Module 1 - Overview

Introduction

Module Learning Outcomes

After successful completion of this module, you will be able to ...

1. Import a Python module and use it in your code.
2. Create your own Python modules.
3. Install a Python package in PyCharm.
4. Explain what virtual environments are and why they're useful.

Key questions:

* What is a Python module?
* How do you import a Python module?
* What does a main function do?
* What is a package in Python?
* What is a virtual environment for?

Explorations

Use the pages within this module to explore the following concepts:

* Exploration: [Classes and objects (review)](https://canvas.oregonstate.edu/courses/1915078/pages/exploration-classes-and-objects-review) (CLO 1c)
* Video Demo: [Classes and objects (review)](https://canvas.oregonstate.edu/courses/1915078/pages/video-demo-classes-and-objects-review) (CLO 1c)
* Activity: [Classes and Objects](https://canvas.oregonstate.edu/courses/1915078/pages/activity-classes-and-objects) (CLO 1c)
* Exploration: [Importing modules, installing packages, virtual environments](https://canvas.oregonstate.edu/courses/1915078/pages/exploration-importing-modules-installing-packages-virtual-environments) (MLOs 1-4)
* Video Demo: [Importing modules](https://canvas.oregonstate.edu/courses/1915078/pages/video-demo-importing-modules) (MLOs 1, 2)
* Activity: [Debugging Skills Review](https://canvas.oregonstate.edu/courses/1915078/pages/activity-debugging-skills-review)
* [Module 1 exercise solutions](https://canvas.oregonstate.edu/courses/1915078/pages/module-1-exercise-solutions)

Optional Resources

* [*Think Python* Chapter 14 section 9Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2015.html#sec173)

Task List

Complete the following assignments and other tasks:

* Read the Exploration pages (linked to above) and do the interactive exercises on those pages (CLO 1c, MLOs 1-4).
* Complete [Assignment 1](https://canvas.oregonstate.edu/courses/1915078/assignments/9227000), which gives you an opportunity to practice the Python skills you learned in CS 161, and gives you practice with importing a module and using its functionality (CLO 1c, MLO 1).
* Take [Quiz 1](https://canvas.oregonstate.edu/courses/1915078/quizzes/2859157) (MLOs 1-4).

# Exploration: Importing modules, installing packages, virtual environments

## Modules

A **module** is just a normal Python file that contains definitions of functions or classes that you would like to use in another file. Let's say you have a file named "mystats.py" that defines some classes and functions for statistical computations. If you want another file to be able to use them, you can put an import statement at the top of your code like in the code example below.

Let's say that a function named find\_mean is defined in mystats.py. Because we imported mystats, we can now use that function, but we have to put "mystats." in front of the function name.

Graphical user interface, text, application, email

Description automatically generated

If we'd rather not have to put "mystats." in front of a function or class name, we can use this form of the import statement:

from mystats import find\_mode

Then we could use find\_mode in the usual way:

mode\_val = find\_mode(some\_list)

If there were multiple functions and/or classes we want to import from the same module, we could list them like this:

from mystats import find\_mode, find\_mean, find\_median

If you want to do this for all of the classes and functions in the module, but don't want to have to list them all, you can do this:

from mystats import \*

There are lots of useful modules that come as part of the Python standard library, which is what people mean when they say Python comes with "batteries included". There are modules for various types of math, working with files and directories, data compression, cryptography, email, web pages, etc. There are also a ton of third-party modules which are provided in "packages" you can install, which we'll discuss in a moment.

### **Main functions**

Say we have the following Python file. If you look on the left, under "work", you can see the files example.py and calc.py. You can click on a file to see its code.

Graphical user interface, text, application

Description automatically generated

If we run this file as a script (or program, for example when you run a code file in PyCharm), the assignment and print statements will execute, which is what you would expect. However, we might want to also be able to import the file as a module so that other programs can use the mult and add functions. If you do import it as a module, as in the code example above, you can see how the assignment and print statements will execute when the file is imported, which might not be what you want. It would be nice if we could have code that executes when the file is run as a script, but doesn't execute when the file is imported. Happily, we can achieve this by using a main function.

Here's an example that modifies our previous code:

Graphical user interface, text, application, email

Description automatically generated

We've enclosed inside the main() function the code that should execute when the file is run as a script. The if statement at the bottom determines whether the main function is called. If the file is being run as a script, the special variable \_\_name\_\_ will equal '\_\_main\_\_' and the main function gets called. If the file is being imported as a module, then \_\_name\_\_ will equal the name of the module and the main function does not get called. The interpreter doesn't actually care if you give the main function a different name, but you should use the usual name so its intent is clear to other people reading your code.

## Packages

**Packages** are a way to group and organize modules. If our mystats module were part of a mymath package, we could import it like this:

import mymath.mystats as stats  
mode\_val = stats.find\_mode(some\_list)

The **as** keyword lets us assign an easier name, so we don't have to keep typing "mymath.mystats". The as keyword can also be used with modules that are not part of a package, but individual module names don't usually need to be simplified.

We could also import just the find\_mode function like this:

from mymath.mystats import find\_mode  
mode\_val = find\_mode(some\_list)

### **Installing a package in PyCharm**

In PyCharm, you can install a package for a particular Python interpreter by following these steps:

1. Click on the "Python Packages" tab that appears at the bottom of the screen when you have any project open.
2. By default you will see a list of currently installed packages.
3. You can use the search bar to find the package you want to install. Try typing "numpy" into the search bar. Now the list is only of packages whose names contain "numpy".
4. With the first one selected - just plain "numpy" - click the "Install" button at the right side of the window.
5. Now if you clear the search window, you should see "numpy" in your list of installed packages.

Another way to install a package in PyCharm is:

1. In Pycharm open a new Python file.
2. Write an import statement in your file that imports the desired package.
3. Hover over the name of the package (which will be underlined in red).
4. Click on “Install package” button that pops up.
5. Wait for the Installation to finish.

## Virtual environments

Virtual environments are the solution to a problem that you hopefully haven't encountered yet, which is how to allow different projects to use different versions of the Python interpreter, as well as different versions of various packages. Software developers generally try to maintain backward compatibility, so that new versions of interpreters or packages don't break old code, but sometimes a change seems beneficial enough to be worth it. The biggest example of this for the Python interpreter is the jump from Python 2 to Python 3. There needs to be a way to use Python 3 for new projects, but still be able to run the old Python 2 projects. The same thing goes for different versions of packages. If a package gets updated in a way that breaks your code, you still want to be able to run your existing projects.

A virtual environment allows a project to have its own tailored environment with its own specific versions of the interpreter and of whatever packages it needs. You can have any number of virtual environments, and each one can run independently of the needs of other projects. No man is an island, but a Python project can be.

There are a few different tools for setting up virtual environments, the main ones right now being venv and conda. We won't go into the details here of how to create a virtual environment. If you've needed one already because of some conflict on your system, you presumably found out and were helped with that in CS 161. If you haven't needed one already, then you shouldn't need one for this course. However it's important to be aware of virtual environments because you will need them sooner or later.

## Exercises

Try these out on your computer using PyCharm:

1. Write a function named distance that takes four parameters: the x- and y-coordinates of the first point, followed by the x- and y-coordinates of the second point. It should return the distance between those two points, using the Pythagorean Theorem, for which you will need to import the **[math](https://docs.python.org/3/library/math.html" \t "_blank)**[Links to an external site.](https://docs.python.org/3/library/math.html" \t "_blank) module and use the math.sqrt() function.

Example1: d = distance(3, 5, -1, 2); Your function should return d = 5

Example 2: d = distance(0, 0, 0, 0); Your function should return d = 0

2. Write a main function that will execute and call the distance function if the file is run as a script, but not if the file is imported into another file. Test it out by running it as a script and by creating another file that imports it and running that file as a script.

Exploration: Classes and Objects (review)

Introduction

You've seen a few of the different types that are built into Python. Using *classes*, we can define new types. Once we've defined a class (a user-defined type), we can then create specific objects of that class. Here's an illustration of the relationship between a class and a few specific objects of that class:

Diagram

Description automatically generated

As you learned in the previous module, functions are a way of helping organize code in a simpler way. Instead of having to repeat the same code in many different places, you can just have it in one place, which makes it much easier to think about and maintain. Classes provide an additional level of helping organize code in a simpler way. Instead of having various related data and the various functions that need to operate on that data just scattered around various parts of your code, you can bundle together that data and those functions into a single class, which makes it much easier to think about and maintain. With the small programs you're working on now that may not be apparent yet, but when we get to larger programs, the benefits should become more clear.

Here's a simple example class:

Graphical user interface, text, application

Description automatically generated

The **class** keyword is used to define a class. The name of a class should start with a capital letter. Classes should have a docstring that describes their purpose and any additional information someone might need to know when using the class. Functions that are part of a class are often referred to as **methods**. This Pet class only has a single method with the special name **\_\_init\_\_** (two underscores on either side). This method creates a Pet with whatever species and name you passed to that method. The species and name are the **data members** (or fields or attributes) of the Pet class. The special parameter **self** refers to the object itself. You don't call the \_\_init\_\_ method by name - instead you use the name of the class. You also don't pass an argument for self - that's done automatically for you. Let's create a couple of Pets:

> pet\_1 = Pet("capybara", "Beatrice")  
> pet\_2 = Pet("kangaroo", "Joey")

Each of these lines creates a new Pet **object**. The Pet class is just a general blueprint for Pets. To create a specific Pet object, we use the name of the class as shown above. This automatically calls the \_\_init\_\_ method. An object is sometimes referred to as an "instance" of a class, and creating an object as "instantiating" a class. Once we've created an object, we can access its attributes using dot notation:

> print(pet\_1.\_species)  
> print(pet\_1.\_name)

We would also use dot notation to call an object's methods, but our Pet class doesn't have any besides \_\_init\_\_, so let's look at another example:

Graphical user interface, text, application, email

Description automatically generated

Notice that the area and perimeter methods have only the self parameter. They don't need to be passed the width or height, because they are already part of the object, and can be accessed via the self parameter. Now that we've defined the Rectangle class, here's an example of using it:

> rect = Rectangle(2.5, 3)  
> print("area = ", rect.area())  
> print("perimeter = ", rect.perimeter())

Note that we call the method on a Rectangle **object**, NOT on the Rectangle class:

> Rectangle.area()  # WRONG  
> rect.area() # CORRECT (instead of rect, it could be the name of any Rectangle object)

How do you know what to put in a class?

When designing a class, we need to decide what are the relevant features, for our purposes, of the thing we're representing. That helps us figure out what data and functions the class should have. For example, if what we're concerned with is the selling of cars, then price would be one of the relevant features, but if we're only concerned with the driving of cars, then it wouldn't be.

"Private" class members

Some languages have a keyword ("private" in Java and C++) that means a certain class data member or method can't be accessed from outside the class. The methods that are part of the class can still access it, but not external code that uses the class. Python does not have such a keyword. Instead, it has the convention that any data member or method whose name begins with an underscore should be treated as private even though the language doesn't enforce it.

Here's an example of a bank account class that has private class variables:

class BankAccount:

"""

Represents a bank account that the user can deposit money to and withdraw money from.

"""

def \_\_init\_\_(self, account\_ID, balance):

"""Creates a bank account object with an account ID and balance."""

self.\_account\_ID = account\_ID

self.\_balance = balance

def get\_account\_ID(self):

"""Returns the account ID."""

return self.\_account\_ID

def set\_account\_ID(self, new\_ID):

"""Sets the account ID to a new value."""

self.\_account\_ID = new\_ID

def get\_balance(self):

"""Returns the current balance."""

return self.\_balance

def deposit(self, amount):

"""Deposits the specified amount into the account."""

self.\_balance += amount

def withdraw(self, amount):

"""Withdraws the specified amount from the account."""

self.\_balance -= amount

Notice that for class names, instead of separating words with underscores (which is called "snake case"), we start each new word with a capital letter (which is called "camel case"). Here's an example of using the BankAccount class:

> account\_1 = BankAccount("235349", 730.29)  
> print("account ID =", account\_1.get\_account\_ID())  
> account\_1.set\_account\_ID("983341")  
> print("account ID =", account\_1.get\_account\_ID())  
> print("balance =", account\_1.get\_balance())  
> account\_1.deposit(200.11)  
> print("balance =", account\_1.get\_balance())  
> account\_1.withdraw(500.00)  
> print("balance =", account\_1.get\_balance())

If we want for other code to be able to access a private data member, we need to provide get and/or set methods. A get method just returns the current value of the corresponding data member. A set method takes a parameter and just sets the corresponding data member to the value of the parameter. The normal naming convention is "get\_" or "set\_" followed by the name of the corresponding data member. For the account ID we provided both get and set methods. For the balance, we only provided a get method, since the balance is changed via the deposit and withdraw methods.

You might wonder why we would want to have private data members if it means writing more methods. The reason is that it allows us to control access to the data members. If we don't want a data member to be accessed, we can just not write get and/or set methods for it. Or if a data member should only be set to a certain range of values, then we can check for that in the set method. If a data member is public, then it can be changed to anything from anywhere, which can make your program logic harder to understand and debug. Controlling access to data members this way is referred to as **information hiding** (or data hiding).

Information hiding is important because it allows you to control how certain parts of the class are accessed. It allows you to separate interface (the names and expected parameters of any public functions) from implementation (the definitions of those functions, and also any private data members or private functions). Making that distinction makes it easier to modify the class later on. If you allow the user to directly access any part of the class, then some users' code will end up depending on specific details of the implementation, which means that if you change the implementation later, you'll break any code that depends on it. It makes code much more modular if you provide a defined interface. The specific implementation details can then be allowed to change as long as the interface remains the same.

In addition to private data members, we can also indicate that methods are private in the same way. The reason we would make a method private is if it's just meant for internal use by the class and not part of the way we expect users to interact with the class. An example of this would be if we had a Fraction class and we wanted the Fractions to always be in reduced form. To accomplish this, we could have a greatest\_common\_divisor() function that can be used within the class. However, we wouldn't normally expect users to interact with a Fraction object by asking it to find the gcd of two integers - we expect them to do more explicitly Fraction-y things, like printing the Fraction or multiplying it by another Fraction. Finding the gcd of two integers is an implementation detail that can be "hidden" from the users.

Everything is an object

In Python, everything is an object. Strings, integers, and all other values are objects. Functions are objects. Even the special value *None* is an object. Given the power and popularity of Python (and other object-oriented languages), it's clear that object-oriented design can be a valuable tool for organizing code in ways that help make programming easier. It can feel very abstract at first, but it's definitely worthwhile to master the concepts involved.

Printing objects of user-defined classes

If you try to print an object of a user-defined class, you'll get something like this:

> print(account\_1)  
> <\_\_main\_\_.BankAccount object at 0x103506d30>

which just tells you what class it belongs to and its address in memory. In order to print out the values of the data members, you need to specifically access those data members in the print statement, as shown earlier. In CS 162, you'll learn how to re-define the behavior of the print function for objects of user-defined classes.

Exercises

1. Define a class named *HourlyWorker* that has three private data members: \_*name*, \_*ID*, and \_*wage*. The class should have a docstring and an init method. The parameters to the init method should be in the order listed above.

Example of how someone might use your class:

facebook = HourlyWorker("Mark", 777777, .0004)  
girlswhocode = HourlyWorker("Reshma", 424242, 108.13)  
print(facebook.\_name)  
print(girlswhocode.\_wage)

Expected output:

'Mark'  
108.13

2. Define a class named *Box* that has three private data members: \_*length*, \_*width*, and \_*height*. The class should have a docstring and an init method. The parameters to the init method should be in the order listed above. The class should also have a method named *volume* that returns the volume of the Box, and a method named *surface\_area* that returns the surface area of the Box.

Example of how someone might use your class:

tesseract = Box(8, 8, 8)  
print(tesseract.volume())  
print(tesseract.surface\_area())

Expected output:

512  
384

Additional Reading:

1. [How to Debug for Absolute Beginners by MicrosoftLinks to an external site.](https://docs.microsoft.com/en-us/visualstudio/debugger/debugging-absolute-beginners?view=vs-2019&tabs=csharp)
2. [JetBrains Tutorial on How to debug in PyCharmLinks to an external site.](https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html)
3. [Using conditional breakpoints in PyCharmLinks to an external site.](https://www.jetbrains.com/pycharm/guide/tips/conditional-breakpoints/)