

COMP(2041|9044) 22T2 — Python Modules

<https://www.cse.unsw.edu.au/~cs2041/22T2/>

Standard Modules

Python has *a lot* of standard library modules

There are 217 standard modules

Any of them can be used via the import statement.

We have already used:

```
sys  
os  
re  
subprocess  
# etc
```

Non-Standard Modules

As a rule of thumb, someone else has already solved every problem.

It's a good idea to look for existing code.

There are about 390,000 packages available on PyPI.

PyPI is the [Py]thon [P]ackage [I]ndex

PyPI is a website that allows you to search for and register your own packages.

Any packages listed on the index can be installed via pip.

A package is a collection of files.

These files contains the source code of, and installation instructions for, one (or more) modules.

The most common format for python packages is called a wheel.

A wheel is basically just a .zip file that contains files in a specially crafted format.

Most of the time you don't need to worry about wheels (or other package types) as they are automatically downloaded and installed.

pip is the standard package manager for Python.

pip stands for [P]ip [I]nstalls [P]ackages.

pip can install any package on PyPI.

Or other repositories that have been configured.

To install a package, you can use the following command:

```
$ python3 -m pip install <package_name>
```

You can also install a package from a local directory or git:

```
$ python3 -m pip install <package>.whl
```

```
$ python3 -m pip install git+<package_url>
```

pip also updates and uninstalls package.

```
$ python3 -m pip install --upgrade <package_name>
```

```
$ python3 -m pip uninstall <package_name>
```

By default, Python installs package in the global (per user) Python environment.

This is usually very messy.

It is preferable to keep your package within the project that is using them.

This can be done by creating a virtual environment.

In python a virtual environment is a directory that contains a copy of the Python interpreter.

It can be using to isolate your project from the rest of the system.

To create a virtual environment, use the following command:

```
$ python3 -m venv <venv_name>
```

It's common to simply name your virtual environment venv.

Once you have a virtual environment, you can activate it by running the following command:

```
$ . <venv_name>/bin/activate
```

OR on Windows:

```
$ <venv_name>/Scripts/activate
```

without activating a virtual environment you are still in the global environment.

Once activated, the python, python3, pip, etc commands will be run from within the virtual environment.

Python packages should be versioned using PEP440.

The full syntax for a PEP440 version is:

```
[N!]N(.N)*[{a|b|rc}N][.postN][.devN]
```

most commonly only the $N(.N)^*$ part is used.

This defines a version of format $X.Y.Z$

Eg:

```
1.0.0  
2.0  
3.9.2  
4.2  
7.4.67.3.32
```

This is called a final release

It is most common to use three numbers `major.minor.micro`

Where:

the `major` version is incremented when there is a forward incompatible change.

the `minor` version is incremented when there is a backward incompatible change.

the `micro` version is incremented when there is a non-breaking change (eg bug fix).

If any number isn't specified, it is assumed to be 0.

Eg, all the following are the same:

```
5.7
5.7.0
5.7.0.0
5.7.0.0.0
# etc
```

version specifiers determine which version of a package to use.
without a version specifier, any version (usually the latest) is used.

An exact version is specified by using the `==` operator.

A minimum version is specified by using the `>=` operator (or exclusively `>`).

A maximum version is specified by using the `<=` operator (or exclusively `<`).

A excluded version is specified by using the `!=` operator.

A strict version is specified by using the `===` operator.

A compatible version is specified by using the `~=` operator.

version specifiers == vs ===

== is used to specify an exact version.

=== is used to specify a strict version.

=== is essentially a string comparison.

where as == takes into account semantic information.

```
1.0 == 1.0.0 # True  
1.0 === 1.0.0 # False
```

version specifiers `~=`

`~=` is used to specify a compatible version.

a compatible version of `X.Y` is `>= X.Y, == X.*`

That is: the minor version is greater than or equal, and the major version is the same.

version specifiers

multiple version specifiers can be used to restrict the version of a package.

```
~= 3.1.0, < 3.1.7, != 3.1.3
```

```
3.1.0
```

```
3.1.1
```

```
3.1.2
```

```
3.1.4
```

```
3.1.5
```

```
3.1.6
```

version specifiers

version specifiers can be used with `pip`.

```
$ python3 -m pip install 'regex~=2022.7.0,>2022.7.23,!=2022.7.24'
```

by default, `pip` will install the latest version of a package.

It can be very annoying to keep track of all the packages you need to install.

So instead, we can put them in a file, conventionally called `requirements.txt`.

The `requirements.txt` file is a simple text file that contains a list of package to install.

These can either not have a version specifier.

le. just a list of package.

Or they can have a version specifiers.

le. when you want to replicate an environment.

```
requests  
beautifulsoup4  
regex
```

```
requests ~= 2.0.0, <= 2.25, != 2.26.0, != 2.27.1  
beautifulsoup4 >= 5.4, < 5.10  
regex ~= 2022.7.0, > 2022.7.23, != 2022.7.24
```

pip can install package from a `requirements.txt` file directly.

```
$ pip install -r requirements.txt
```

pip can generate a `requirements.txt` file with version specifiers.

```
$ pip freeze > requirements.txt
```

`pip freeze` gives you a list of *all* packages and their versions.
Even those that were not directly installed (indirect requirements).
This can clutter up your `requirements.txt` file.
So `pip` also supports a `constraints` file.

constraints.txt

constraints.txt works exactly like requirements.txt

except that a package in constraints.txt will only be installed if they are also in requirements.txt.

```
$ pip install -r requirements.txt -c constraints.txt
```

```
# requirements.txt
```

```
requests
```

```
beautifulsoup4
```

```
regex
```

```
# constraints.txt
```

```
beautifulsoup4==4.11.1
```

```
certifi==2022.6.15
```

```
charset-normalizer==2.1.0
```

```
idna==3.3
```

```
regex==2022.7.25
```

```
requests==2.28.1
```

```
soupsieve==2.3.2.post1
```

```
urllib3==1.26.11
```