Python Modules

Python organizes code into units called *modules*. A module has its own namespace (meaning that the names of variables, functions, and other Python objects are unique to that module and do not overwrite the names in other modules) and can contain arbitrary Python code. You can access the code in a module by *importing* the module.

Module Import

You import a module using the **import** **statement**, as follows:

import math

print(math.pi)

Line 1 shows how to import the math module. The \color{red}{\verb|import|}import keyword is followed by the name of the module (in this case \color{red}{\verb|math|}math). There are other forms of the import statement, but this is the basic form that we will use throughout this specialization. Once you import a module, you then have access to all of the components of that module. However, because the module has its own namespace, the names of the functions and constants within the module are not directly accessible. If they were, there would be potential conflicts between the names in the imported module and the names in the current module. To resolve this, to access objects in the imported module, you must prefix module objects with the module name and a dot, as is done on line 3 of the above code to access the constant \color{red}{\verb|pi|}pi from within the \color{red}{\verb|math|}math module.

Python Modules

Python has numerous built-in modules. You can find an exhaustive list of these built-in modules in the [Python Library Reference](https://docs.python.org/3/library/index.html).

You can also write your own modules. In fact, every Python file is its own module. You can import it into code in another file with the import statement. The name of the module is simply the file name without the ".py" extension. So, for example, if you create a file called "utilities.py", you can import it into another file with the statement \color{red}{\verb|import utilities|}import utilities.

Imports work exactly the same way in CodeSkulptor3. If you want to import a file that you wrote yourself, you simply use the file name without the ".py" extension. So, for example, if you save a file and it gets the name "user300\_abc123.py", you can import it into another file in CodeSkulptor3 with the statement \color{red}{\verb|import user300\_abc123|}import user300\_abc123.

Using "as"

Sometimes the module name is long and/or cryptic (which is the case with your CodeSkulptor3 files). Python allows you to change the name you use to refer to the module within your code by using the \color{red}{\verb|as|}as modifier. For example, you can do the following:

import user300\_abc123 as mycode

print(mycode.myfunction(3))

As this example shows, you can now access the function \color{red}{\verb|myfunction|}myfunction within the module \color{red}{\verb|user300\_abc123|}user300\_abc123 using \color{red}{\verb|mycode.myfunction|}mycode.myfunction instead of \color{red}{\verb|user300\_abc123.myfunction|}user300\_abc123.myfunction. Obviously, you should name the module (and function within it) with more descriptive names indicating what they are.