Creating Modules

by Sophia

WHAT'S COVERED

In this lesson, we'll be creating a custom module for temperature conversion to be used in another program. Specifically, this lesson covers:

1. Create a Temperature Conversion Module

There are many instances where the existing modules don't have the necessary code for us or we may simply want to create code so that we can reuse it later on. Many times developers have some standard functionality that they need to use often. Rather than rewriting it for every single program, they can store that functionality in a module and import it as necessary.

Let's take a real-world example to build off of here. We have two different functions that we've created to help with the conversion of temperature.

EXAMPLE

```
def to_celsius(fahrenheit):
  celsius = (fahrenheit - 32) * 5/9
  return celsius

def to_fahrenheit(celsius):
  fahrenheit = celsius * 9/5 + 32
  return fahrenheit
```

We have used the conversion formula a few times in prior lessons but here we have defined both as functions. The function `to_celsius` takes an argument as a temperature in Fahrenheit and will return the Celsius equivalent. The function `to_fahrenheit` does the same for Celsius and returns the temperature in Fahrenheit.

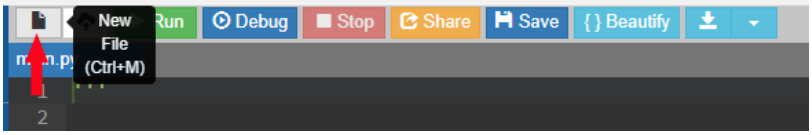
Together they are just functions. To make them reusable for multiple programs, we can turn them into a module.

Add a File

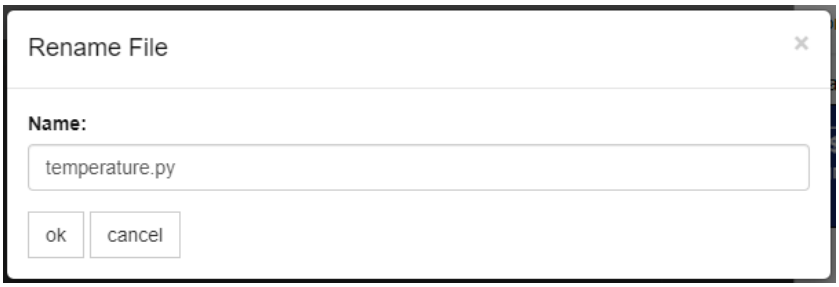
TRY IT

Directions: Follow the next few steps to create a new file.

1. The first thing we need to do in the IDE is create a new file. We can do that by selecting the "New File" icon in the top menu.

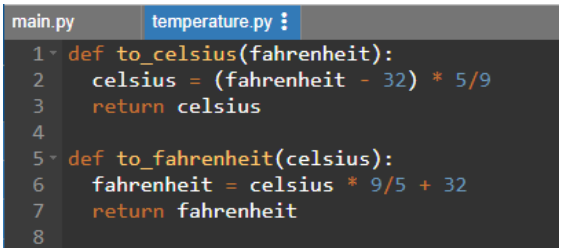


2. Next, give your new file a name. We will call it `temperature.py` since that describes the topic of what the functionality of these functions are. We are giving the filename the extension ".py" to designate this as a Python file.



Note: Ensure that the filename ends with ".py".

3. Our new file will appear as a seperate tab at the top. Finally, add the code for the two functions in the code editor pane inside the temperature.py file.



This new file that we've created called `temperature.py` contains these two functions and could be used as a module. The name of the module comes from the name of the file, so to create a module called temperature, we store the code in a file named `temperature.py`—which is what we have here.

Access to the New Module

The easiest way to make the functions in a module available to other programs and/or modules is to store the module file in the same directory as the other modules. In our example, we notice that our new file now exists next to the `main.py` file. The **main.py** is the top-level file for all Python programs. Anytime you create a new Python program (new project), `main.py` will automatically be created for you to use. So, both the `temperature.py` and `main.py` are in the same directory/location and can use each other.

If we want to store the module file in a central location and access it from multiple directories, we need to store the module in a directory that's in the search path. Remember when we were importing the `math` module? If any changes or additions were made to the files in this module, they would automatically be imported in. However, if we placed the file outside of any modules or packages that we import, Python will not know this new module exists unless we directly import it in. The search path includes all of the packages and modules that have been imported into our program. How we set this path varies from system to system so we can't demonstrate that directly here within this IDE.

Using Our File as a Module

To use our functions in the `temperature.py` file as an imported module, let's go back to the `main.py` file.

Let's use the correct syntax and `import` the `temperature.py` file. Remember, now that it exists as a Python file, we only need to add the module name.

↷ EXAMPLE

```
import temperature

temp = 20
print(temp, "Celsius =", round(temperature.to_fahrenheit(temp)), "Fahrenheit")
print(temp, "Fahrenheit =", round(temperature.to_celsius(temp)), "Celsius")
```

In this example code, we are importing the `temperature.py` module. We also added some other statements that set a variable called `temp` and use that variable within two `print()` functions. We see the module name, then (.) dot followed by the two temperature functions in the module that use the `temp` variable. This is all encompassed in a string and output to the screen.

```
20 Celsius = 68 Fahrenheit
20 Fahrenheit = -7 Celsius
```

TRY IT

Directions: Go ahead and add the code in the example above to the `main.py` file and run the program. How cool is that? We have created a module and used it.

Can Others Use It?

BIG IDEA

In order to share code with others, you could simply send the individuals the .py files that contain the modules. There are also other approaches using tools like GitHub in which these files can be shared. GitHub is a centralized repository where programs can be saved under your own account and kept track of within different versions.

Documenting the Module

If a module is going to be shared and used by other programmers, it's good to document the functions of the module. To do that, we can use docstring. A Python **docstring** (short for documentation string) is a string used to document a Python module, class, function, or method. This makes it easy for someone to understand what that object does.

Initially, there is some basic information about the module. If we import our temperature module and use the `help()` function, we do see a little detail.

EXAMPLE

```
import temperature

help(temperature)
```

Here is that output.

```
Help on module temperature

NAME
    temperature

FUNCTIONS
    to_celsius(fahrenheit)

    to_fahrenheit(celsius)

FILE
```

```
/home/temperature.py
```

The `help()` function details are very simple: the module name and what functions exist in the module.

Adding docstrings helps to add additional details on modules. Remember when we used the `help()` function on the random module? There were pages and pages of documentation to scroll through. This is very useful, especially for other programmers who did not develop the module to know how particular methods or properties of a module function work. Adding docstrings is always a good practice.

First, there are a few items to keep in mind when writing docstrings.

- A docstring begins and ends with three double-quotes. Like the hashtag (#) is used for commenting, these three double quotes let Python know where the documentation starts and ends.

EXAMPLE

```
"""
This is details about this module
"""
def my_function(parm):
  """
  This is details for this function
  more details
  """
  return(parm)
```

- Secondly, docstrings must start at the proper indentation levels. Notice in the example code above, the first docstring that details the module is not indented at all, which is correct since it is not connected to an internal function or method. The docstring inside the function is indented along with the statements inside the function. If these indentations were not correct, we would get an error when trying to run it.

Let's go ahead and add some docstings to our `temperature.py` file.

```
"""
This module contains functions for converting temperature between degrees Fahrenheit
and degrees Celsius
"""

def to_celsius(fahrenheit):
  """
  Accepts degrees Fahrenheit (fahrenheit argument)
  Returns degrees Celsius
  """
  celsius = (fahrenheit - 32) * 5/9
  return celsius

def to_fahrenheit(celsius):
  """
  Accepts degrees Celsius (celsius argument)
  Returns degrees Fahrenheit
  """
```

```
    fahrenheit = celsius * 9/5 + 32
    return fahrenheit
```

Here we added some documentation on what the module contains and more details on what each function does.

TRY IT

Directions: Go back to the `temperature.py` file and add the docstrings to the code. Remember to always use three double quotes to start and end the docstrings. Also, watch out for the proper indentations.

Now let's see what this looks like with those docstrings added in, using the `help()` function again.

EXAMPLE

```
import temperature

help(temperature)
```

TRY IT

Directions: Back in the `main.py` file, try running the program again.

```
Help on module temperature:

NAME
    temperature

DESCRIPTION
    This module contains functions for converting temperature between degrees Fahrenheit and degrees Celsius

FUNCTIONS
    to_celsius(fahrenheit)
        Accepts degrees Fahrenheit (fahrenheit argument)
        Returns degrees Celsius

    to_fahrenheit(celsius)
        Accepts degrees Celsius (celsius argument)
        Returns degrees Fahrenheit

FILE
    /home/temperature.py
```

Now we see the added description to the top of the module and more details at each function level. This is a lot clearer with the description and explanation of what the code does.

It's a good practice to use docstrings to document the purpose of the modules and functions that might be used by other programmers. Remember to display the documentation for a module, import the module, and pass the name of the module to the `help()` function.

TRY IT

Directions: Give it a try to see how you could extend this program to create a program to convert Celsius to Kelvin and Kelvin to Celsius. How would you format and update the code so that it follows the same guidelines as the Celsius to Fahrenheit?

TERMS TO KNOW

main.py

The `main.py` is the top-level file for all Python programs. Anytime you create a new Python program, `main.py` will automatically be created for you to use.

Docstring

A Python docstring (short for documentation string) is a string used to document a Python module, class, function, or method. This makes it easy for someone to understand what that object does.

SUMMARY

In this lesson, we created a temperature **conversion module**. We started by creating a new file called `temperature.py`. This file contained two functions that converted Celsius and Fahrenheit to one another. We used this new file as a module import to another program and used it for conversions. We also added docstrings into our temperature module to make it clearer for future developers to make use of those modules as needed.

Best of luck in your learning!

TERMS TO KNOW

Docstring

A Python docstring (short for documentation string) is a string used to document a Python module, class, function or method. This makes it easy for someone to understand what that object does.

main.py

The main.py is the top-level file for all Python programs. Anytime you create a new Python program, main.py will automatically be created for you to use.