**1.4 Modules**

We have seen that Python has built-in functions. Many other functions are stored in ***modules***. Modules in Python can be used to store groups of related functions (and sometimes other definitions). Modules are separate from the core Python, and must be ***imported*** before the functions contained in them can be used. A module is imported using the **import** statement, and then functions contained in the module can be referenced using the ***dot operator***. Modules are stored in files that have a .py extension on the filename.

There are modules that are built into Python’s ***standard library***, and you can also define your own modules. Some useful standard modules include the **math** module, the **random** module, and the **statistics** module. There are also ***external library modules*** that have been developed and are widely distributed. Some of these, such as **numpy**, **pandas**, and **scipy**, contain functions

that are useful in data science applications. The **matplotlib** module has functions that help to visualize data. These modules will be introduced in Part II of this book.

The **math** module contains math functions, as well as some built-in ***constants***. Unlike variables, constants cannot change. The constants in the **math** module include:

pi 3.14159

e 2.71828

tau 6.28318

inf infinity

nan not a number

Notice that the constant **tau** is equivalent to 2\***pi**. The constant **e**, called Euler’s number, is the base for natural logarithms. Do not confuse the constant **e** here with the e used in scientific notation!

Although by convention user-defined constants are variables that are named using upper case letters, the constants that are in the **math** module are named using lower case letters. Unfortunately, they are not true constants in that their values can be modified.

To use one of them, to calculate π\*radius2, for example, we might enter the following:

*>>> import math*

*>>> radius = 3*

*>>> math.pi\*radius\*\*2*

28.274333882308138

The **import** statement imports the **math** module and then the dot operator is used as math.pi to refer to the constant **pi**.

The **round** function could be used here to reduce the displayed number of decimal places.

Note that to use the constant **pi**, it is necessary to give the name of the module, the dot operator, and then the name of the constant. In order to make this simpler, and avoid the use of the dot operator, certain or all constants and functions from a module can be imported using the **from** statement. For example, to just import **pi**:

*>>> from math import pi*

After importing **pi**, just the name **pi** can be used without first specifying **math** and the dot operator.

*>>> from math import pi*

*>>> radius = 3*

*>>> round(pi\*radius\*\*2,2)*

28.27

To import both **pi** and **e**:

*>>> from math import pi, e*

To import everything:

*>>> from math import \**

The \* is sometimes called a ***wildcard*** and in this case means everything. Importing everything from a module is not generally recommended since you probably do not know the names of everything that will be imported, and some names could interfere with names that you already have (variable names, for example).

Once the **math** module has been imported, all of the constants and functions can be seen using the **help** function.

*>>> help(math)*

The **math** module contains many useful math functions. These include trig functions such as **sin**, **cos**, and **tan**. Another function that is used a lot is the square root function, **sqrt**. There is also a function **exp**, which returns the constant **e** raised to a specified power.

To read about a particular function, use the dot operator. For example,

*>>> help(math.sqrt)*

Help on built-in function sqrt in module math:

sqrt(x, /)

Return the square root of x.

**1.5 Objects and Methods**

Python is based on objects. Objects have special functions, called ***methods***, that are associated with every type. Methods are different from functions in that, instead of explicitly passing an expression to them in parentheses, they are called implicitly with an expression using the dot operator.

For example, for strings there is a method called **upper**, which will convert all letters of the alphabet in a string to upper case.

*>>> mychars = 'Hello123'*

*>>> mychars.upper()*

'HELLO123'

This displays the letters in the string in upper case, but it does not change the variable *mychars*. (Recall that string variables are immutable; they cannot be changed.) The string variable *mychars* is not passed explicitly to the **upper** method, but rather is passed implicitly by using the dot operator. In this case, there is no need to pass any arguments so the parentheses after the method name are empty. It is necessary, however, to have the empty parentheses. Note that any non-alphabetical characters are not changed.

If it is desired to change the variable, the result of the expression must be assigned back to the variable.

*>>> mychars = 'Hello123'*

*>>> mychars = mychars.upper()*

*>>> mychars*

'HELLO123'

Similarly, there is a method called **lower** that will convert letters in a string to lower case.

In the previous example, a variable was used as the calling expression, but that is not necessary. The calling expression in this case can be just a string.

*>>> 'Hello123'.lower()*

'hello123'

Another method for strings is **index**, which returns the index of the first occurrence of a specified character, or the beginning of a specified ***substring***. Recall that the indices begin at 0.

*>>> urname = 'monty python'*

*>>> urname.index('y')*

4

*>>> urname.index('py')*

6

Note that with the **index** method, the string is passed implicitly using the dot operator, and the substring is passed explicitly in parentheses. The call to the method urname.index(‘y’) is asking for the location of the string ‘y’ in the variable *urname*.

Sometimes it is not known whether the letters in a string are lower or upper case, so they can be converted to all lower (or upper) before finding an index.

*>>> urname = 'Monty Python'*

*>>> urname.lower().index('py')*

6

Notice the use of the dot operator twice here. First, the **lower** method was used to convert the string to lower case, and then the **index** method was used to find the location of ‘py’.

There are some methods that return **True** or **False**. For example, there are “is” methods that ask a question and return either **True** or **False**. The **islower** method will test to see whether or not all letters of the alphabet in a string are lower case.

*>>> mychars = 'Hello123'*

*>>> mychars.islower()*

False

*>>> mychars = 'hello123'*

*>>> mychars.islower()*

True

Similarly, the **isupper** method will test to see whether or not all letters of the alphabet in a string are upper case.

Another method that can be used with strings is **startswith**, which determines whether or not a string starts with a particular character or string, and returns **True** if it does or **False** if it does not.

*>>> word = "Monty"*

*>>> print(word.startswith('Mo'), word.startswith('&'))* True False

In this case the **startswith** method is called with the string variable *word*, and a string is passed as an argument to test to see whether or not the variable begins with that string (of one or more characters). Similarly, there is a method **endswith**.

More string methods will be covered in Chapter 7.

There are methods for all variable types in Python, not just strings. For **float** numbers, for example, the method **is\_integer** will return **True** if the value stored is actually an integer or **False** if it is not.

*>>> num = 33.0*

*>>> print(type(num))*

<class 'float'>

*>>> num.is\_integer()*

True

There are many methods that can be used with lists. The **index** method can be used with lists as well as strings. For example,

*>>> [4, 33, 11, 5].index(11)*

2

**1.6 Random Numbers**

It is often useful to be able to generate ***random numbers***. The **random** module has several functions that facilitate this.

The **randint** function will return a random integer. Called as **randint(a,b)**, it will return a random integer in the inclusive range from a to b. For example, the following prints a random integer in the inclusive range from 2 to 10.

*>>> from random import randint*

*>>> print(randint(2,10))*

5

The **random** function returns a random real number in the range from 0 to 1, not including 1. No arguments are passed to it, so when calling it the parentheses are empty.

*>>> from random import random*

*>>> rf = random()*

*>>> rf*

0.6405322431584719

Another function in the **random** module is the **choice** function, which chooses and returns a random item from a sequence. For example, the **choice** function can choose a random character from a string.

*>>> from random import choice*

*>>> mystring = 'abc\*&def'*

*>>> choice(mystring)*

'c'

We will see examples using other sequence types in later chapters.

**1.7 Help Utility**

We have seen that the **help** function can be used to show the functions in a given module. For example, **help(math)** will display information about the **math** module once the **math** module has been imported. To find out about a particular function in a module, use the dot operator, e.g., **help(math.sqrt).**

The **help** function can also be used to display information about a particular function that is in core Python, e.g., **help(round)** will explain the usages of the **round** function.

To find out what methods can be used with expressions and variables of a particular type, pass the name of the type to the **help** function. For example, **help(str)** will display a list of the methods that can be used with strings. To find out about a particular method, use the dot operator. For example,

*>>> help(str.islower)*

Help on method\_descriptor:

islower(self, /)

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

A slightly more readable version can sometimes be obtained using a variable that has been created:

*>>> word = "Monty"*

*>>> help(word.islower)*

Help on built-in function islower:

islower() method of builtins.str instance

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

Typing just **help()**, without any arguments, brings up a help utility, which has a special prompt of help>. At this prompt, the following can be entered:

modules shows a list of all available modules

keywords shows a list of all keywords

symbols shows a list of all symbols that are used as operators

topics shows a list of all documentation topics

quit exits the help utility

***Keywords*** (also called ***reserved words***) are words that are built in to Python and can never be used as variable names. These include commands such as **import**.