2.5 Importing Python Modules

So far we have learned about Python's built-in functions and seen some examples data type methods. However, we've been limited to just the functions and data types that are always made automatically available by the Python interpreter. It turns out that the Python programming language comes with many, many other functions (and even other data types) that are organized into different **modules**, which is another name for Python code files.

Unlike the functions and data types we've seen so far, these modules are not automatically loaded when we run the Python interpreter, as they contain more specialized functions and data types that are only useful in specific situations. In this section, we're going to learn how to load one of these modules and use their definitions, so that you can explore and use these modules in your own Python programming.

The `import` statement

To load a Python module, we use a piece of code called an **import statement**, which has the following syntax:

```
import <module_name>
```

For example, here is how we could load the `math` module in the Python console:

```
>>> import math
```

Like the other statements we've seen so far, import statements do not produce a value, but they do have an important effect. An `import` statement introduces a new variable (the name of the module being imported) that can be used to refer to all definitions from that module.

For example, the `math` module defines a function `log2` which computes the base-2 logarithm of a number. To access this function, we use dot notation:[1]

```
>>> math.log2(1024)
10.0
```

[1] This notation is the same as accessing data type methods, but `log2` is *not* a method. It's a top-level function, just one that happens to be defined in the `math` module.

What other functions are contained in the `math` module? We'll make use of a few other later in this course, but if you're curious you can call the special built-in function `dir` on the `math` module (or any other module) to see a list of functions and other variables defined in the module:

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asir
```

Ignoring the first few with the double underscore, we see some familiar looking names, like `ceil`, `floor`, `pi`, and `sin`. We've linked to the documentation for the `math` module in the References section below.

The `datetime` module

Python comes with far more modules than we'll have time to learn about in this section.[2] However, just to illustrate the breadth of these modules, we'll briefly introduce one more that will be useful occasionally this year.

[2] Or even this whole course!

The `datetime` module provides not just functions but new *data types* for representing time-based data. The first data type we'll study here is `date`, which (unsurprisingly) represents a specific date.

```
>>> import datetime
>>> canada_day = datetime.date(1867, 7, 1)  # Create a new date, 1867-07-01
>>> type(canada_day)
<class 'datetime.date'>
>>> datetime.date.weekday(canada_day)  # Return the day of the week of the date
0                                      # 0 = Monday, 1 = Tuesday, etc.
```

Note the double use of dot notation in that last expression. `datetime.date` is the data type being accessed, and `.weekday` accesses a method of that data type. Using this method tells us that Canada's confederation was on a Monday![3]

[3] Fun fact: Canada's confederation first consisted of only four provinces: Ontario, Quebec, Nova Scotia, and New Brunswick.

We can compare dates for equality using `==` and chronological order (e.g., `<` for comparing one date comes before another). We can also subtract dates, which is pretty cool:

```
>>> david_fake_birthday = datetime.date(2000, 3, 22)
>>> david_fake_birthday - canada_day
datetime.timedelta(days=48469)
```

The difference between two dates is an instance of the `datetime.timedelta` data type, which is used to represent an interval of time. What the above expression tells us is that 48,469 days have passed between David's (fake) birthday[4] and the day of Canada's confederation.

[4] Hint: one of the three arguments to `datetime.date` matches David's correct birthday.

There's a lot of Python out there—don't worry!

Up to this point, we've covered several different data types, functions, methods, and now modules in Python. It might be feeling a bit daunting, and we wanted to take a moment to pause and look at the bigger picture. Our goal in showing you these elements of Python is not to overwhelm you, but instead to give you a taste of the language's powerful computational capabilities. But this course is not about memorizing different functions, data types, and modules in Python! All throughout this course, you'll have access to references and documentation that describe the functionality of these different elements, and will have lots of opportunities to practice using them. For now, all we want you to know is simply that these capabilities exist, how to experiment with them in the Python console, and how to look up information about them using these course notes, other online documentation sources, and Python's built-in `help` function.

References

- `datetime` module documentation
- `math` module documentation