random` module.

If you recall from the previous lesson, we can learn more about a module by using the `help()` function.

EXAMPLE

```
import random

help(random)
```

TRY IT

Directions: Import the `random` module and run the `help()` function on it.

```
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.10/library/random.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
        bytes
        -----
```

```
               uniform bytes (values between 0 and 255)

        integers
        --------
               uniform within range

        sequences
        ---------
               pick random element
               pick random sample
               pick weighted random sample
               generate random permutation

        distributions on the real line:
        ------------------------------
```

You now see a very lengthy description of all the properties and methods that this module contains. You would need to scroll for quite some time to see all the details.

Help Note: If you are having difficulty scrolling your IDE output, you may see a “--More–” label at the bottom of the output. Use your ‘spacebar’ key to move one page at a time. Once the details are shown, your scrolling should work once again.

Depending on the existing module, details can become a bit lengthy. For now, we are going to review a specific section of all these details called Functions. This section is near the bottom of the detailed output. We are going to focus on a few functions in particular.

```
    randint(a, b) method of Random instance
        Return random integer in range [a, b], including both end points.

    random() method of Random instance
        random() -> x in the interval [0, 1).

    randrange(start, stop=None, step=1) method of Random instance
        Choose a random item from range(start, stop[, step]).

        This fixes the problem with randint() which includes the
        endpoint; in Python this is usually not what you want.
```

These are some of the basic functions that exist in the `random` module. These include the `random()`, `randint()`, and `randrange()` functions.

The examples that follow show how these functions work.

random() function

This first example shows how the `random()` function works.

EXAMPLE

```
# the use of the random() function
number = random.random()                    # a float value >= 0.0 and < 1.0
number = random.random() * 100              # a float value >= 0.0 and < 100.0
```

The **random() function** returns a float value greater than or equal to 0.0 and less than 1.0. If you want larger values, you can multiply the value that's returned. For instance, if you multiply the returned value by 100, you get a value from 0.0 to less than 100.0. If you multiply it by 70, you get a value from 0.0 to less than 70.0.

TRY IT

Directions: Using the example code below, try running it a few times to see what the `random()` function returns.

EXAMPLE

```
import random
# the use of the random() function
number1 = random.random()                   # a float value >= 0.0 and < 1.0
print(number1)
number2 = random.random() * 100             # a float value >= 0.0 and < 100.0
print(number2)
```

randint() function

This next example demonstrates the use of the `randint()` function.

EXAMPLE

```
# the use of the randint() function
number = random.randint(1, 100)             # an int from 1 to 100
number = random.randint(101, 200)           # an int from 101 to 200
number = random.randint(0, 7)               # an int from 0 to 7
```

The **randint() function** returns a random integer that's equal to or between the two values that are supplied as arguments.

TRY IT

Directions: Using the example code below, try running it a few times to see what the `randint()` function returns. Try changing the values to get different results.

EXAMPLE

```
import random
# the use of the randint() function
```

```
number1 = random.randint(1, 100)        # an int from 1 to 100
print(number1)
number2 = random.randint(101, 200)      # an int from 101 to 200
print(number2)
number3 = random.randint(0, 7)          # an int from 0 to 7
print(number3)
```

randrange() function

The following example shows how the `randrange()` function works.

EXAMPLE

```
# the use of the randrange() function
number = random.randrange(1, 100)       # an int from 1 to 99
number = random.randrange(100, 200, 2)  # an even int from 100 to 198
number = random.randrange(11, 250, 2)   # an odd int from 11 to 249
```

It works similarly to the `range()` function that you learned about in a prior lesson. However, the **randrange() function** returns a random integer value within the range of integers that are specified. The default step value is 1, but you can change that by adding the optional third argument.

With the `randrange()` function, we can see that the function definition looks like the following when we use the `help()` function on this module.

Help on method randrange in module random:

```
randrange(start, stop=None, step=1) method of Random instance
        Choose a random item from range(start, stop[, step]).

        This fixes the problem with randint() which includes the
        endpoint; in Python this is usually not what you want.
```

Referring to the code example above, there is a starting value (start) where a number would be entered, as well as the stopping value (stop) where a number would be entered, so the range would be up to that stop entered number. The step amount is a random number that would increase from the starting number. For example, if we passed in no value or a 1 (which is the default), numbers would be selected from the full sequence of integers between, and including, the starting and ending values. If we entered a 2, the random selectable number would go up by 2 starting from the start number. For example, if we use 2 for the starting value with a stopping value of 10, and had the step set to 2, it would look like (2,10,2), and the numbers that could be returned would be 2, 4, 6 or 8.

Let’s see an example of the `randrange()` function by using a guessing game that we worked on in a previous unit. We have commented out the lines that are used to follow the user's guess to slim down the game to the essential lines for actual execution.

```
#We are importing the random module that we need to be able to generate random numbers
import random

#We are creating a random even number between 2 and 10 by
#first randomizing a number between 1 and 5. This will be our
#final number. The number to add will take that and multiply it by 2.
numberToGuess = random.randrange(1, 5)
print(“The random number generated is: “, numberToGuess)
numberToAdd = numberToGuess * 2

#Asking the user to enter in their name
name = input("Hello! What is your name? ")

#Script to walk through each of the steps
print("Welcome " +name +", we’ll perform some mind reading on you.")

enteredNumber = int(input("Enter in a number between 1 and 10: "))

print("Multiply the result by 2.")

#userNumber = enteredNumber * 2
#print(">> userNumber at this step = " + str(userNumber))

answer = input("Ready for the next step? ")
print("Now, add...let's see...")
print(“The number to add is: “, numberToAdd)

#userNumber = userNumber + numberToAdd
#print(">> userNumber at this step = " + str(userNumber))

answer = input("Ready for the next step? ")
print("Now, divide the number have by 2.")

answer = input("Ready for the next step? ")

#userNumber = userNumber / 2
#print(">> userNumber at this step = " + str(userNumber))

print("Now, subtract the original number that you thought about.")
answer = input("Ready for the last step? ")
```

```
#Guessing the number
print("Well " +name +", let me read your mind...The number that you have right now a....")
print(numberToGuess)

#userNumber = userNumber - enteredNumber
#print(">> userNumber at this step = " + str(userNumber)
```

Does this guessing game look familiar? We looked at this example back at the end of Unit 1. We also added a new line to the program that prints out what the `randrange()` function originally returned.

TRY IT

Directions: Add the game into the IDE and try it out again. Notice there is really no “mind” guessing about it. It is simply the random number returned by the `randrange()` function and simple calculation.

In this example, we had a number defined to be randomly selected between 1 and 5. However, if we want to have the random number be in a different range, we can change the starting and ending value like this:

```
numberToGuess = random.randrange(20, 100)
```

This would give us a number to choose between 20 and 100. However, we have a third parameter that we can pass into the `randrange()` function to identify the step. The step value means the amount the number increases by in random values between the starting and ending number. For example, if we wanted the odd numbers between 1 and 10, we would do the following:

```
numberToGuess = random.randrange(1, 10, 2)
```

This would return a number that would be 1, 3, 5, 7 or 9. This is because the step is set to 2, the number starts at 1 and ends at 10. However, 10 is not in the step value. If we wanted an even number, we would do the following:

```
numberToGuess = random.randrange(2, 10, 2)
```

This would return a number that would either be 2, 4, 6, or 8. If we wanted to get all of the multiples of 5 between 5 and 50, we would do the following:

```
numberToGuess = random.randrange(5, 50, 5)
```

TRY IT

Directions: Try changing the values of the `randrange()` function and see if you can predict the output.

There are many different modules that exist in Python’s standard library. We just used functions of the `random` module so we did not have to develop code from scratch. We made use of these existing modules and used them as needed. It can be useful to become familiar with the list of standard modules within Python.

docs.python.org/3/py-modindex.html

In this list, we can find the `random` module. Clicking through the link for the `random` module, we can find the necessary information about the module.

docs.python.org/3/library/random.html#module-random

Take the time to explore what other modules exist, and think of ways you can make use of them.

TERMS TO KNOW

random()
The `random()` function returns a float value greater than or equal to 0.0 and less than 1.0.

randint()
The `randint()` function returns a random integer that's equal to or between the two values that are supplied as arguments.

randrange()
The `randrange()` function returns a random integer value within the range of integers that's specified.

SUMMARY

In this lesson, we started with a **recap** of ways to import a module or part of a module. We learned how to make use of existing modules. We learned more about a specific module, the `random` **module**, and explored it with the `help()` function. We also looked at how to use some functions within the module based on the information provided in the module help. Finally, we used some of the common functions within the `random` module, including the `random()`, `randint()`, and `randrange()` functions.

Best of luck in your learning!

TERMS TO KNOW

randint()
The `randint()` function returns a random integer that's equal to or between the two values that are supplied as arguments.

random()
The `random()` function returns a float value greater than or equal to 0.0 and less than 1.0.

randrange()

The `randrange()` function returns a random integer value within the range of integers that’s specified.