# Libraries (1A)

Young Won Lim
12/29/24

# Modules, Packages, and Libraries (1)

When you start working with Python, you'll often hear the terms module, package, and library. While they seem similar, these concepts represent different levels of organization within Python. Understanding the differences between them will help you navigate Python's resources more effectively.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules, Packages, and Libraries (2)

Key Differences in a Nutshell:

**Modules** are single Python files (.py)
that contain code, functions, and classes.

For example, calculator.py.

**Packages** are directories
that contain multiple related modules.

**Packages** are defined by the presence of
an __init__.py file.

For example, math_tools/.

**Libraries** are collections of packages and
modules designed to provide broader functionality.

For example, NumPy or Requests.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules, Packages, and Libraries (3)

Real-World Examples

**os Module**: A widely used Python module
that provides functions for interacting
with the operating system.

It’s a single file located in the standard library,
and you can import it as a module:

```
import os
print(os.getcwd())  # Prints the current working directory
```

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules, Packages, and Libraries (4)

2. **Django**: A web development framework, which is a library composed of multiple packages and modules.

The django package includes sub-packages like db, core, and views.

You can install Django using pip and import its modules to build web applications:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello, World!")
```

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules, Packages, and Libraries (5)

3. **Pandas**: A popular library
for data manipulation and analysis.
Although you use it as import pandas as pd,
it’s made up of multiple packages
and modules under the hood.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules (1)

A module in Python is a file containing Python definitions
and statements that organize code into manageable chunks.

They allow the reuse code across projects
without needing to rewrite it each time.

A module as a single file that contains
functions, variables, and classes.

You can import this file into another script
and use its functions.

For example, let’s say you create a module called calculator.py.
Inside this file, you define several functions:

```
# calculator.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a – b
```

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules (2)

You can then use this module
in another Python file like this:

```
# main.py
import calculator

print(calculator.add(5, 3))  # Outputs: 8
print(calculator.subtract(10, 4))  # Outputs: 6
```

Here, calculator.py is the module,
and main.py is importing and using its functions.

Modules can be as simple as this,
or they can contain hundreds of lines of code.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Packages (1)

A package in Python organizes related modules
into a directory hierarchy.

Packages allow you to structure code logically,
especially when a project grows large.

A package typically contains multiple modules,
and a special file called __init__.py.

This file tells Python that the directory
should be treated as a package.

For example, consider a package called math_tools,
which contains several modules
for different mathematical operations.
The folder structure might look like this:

```
math_tools/
    __init__.py
    addition.py
    subtraction.py
    multiplication.py
    division.py
```

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Packages (2-1)

The modules in the math_tools package
focuses on a specific operation.

The __init__.py file allows the import the entire package,
as well as its individual modules, in the code.

Here’s an example of how you might
use the math_tools package:

```
# main.py
from math_tools import addition, subtraction

print(addition.add(2, 3))  # Outputs: 5
print(subtraction.subtract(7, 2))  # Outputs: 5
```

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Packages (2-1)

In this case, math_tools is a package
that organizes related modules
for different mathematical operations.
Each module within the package handles a specific function,
such as addition or subtraction.

The use of packages makes it easier to manage large projects
and keep your code organized.

Rather than having one massive file with all your functions,
you can break them into logical pieces and group them in packages.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Libraries (1)

The term library in Python is often used
more loosely compared to modules and packages.

In simple terms, a library is a collection of packages
or modules that provide specific functionality.

It’s a broader concept than both modules and packages.

Libraries often include multiple packages,
but they can also be made up of a single package or module.

For example, the popular library NumPy
provides support for large, multi-dimensional arrays and matrices,
along with a large collection of mathematical functions
to operate on these arrays.

NumPy is a library, but within it are multiple packages
and modules that handle different parts
of array manipulation and numerical computing.

Another example is Requests, a popular library used
for making HTTP requests in Python.
Although Requests is considered a library,
it’s composed of multiple modules and packages
working together to simplify working with HTTP operations.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Libraries (2)

```
import requests

response = requests.get('https://api.github.com')
print(response.status_code)
```

Here, you import the requests library as a whole,
without needing to understand the internal modules or
packages that make it up.

The goal of a library is to provide a high-level interface
that abstracts away the complexity,
letting you use its features with minimal effort.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Libraries (2)

Here, you import the requests library as a whole, without needing to understand the internal modules or packages that make it up. The goal of a library is to provide a high-level interface that abstracts away the complexity, letting you use its features with minimal effort.
Key Differences in a Nutshell:

Modules are single Python files (.py) that contain code, functions, and classes. For example, calculator.py.
Packages are directories that contain multiple related modules. Packages are defined by the presence of an __init__.py file. For example, math_tools/.
Libraries are collections of packages and modules designed to provide broader functionality. For example, NumPy or Requests.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

# Real World Examples (1)

os Module: A widely used Python module that provides functions for interacting with the operating system. It’s a single file located in the standard library, and you can import it as a module:

```
import os
print(os.getcwd())  # Prints the current working directory
```

2. Django: A web development framework, which is a library composed of multiple packages and modules. The django package includes sub-packages like db, core, and views. You can install Django using pip and import its modules to build web applications:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello, World!")
```

3. Pandas: A popular library for data manipulation and analysis. Although you use it as import pandas as pd, it’s made up of multiple packages and modules under the hood.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Real World Examples (2)

os Module: A widely used Python module that provides functions for interacting with the operating system. It’s a single file located in the standard library, and you can import it as a module:

```
import os
print(os.getcwd())  # Prints the current working directory
```

2. Django: A web development framework, which is a library composed of multiple packages and modules. The django package includes sub-packages like db, core, and views. You can install Django using pip and import its modules to build web applications:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello, World!")
```

3. Pandas: A popular library for data manipulation and analysis. Although you use it as import pandas as pd, it’s made up of multiple packages and modules under the hood.

https://medium.com/@ccpythonprogramming/the-difference-between-python-libraries-modules-and-packages-07066dca8afc

## Modules and Libraries (1-1)

An API is not a collection of code per se -
it is more like a "protocol" specification
how various parts (usually libraries) communicate
with each other.

There are a few notable "standard" APIs in python.
E.g. the DB API

In my opinion, a library is anything that is not an application
- in python, a library is a module - usually with submodules.

The scope of a library is quite variable -
for example the python standard library is
vast (with quite a few submodules)
while there are lots of single purpose libraries in the PyPi,
e.g. a backport of collections.OrderedDict for py < 2.7

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

## Modules and Libraries (1-1)

A package is a collection of python modules
under a common namespace.

In practice one is created by placing multiple python modules
in a directory with a special __init__.py module (file).

A module is a single file of python code
that is meant to be imported.

This is a bit of a simplification since in practice
quite a few modules detect
when they are run as script and do something special in that case.

A script is a single file of python code
that is meant to be executed as the 'main' program.

If you have a set of code that spans multiple files,
you probably have an application instead of script.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

## Modules and Libraries (2)

Library : It is a collection of modules.

Library either contains
built in modules(written in C)
+ modules written in python

Module : Each of a set of standardized parts or independent units
that can be used to construct a more complex structure.

Speaking in informal language, A module is
set of lines of code which are used for a specific purpose
and can be used in other programs as it is ,
to avoid DRY(Don’t Repeat Yourself) as a team
and focusing on main requirement. source

API is an interface for other applications to interact
with your library without having direct access.

Package is basically a directory with files.

Script means series of commands within a single file.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

# Modules and Libraries (3)

Anyways context matters, greatly.

Library- Most often will refer to the general library
or another collection created with a similar format and use.

The General Library is the sum of 'standard',
popular and widely used Modules,
which can be thought of as single file tools,
for now or short cuts making things possible or faster.

The general library is an option
most people enable when installing Python.

Because it has this name "Python General Library"
it is used often with similar structure, and ideas.
Which is simply to have a bunch of Modules,
maybe even packages grouped together, usually in a list.

The list is usually to download them.

Generally it is just related files, with similar interests.
That is the easiest way to describe it.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

## Modules and Libraries (3)

Module- A Module refers to a file.

The file has script 'in it' and the name of the file
is the name of the module, Python files end with .py.

All the file contains is code that ran together
makes something happen, by using functions, strings ect.

Main modules you probably see most often are popular
because they are special modules that can get info
from other files/modules.

It is confusing because the name of the file and
module are equal and just drop the .py.

Really it's just code you can use as a shortcut written
by somebody to make something easier or possible.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

## Modules and Libraries (4)

Package- This is a term is used to generally sometimes, although context makes a difference.

The most common use from my experience is
multiple modules (or files) that are grouped together.

Why they are grouped together can be for a few reasons,
that is when context matters.

These are ways I have noticed the term package(s) used.

They are a group of Downloaded, created and/or stored modules.
Which can all be true, or only 1,
but really it is just a file that references other files,
that need to be in the correct structure or format,
and that entire sum is the package itself,
installed or may have been included
in the python general library.

A package can contain modules(.py files)
because they depend on each other
and sometimes may not work correctly, or at all.

There is always a common goal of every part (module/file)
of a package, and the total sum of all of the parts is the package itself.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

# Modules and Libraries (5)

Most often in Python Packages are Modules,
because the package name is the name of the module
that is used to connect all the pieces.

So you can input a package because it is a module,
also allows it to call upon other modules,
that are not packages because they only perform
a certain function, or task don't involve other files.
Packages have a goal, and each module works together
to achieve that final goal.

Most confusion come from a simple file file name
or prefix to a file, used as the module name
then again the package name.

Remember Modules and Packages can be installed.
Library is usually a generic term for listing,
or formatting a group of modules and packages.
Much like Pythons general library.
A hierarchy would not work,
APIs do not belong really, and
if you did they could be anywhere and
every ware involving Script, Module, and Packages.
the worl library being such a general word,
easily applied to many things, also makes API able
to sit above or below that.
Some Modules can be based off of other code,
and that is the only time I think it would relate

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

## The Python Package Index and third-party packages (1)

The Python Package Index (PyPI) is
a repository of third-party Python packages
that extend the functionality of the Python Standard Library.

These packages cover a wide range of domains
and provide solutions to various programming challenges.

These packages are created by the open-source community.

We can also create our own package
and publish it with the repository.

To manage third-party packages,
Python uses a tool called pip (Python Package Installer).

pip allows us to easily install, upgrade,
and manage packages from PyPI.

https://www.sitepoint.com/python-modules-packages/

## The Python Package Index and third-party packages (2)

We can install any third-party library using pip:

```
pip install package_name
```

For instance, to install the Django package
(which is used for web development) we can run this:

```
pip install django
```

https://www.sitepoint.com/python-modules-packages/

## The Python Package Index and third-party packages (2)

Here are examples of some popular third-party packages:

**NumPy**: a powerful library for numerical computing in Python.
It provides support for large, multi-dimensional arrays and matrices,
along with a variety of mathematical functions to operate on these arrays.

**Pandas**: a library for data manipulation and analysis.
It provides data structures like DataFrames
for efficiently handling and analyzing tabular data.

**Matplotlib**: a widely-used library for creating
static, animated, and interactive visualizations in Python.
It offers a MATLAB-like interface
for plotting various types of graphs and charts.

**SciPy**: built on top of **NumPy**,
**SciPy** provides additional functions
for optimization, integration, linear algebra,
signal processing, and more.

https://www.sitepoint.com/python-modules-packages/

## The Python Package Index and third-party packages (3)

**Django**: a high-level web framework
for building web applications.
It follows the Model-View-Controller (MVC) architecture
and offers features for handling databases,
URLs, templates, and more.

**Flask**: another web framework,
**Flask** is more lightweight and minimal compared to **Django**.
It’s ideal for building smaller web applications or APIs.

**Requests**: a package for making HTTP requests
and handling responses.
It simplifies working with web APIs
and fetching data from the Internet.

The packages listed above are just a few examples
of the vast ecosystem of third-party packages available on PyPI.
Packages like these can save us a lot of time and effort.

https://www.sitepoint.com/python-modules-packages/

## Modules (1)

The module is a simple Python file that contains
collections of functions and global variables
and with having a .py extension file.

It is an executable file and to organize
all the modules we have the concept
called Package in Python.

Examples of modules:

- Datetime
- Regex
- Random etc.

```
def myModule(name):
        print("This is My Module : "+ name)
```

Import module named demo_module
and call the myModule function inside it.

```
import demo_module

demo_module.myModule("Math")
```

https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/

```
This is My Module : Math
```

## Packages (1)

The package is a simple directory
having collections of modules.

This directory contains Python modules
and also having __init__.py file
by which the interpreter interprets it as a Package.

The package is simply a namespace.

The package also contains sub-packages inside it.

Examples of Packages:

Numpy
Pandas

```
Student(Package)
| __init__.py (Constructor)
| details.py (Module)
| marks.py (Module)
| collegeDetails.py (Module)
```

https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/

## Libraries (1)

The library is having a collection of
related functionality of codes that allows you
to perform many tasks without writing your code.

It is a reusable chunk of code that we can use
by importing it into our program,
we can just use it by importing that library
and calling the method of that library with a period(.).

However, it is often assumed that
while a package is a collection of modules,
a library is a collection of packages.

Examples of Libraries:

- Matplotlib
- Pytorch
- Pygame
- Seaborn etc.

Example:

https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/

Importing pandas library and call read_csv method
using an alias of pandas i.e. pd.

```
import pandas as pd

df = pd.read_csv("file_name.csv")
```

## Install Python modules (1)

A module is simply a file containing Python code. Functions, groups, and variables can all be described in a module. Runnable code can also be used in a module.
What is a Python Module?

A module can be imported by multiple programs for their application, hence a single code can be used by multiple programs to get done with their functionalities faster and reliably. In this article, we will see how to install modules in Python.

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (2)

Prerequisites

The pip package manager is a preferred installer program to install modules in Python. We have also used pip to install a module on the computer.

If the pip is not installed then refer to the articles:

How to install PIP in Linux?
How to install pip in macOS?
How to install pip in Windows?

If a module is not compatible with pip, we have also shown the manual way of installing the module in Python.

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (3)

Install a Python Module with pip

Below are some of the steps by which we can follow to install a Python module with pip in Windows:
Step 1: Open the Command Prompt

Open the command prompt (Windows) or terminal (Mac or Linux) on your computer.
Step 2: Installing Python Modules with Pip

Use the following command to install a module via pip, which is the package installer for Python:

```
pip install <module name>
```

Replace <module name> with the name of the module you want to install. For example, to install the popular numpy module, you would use:

```
pip install numpy
```

Note: If you are using Python 3.x, you may need to use the command pip3 instead of pip.

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (4)

Once you run the command, pip will download the module from the Python Package Index (PyPI) and install it on your computer.
Step 3: Verifying Module Installation

You can now use the module in your Python code by importing it at the beginning of your script. For example, if you installed the numpy module, you would add the following line at the top of your script:

```
import numpy
```

Note: Some modules may have additional dependencies that need to be installed first. In this case, pip will automatically download and install the necessary dependencies.

That’s it! You have successfully installed a Python module using pip. You can repeat these steps to install modules in Python.

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (5)

More Operations on Python Modules
Upgrade Python Modules

To upgrade the modules that are already installed, use the following command:

```
pip install --upgrade <modulename>
```

Uninstall Python Modules

To uninstall a modules that is already installed, use the following command:

```
pip uninstall <modulename>
```

Install Python Modules from Other resource

To install the modules from the other resources, use the following command :

```
pip install -e git+<https://github.com/myrepo.git#egg=modulename>
```

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (6)

Installing Python Modules on Unix/macOS

Make sure that you have pip installed. Type the below command in your terminal to verify if the pip is installed or not.

```
python3 -m pip --version
```

Type the below command to install the module using pip.

```
python3 -m pip install "ModuleName"
```

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (7)

More Operation on Modules in Unix/macOS
Install Specific Version of Module

To install the specific version of the module use the following command:

```
python3 -m pip install "ModuleName==2.2"
```

Install Module Version Between Two Numbers

To install the version of a module in between any two numbers:

```
python3 -m pip install "ProjectName>=2,<3"
```

Install Specific Compatible Version

To install a specific compatible version of full length:

```
python3 -m pip install "ModuleName~=2.2.3"
```

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (8)

Upgrade Module

To upgrade the project version use the following command:

```
python3 -m pip install --upgrade ModuleName
```

Install Required Module from Text Document

To install a required module that is in the text document:

```
python3 -m pip install -r requirements.txt
```

Install Directories Present in Local System

To install the directories that are present in the local system using the following command:

```
python3 -m pip install --no-index --find-links=file:///local/dir/ ProjectName
python3 -m pip install --no-index --find-links=/local/dir/ ProjectName
python3 -m pip install --no-index --find-links=relative/dir/ProjectName
```

Update a Package

To update the installed pip and setup tools copies use the following command:

```
python -m pip install --upgrade pip setuptools wheel
```

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Install Python modules (9)

Manual Installation of Python Modules

The majority of Python modules are now designed to work with pip. If you have a kit that isn't compatible, you'll have to install it manually.

Install the kit by downloading it and then extracting it to a local directory. If the kit has its own set of installation instructions, obey them, if the module is not present there then use the following command to install the module using the command manually:

```
python <FILE_NAME>.py install
```

So we have covered how to install a module in Python. We have show the methods to install modules using pip package installer and manually using .py install method.

https://www.geeksforgeeks.org/how-to-install-a-python-module/

## Downloading and installing Packages (1)

What is pip?

Pip is a package management system used to install and manage Python packages. It is included in the installation of Anaconda.
The use of pip:

To install a package: pip install package

To uninstall a package: pip uninstall package

To upgrade a package: pip install --upgrade package OR pip install -U package

To search a package: pip search "package"

To list all packages installed: pip list

To get help for using pip: pip help

More pip methods can be found pip Docs.

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/beginners/downloading-and-installing-packages.html

## Downloading and installing Packages (2)

How to install libraries / packages / modules?

First open Spyder and click Tools --> Open command prompt.

You should see the Command Window appear in the bottom right of your screen.

Here we install the Python package seaborn as an example.

```
# In the command line, type pip install seaborn

C:\Users\your_username\Documents\Python Scripts>pip install seaborn
```

This will install seaborn on your machine.

Note:

To upgrade the pip version on Windows, type python -m pip install --upgrade pip on the command line.
On Windows, all of your Python packages can be found in the directory of C:\Anaconda2\Lib\site-packages if you use the default path when you install Anaconda.
To upgrade the pip version on OS X, type pip install --upgrade pip on the command line.

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/beginners/downloading-and-installing-packages.html

# Downloading and installing Packages (3)

Modules and Packages

A module is simply a python file full of functions, classes, and variables; like all python files, it has the .py extension. We can use modules to group related codes. The dir() function can be used to see which functions are implemented in a module and help() can be used to see more detail about the function we want to use. Here is an example:

```
import math
dir(math)
Output: ['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh','atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees',
 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

A package is a directory that consists of multiple modules and sub-packages.

dir() and help() can be used to explore packages as well.

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/beginners/downloading-and-installing-packages.html

## Downloading and installing Packages (4)

Import modules or packages

Python provides several ways to import modules; we introduce three ways below.

import math imports the module math. After we run this statement, we can use math.name to refer to functions defined in module math. For example, math.sin can be used to evaluate the value of trigonometric function sine and math.sin(math.pi/2) = 1.

from math import * imports the module math. After we run this statement, we can simply use functions defined in module math without specifying the module they came from. For example, sin(pi/2) will evaluate sine of pi/2.

from math import sin, cos imports the module math. After we run this statement, we can use sine and cosine without specifying the module, but not other functions in this module. For example, sin(math.pi/2) will evaluate sine of pi/2.

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/beginners/downloading-and-installing-packages.html

## Downloading and installing Packages (5)

Which way is best?

We use modules in different ways depending on our needs. Sometimes, it is better to use import module, especially if we are using two modules that have a function of the same name - using this method will allow us to distinguish between the two. For example, numpy.sin can evaluate values of sine of an array but math.sin can only be used to evaluate the sine of a single value. An alternate example is the pi function: you can use math.pi, numpy.pi or scipy.pi and they all give the same value for pi. Because all three modules give pi the same value, we can choose one and have easy access to pi without having to import or specify another module.

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/beginners/downloading-and-installing-packages.html

## Installing libraries and packages (1)

Python packages are a set of python modules, while python libraries are a group of python functions aimed to carry out special tasks. There are over 137,000 python libraries and over 235,000 python packages. These libraries and packages can ease a developer’s experience and avoid the need to re-invent the wheel, as the saying goes. Henceforth, the word package(s) will be used when referring to both packages and libraries. While there are distinctions within python, the words packages and libraries are frequently interchanged, and the process for installing them is generally the same.

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (2)

There are several ways that Python packages can be installed and managed. ARC support staff can install packages that can be made available to anyone who loads the appropriate module, while individuals can install packages for their own use. There are two main routes for users to install packages. One is to install directly into the user’s personal python library using the pip command. In that case, all packages (for a given version of python) are stored in the same directory. However, this approach can result in conflicts with package version requirements.

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (3)

Sometimes one application needs a particular version of a package but a different application needs another version. Since the requirements conflict, installing either version will leave one application unable to run. This situation can be resolved by using virtual environments. A virtual environment is a semi-isolated Python environment that allows packages to be installed for use by a particular application or for a particular project.

Common tools used for Python package installation and environment management include:

pip – the Python package installer, can be used on its own or within a virtual environment
venv – an environment manager within which pip is used to install packages
virtualenv – an environment manager within which pip is used to install packages
conda – an environment manager and a package installer, it does not rely on pip to install packages

You should pick one method and use it exclusively to avoid mixing types of installations.

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (4)

Python packages are installed for the specific version of Python that is in use during installation. If you switch from using a module for one version of Python to a different one with where either the major or minor version changes, then you will have to re-install any packages/libraries in order to make them available in the library of the new version of Python. You only need to install Python packages once for each cluster on which you wish to use the library and, separately, for each version of Python that you use. Please note, Python packages should be installed using the command line from a login node, not from within Jupyter Notebook or the JupyterLab app.

A brief description of how to use each tool to install packages and manage virtual environments, along with a description of where you can expect to find installed packages, is provided below.

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (5)

Pip¶

To install a Python package into your personal library using pip, enter the following command, replacing with the actual package name:

```
$ pip install --user <package_name>
```

The --user tag will, by default, place packages in

```
$HOME/.local/lib/python?.?/site-packages
```

where ?.? indicates the versioning of the Python release. The library will then be available to you for this and future sessions.

You can install a specific version of a package by giving the package name followed by == and the version number. For example:

```
pip install --user tensorflow==2.5.0
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

# Installing libraries and packages (6)

Venv¶

venv is the standard Python tool for creating virtual environments, and has been part of Python since version 3.3. Starting with Python 3.4, it defaults to installing pip into all created virtual environments. Installing packages into an active venv is done via the pip command, as described above.

Virtual environments are created as follows:

```
$ python -m venv /path/to/new/virtual/environment
```

Alternatively, you can change into the directory of the project you are working on and simply provide a name for the virtual environment in place of the full path:

```
$ cd /path/to/my/project
$ python -m venv myenv
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (7)

To activate a virtual environment, enter:

```
$ source myenv/bin/activate
```

If you are not in your project directory, then you must provide the full path to the virtual environment you specified when creating the virtual environment:

```
$ source /path/to/my/project/myenv/bin/activate
```

When you are in an active virtual environment, you will see the name of the environment in the prompt. The PATH environment variable is updated so that the virtual environment’s bin directory is at the beginning:

```
(myenv) $ which pip python
~/my_project/myvenv/bin/pip
~/my_project/myvenv/bin/python
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (8)

At this point, you would use the pip command to install any needed packages. Packages that you install using pip while in a virtual environment will be placed in the myenv folder, isolated from the global Python installation, and only available to you from within the virtual environment.

You can deactivate a virtual environment by typing deactivate in your terminal.

https://docs.support.arc.umich.edu/python/pkgs_envs/

# Installing libraries and packages (9)

Virtualenv¶

virtualenv is a third party alternative (and predecessor) to venv. It comes installed with the Developer Python modules. However, it is not installed for the Anaconda modules. If you would like to work within a virtual environment using virtualenv with one of the Anaconda modules, you will need to install it.

Install virtualenv with pip:

```
pip install --user virtualenv
```

Create a virtual environment for a project:

```
$ cd project_folder
$ virtualenv <env_name>
```

Similar to the way venv works with Python 3, virtualenv myenv will create a folder in the current directory which will contain the Python executable files and a copy of the pip library which you can use to install other packages. To begin using the virtual environment, it has to be activated:

```
$ source myenv/bin/activate
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

# Installing libraries and packages (10)

The name of the current virtual environment will now appear in parenthesis to the left of the prompt to let you know that it's active.

When done working in the virtual environment, simply deactivate it:

```
(myvenv) $ deactivate
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (11)

Conda¶

Conda is both a package installer, like pip, and an environment manager, like venv and virtualenv. While pip, venv, and virtualenv are for Python, conda is language agnostic and works with other languages as well as Python.

To create a virtual environment for Python with conda, enter the following:

```
$ conda create --name conda-env python
```

where conda-env can be replaced with whatever name you choose for your virtual environment. Also, -n can be used in place of --name. This environment will use the same version of Python as your current shell's Python interpreter. To specify a different version of Python, specify the version number when creating your virtual environment as follows:

```
$ conda create -n conda-env python=3.7
```

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Installing libraries and packages (12)

You can install additional packages when creating an environment, by specifying them after the environment name. You can also specify which versions of packages you'd like to install.

```
$ conda create -n conda-env python=3.7 numpy=1.16.1 requests=2.19.1
```

It's recommended to install all packages that you want to include in an environment at the same time in order to avoid dependency conflicts.

You can then activate your conda environment as follows:

```
$ conda activate conda-env
(conda-env) $
```

As when using other virtual environment programs, the name of the current virtual environment will now appear in parenthesis to the left of the prompt to let you know that it's active. When you are finished working in the environment, simply enter $ conda deactivate and your normal prompt will return.

Virtual environments created with conda reside, by default, in the envs directory found in the following path: /home/$USER/.conda/envs

https://docs.support.arc.umich.edu/python/pkgs_envs/

## Package **sound6** (3)

https://python-course.eu/python-tutorial/packages.php