Learning Objectives: Conda

Learners will be able to...

- **Define and install Conda**
- **Define the base virtual environment**
- **Activate and deactivate a virtual environment**
- **Create a virtual environment with a specified name**
- **Add and remove packages with Conda and `pip`**
- **Create a requirements file that includes all packages**
- **Update packages in a virtual environment**
- **Build a package for distribution**

info

Make Sure You Know

You are familiar with `pip` and virtual environments. You can work with command line utilities like `grep`.

Limitations

The material only talks about building a package in the most basic way. No hands-on work is done for building a package for `conda-forge`; it is only discussed in a high-level manner.

What is Conda?

The Conda Package Manager

important

Conda Installation

There were some problems having learners manually install Conda. Instead, Conda has already been installed for you. The instructions on this page are to inform you how to install Conda on your own machine. Start interacting with the terminal in the next section entitled **"The Base Environment"**.

Anaconda is an all-encompassing product aimed at the data science, machine learning, and artificial intelligence communities. They maintain their own repository of popular packages, provide graphical tools for package management (both paid and free), as well as a command line interface. We will be focusing on the command line tool called Conda. Anaconda (and Conda) work not only with Python, but a wide variety of languages like R, Ruby, Java, JavaScript, C++, etc.

We will start by downloading Conda. The `wget` command downloads the content from the given web address. The URL is a link for the installation script, so the command will download the installer script.

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-
        x86_64.sh
```

▼ **Did you notice?**
Conda works with Python, but we did not use `pip` to install it. There is a Conda package in PyPI, but Anaconda recommends you use their installation script.

Next, start the installation script by running the newly downloaded file.

```
bash Anaconda3-2021.11-Linux-x86_64.sh
```

The installation of Conda is a bit different. For starters, you are asked to look over the license. Pressing `ENTER` will slowly advance the license. Once you get to the end, type `yes` to confirm your intent to install Conda. Conda will ask about the installation location and initialization. Press `ENTER` or `RETURN` to use the default location. Then the installation will begin. the could take some time. Type `yes` when asked about initialization.

You will know that Conda is properly installed because `(base)` appears at the beginning of prompt in the terminal. **Note**, the text after `codio@` will be different for you.

```
(base) codio@statichuman-typebeatles:~/workspace$
```

challenge

Try these variations:

- Check the version of Conda that you installed.

```
conda --version
```

- Display information about your current Conda installation.

```
conda info
```

The Base Environment

Once Conda is installed, you will see `(base)` at the prompt. This is the base virtual environment for Conda, and it starts automatically. The base environment is a specialized environment created for the needs of Anaconda (and Conda) users. This environment is vastly different from the standard Python installation on your system. Start by taking a look at the list of pre-installed packages in the base environment.

```
conda list
```

There are almost 400 packages installed in the base environment. While the base virtual environment is separate from your standard Python installation, you should not use this for development purposes. The idea

behind virtual environments is that each project has its own area for packages. Keeping projects separate reduces the risk of conflicting packages breaking other projects.

challenge

Try these variations:

- Deactivate the base virtual environment. `(base)` should no longer be present in the terminal prompt.

```
conda deactivate
```

- Activate the base virtual environment. `(base)` should once again be present in the terminal prompt.

```
conda activate
```

Virtual Environments

Creating a Virtual Environment

Aside from the base environment, Conda does not automatically create or activate any virtual environments. Just like using `pip`, you should create a separate virtual environment for each project.

Create the Conda environment called `learning-conda` by using the `create` command and the `--name` flag. Conda will ask you to type `yes` to confirm your environment creation.

```
conda create --name learning-conda
```

Notice, Conda does not immediately activate the new environment. You need to do this manually. Use the `activate` command followed by the name of virtual environment you want to activate.

```
conda activate learning-conda
```

You should now see the name of the virtual environment appear at the beginning of your prompt.

```
(learning-conda) codio@statichuman-typebeatles:~/workspace$
```

▼ **Did you notice?**

The `learning-conda` virtual environment does not,in contrast to the `base` environment, have any packages installed by default.

```
conda list
```

challenge

Try these variations:

- Create a copy of the `learning-conda` virtual environment.

```
conda create --clone learning-conda --name lc2
```

- List all of the virtual environments on your system.

```
conda env list
```

▼ Did you notice?
When Conda lists out all of the virtual environments, it places a `*` next to one of them. This indicates the virtual environment you are currently using.

Deactivating and Removing Virtual Environments

Unlike Poetry, you have to manually manage your virtual environments in Conda. That means if you are currently using one virtual environment but need to use another, then you have to deactivate the first virtual environment and then active the other one.

If `conda activate` activates a virtual environment, then `conda deactivate` should deactivate it. You will now see `(base)` at the beginning of the prompt after deactivating a specific virtual environment.

```
conda deactivate
```

Up above, we made a copy of the `learning-conda` virtual environment. We don't really need this environment. It was only used to introduce how to clone an existing environment. So let's remove the `lc2` environment.

The command is `env remove`, but we need to specify the name of the environment to remove. Use the `--name` flag followed by `lc2`.

```
conda env remove --name lc2
```

▼ **Did you notice?**
The command used to remove an environment is `env remove`. Using just `remove` tells Conda to remove a package.

You should no longer see `lc2` in the list of virtual environments on your system. It is also important to note that Conda removes all contents (packages and their dependencies) when deleting a virtual environment.

```
conda env list
```

Adding Packages

Add a Package

It probably does not come as a surprise that the `install` command is used to install a package in Conda. Activate the `learning-conda` virtual environment and then install the `requests` package. Conda will ask you to confirm your desire to install a package. Type `y` or press `ENTER`.

```
conda activate learning-conda
conda install requests
```

Using the list command, you see a list of all the packages (`requests`) and their dependencies.

```
conda list
```

We can install multiple packages at once. The command below will install `matplotlib` and `scipy`.

```
conda install -y matplotlib scipy
```

▼ **Did you notice?**
Conda did not ask you to confirm your desire to install these packages. That is because of the `-y` flag. This is called silently installing a package since you are no longer bothered with confirmation.

challenge

Try these variations:

- Create a new virtual environment and install a package at the same time. The command below creates the `lc3` environment and installs the `requests` package.

```
conda create --name lc3 requests
```

- You should be in the `learning-conda` virtual environment. Use the command below to install the `scipy` package to the `lc3` environment.

```
conda install --name lc3 scipy
```

▼ **Did you notice?**
The `-name` flag is used to denote the virtual environment, not the package.

Limitations of the Conda Package Repository

Remember, Conda is more than a package management tool. It is also a repository of packages aimed at the data science, machine learning, and AI audience. You can find their repository here. Anaconda maintains an extensive repository where you can find most of the popular packages. However, it is not identical to PyPI.

If you want to check the Anaconda repository for a package, use the `search` command. The `python-twitter` package is a wrapper for the Twitter API. Search Anaconda for this package.

```
conda search python-twitter
```

Conda should say that it cannot find the package. Yet it exists on PyPI. This isn't a problem, however. You can use `pip` to install packages for a Conda project. First, install `pip` in your virtual environment.

```
conda install pip
```

Then install `python-twitter` using `pip`.

```
python -m pip install python-twitter
```

▼ **Did you notice?**
Installing with `pip` did not use `python3 -m pip`. This is because there is only one version of Python installed (version 3.9 at the time of writing) in the virtual environment. You do not need to specify the version number when there is only one version of Python installed.

You should now be able to see `python-twitter` when listing all of the packages in the environment.

```
conda list
```

Removing Packages

Removing a Package

As the title implies, the `remove` command is used to remove a package from a virtual environment. Let's activate our `learning-conda` virtual environment.

```
conda activate learning-conda
```

Instead of listing all of the installed packages in the environment, let's count them. The `wc` command counts words. Since the `list`command prints columns for name, version, and build channel, we do not want to count words. Instead we want to count lines, which can be done with the `-l`flag. Call the `list` command and pipe (`|`) its output to `wc` so it can count the lines. At the time of writing, there were 109 lines in total. Your numbers maybe different if dependencies changed with new version numbers.

```
conda list | wc -l
```

Now remove the `requests` library from the virtual environment.

```
conda remove requests
```

Count the lines one more time. You should now see 99 lines in total. Your number may be different.

```
conda list | wc -l
```

Using the `remove` command not only removes `requests` but it also removes the dependencies as well.

challenge

Try these variations:

- Remove the `requests` package from the `remove-example` virtual environment. You do not need to change environments to do this. Use the `--name` flag to specify the environment from which the package should be removed.

```
conda remove --name remove-example requests
```

We can verify the remove by listing the packages from the `remove-example` environment. The `requests` package is no longer listed.

```
conda list --name remove-example
```

- Remove the `python-twitter` package from the `remove-example` environment.

```
conda remove python-twitter --name remove-example
```

▼ **Why is there an error?**

You should see the following error:

```
PackagesNotFoundError: The following packages are missing
from the target environment:
  - python-twitter
```

Remember, the `python-twitter` package is not found in the Anaconda repository. Since it had to be installed with `pip`, you must uninstall it with the same tool. Since you are using `pip` you need to deactivate your current environment and activate `remove-example` before removing the package.

```
conda deactivate
conda activate remove-example
pip uinstall python-twitter
```

Removing Multiple Packages

You can also remove more than one package at a time. You will still use the `remove` command but list out the packages you want to remove by separating them with a space.

```
conda remove matplotlib scipy
```

challenge

Try this variation:

- Remove all of the packages from an environment. The `--all` flag tells Conda to remove all of the packages. Use the `--name` flag to specify `remove-example` environment.

```
conda remove --all --name remove-example
```

Now list all of the installed packages for the `remove-example` virtual environment.

```
conda list --name remove-example
```

▼ **Why is there an error?**

Listing the contents of the `remove-example` environment should produce the following error:

```
EnvironmentLocationNotFound: Not a conda environment:
/home/codio/anaconda3/envs/remove-example
```

Removing all of the packages from an environment is the same as removing the environment itself.

Requirements File

Creating and Using a Requirements File

Conda, like other package managers, allows you to generate an official list of all packages used in a particular virtual environment.

```
conda activate learning-conda
```

Use the `list` command followed by the `--explicit` flag to generate a list of installed packages. We do not want this list sent to the terminal, so use the `>` operator to redirect the list of packages to the file `requirements.txt`.

```
conda list --explicit > requirements.txt
```

Click the link below to open the `requirements.txt` file. This is the list of packages in the environment, and it can be used to generate copies of the environment.

Notice toward the top of the requirements file, Conda tells you how to use this file to generate a new environment with the same packages. Here is the provided template:

```
# This file may be used to create an environment using:
# $ conda create --name ‹env› --file ‹this file›
# platform: linux-64
```

That means if we want to create the environment `new-env` that has the same packages as `learning-conda`, then we would use the following command:

```
conda create --name new-env --file requirements.txt
```

Packages Installed with `pip`

We know that using the `list` command produces a list of all of the packages installed. This includes packages installed with `pip`. Let's pipe list the packages and then pipe the result to the `grep` utility. We want to search for only those packages with `"python"` in the title.

```
conda list | grep "python"
```

There are two packages; Python 3.9 and `python-twitter`. **Note**, the version numbers are from the time of writing. Your version numbers may be different.

```
python                    3.9.12               h12debd9_0
python-twitter            3.5                      pypi_0
pypi
```

The `--explicit` flag is supposed to create a list of packages for recreating an environment. Using `grep` again, search the explicit list of packages for those containing `"python"`.

```
conda list --explicit | grep "python"
```

This time, however, there is only one package — Python 3.9. There is no mention of `python-twitter`.

```
https://repo.anaconda.com/pkgs/main/linux-64/python-3.9.12-
h12debd9_0.conda
```

We could use `pip` to create a requirements file. The `python-twitter` is included, but Python 3.9 is not listed in the output.

```
pip freeze
```

If your environment mixes packages from Anaconda and PyPI, the better option is to export the environment itself as `requirements.yaml`. Run the following command and click on the link below to open the newly created file.

```
conda env export > requirements.yaml
```

▼ **What is the `yaml` file extension?**
`yaml` is the file extension for YAML files. YAML stands for "Yet Another Markup Language" and was designed to be easy to read by humans. YAML is often used for configuration files.

If you scroll down a bit, you will see a section entitled `pip`. Here, you can find all of the dependencies installed via `pip`. Exporting the environment is the best way to generate a requirements file for your environment.

You can create a virtual environment from the YAML file by using `conda env create` as the command. Use the `--name`flag to give the new environment a name, and use the `--file` flag to indicate the name of the YAML file to be used.

```
conda env create --name from-yaml --file requirements.yaml
```

We can use `grep` to search the list of installed packages for those containing the phrase `"python"`. We should two results, Python 3.9 and the `python-twitter` package.

```
conda list --name from-yaml | grep "python"
```

Updating Packages

Updating Specific Packages

Click the button below to setup the `update-env` environment and install packages.

Once the environment is ready to go, activate it.

```
conda activate update-env
```

Conda uses the `update` command to update a particular package. However, Conda does not update the package to the latest version. Instead, it updates the package to the latest version that is compatible with all of the other packages in the environment. Update `scipy` with the following command.

```
conda update scipy
```

▼ **Updating Conda**

You may have seen some messages from Conda about a newer version being available.

```
==› WARNING: A newer version of conda exists. ‹==
  current version: 4.10.3
  latest version: 4.12.0

Please update conda by running
 $ conda update -n base -c defaults conda
```

Running the suggested command will update Conda.

```
conda update -n base -c defaults conda
```

You can also specify several packages to be updated all at once. Use a space to separate each package. The command below updates `matplotlib` and the `requests` packages.

```
conda update matplotlib requests
```

challenge

Try these variations:

- Update `matplotlib` and `requests` in a single command.

```
conda update matplotlib requests
```

- Update the `python-twitter` package in the `learning-conda` environment.

```
conda update python-twitter --name learning-conda
```

▼ **Why is there an error?**

Conda throws a `PackageInstalledError`. However, `python-twitter` is installed on the `learning-conda` environment.

```
conda list --name learning-conda | grep "twitter"
```

Remember, `python-twitter` was installed with `pip` since the package is not in the Anaconda repository. You will need to deactivate the current environment, activate `learning-conda`, and then update `python-twitter` with `pip`.

- Update all of the packages listed in a file with the `--file` flag.

```
conda update --file to-be-updated.txt
```

▼ **What is happening?**

The file `to-be-updated.txt` is a text file with the name of a package on each line. You can see the contents of the file with the following command.

```
cat to-be-updated.txt
```

Conda goes through the file, updating each package listed.

Updating All of the Packages

Conda can update all of the packages in a virtual environment with the `--all` flag.

```
conda update --all
```

challenge

Try this variation:

- Update all of the packages in the `learning-conda` environment.

```
conda update --all --name learning-conda
```

Other Conda Commands

Various Conda Commands

Conda has many more commands than what is covered here. If you are looking for a good resource that gives concise examples of the most popular commands, download the Conda cheat sheet. Here are a few more commands that you may find useful. You can also look at the reference documentation for all the Conda commands.

Different Versions of Python

You can create an environment with an older version of Python. At the time of writing, this version of Conda defaults to Python 3.9. However, we can use the = after `python` to specify the version number we want. The command below creates the `older` virtual environment with Python 3.7.

```
conda create --name older python=3.7
```

▼ Why do this?
Newer versions of Python (or any package) can break existing code. Sometimes developers want to choose stability over the latest feature set.

Virtual Environment Revisions

As you modify your virtual environments, Conda is taking snapshots of the state of the environment. First, activate the `learning-conda` environment.

```
conda activate learning-conda
```

You can a list of these snapshots by using the `--revisions` flag with the `list` command.

```
conda list --revisions
```

Conda provides a list of revisions to the virtual environment. Each revision has a number as well a date and time stamp. You can see the packages that are associated with the environment at each date and time. Restore `learning-conda` to revision 1.

```
conda install --revisions 1
```

Revision 1 represents the state of your virtual environment at a given date and time. If you want to "undo" lots of changes to your environment, you can roll it back to a previous revision instead of manually making changes.

Cleaning an Environment

The `clean` command can remove unused packages from an environment. We are **not** going to actually clean up an environment. However, we can use the `--dry-run` flag that will take us through the process. It will show us what packages would be removed, how much space would become free, etc. But, because it is a dry run, it will not actually remove anything. Using `--dry-run` is a good way to see what a command would do without actually making the change.

First, deactivate the `learning-conda` command. We want to see what packages would be removed from the `base` environment if we were to actually clean it.

```
conda deactivate
```

Conda gives you the ability to specify what you would like to clean from an environment. This could be packages, tarballs, cache, etc. We want to clean the environment of everything, so use the `--all` flag.

```
conda clean --all --dry-run
```

Remember, this was a dry run. Nothing was actually removed.

Packages for Conda

Conda Build

Anaconda maintains their own repository of packages. They also host packages created by their community. This brief tutorial will introduce the bare minimum to create a package ready for Conda. The code for the project is the `shapes` package, which can be found in this GitHub repository. It should be noted that Conda expects the source code for a package to be hosted on GitHub.

Start by creating a virtual environment called `build-pkg` and install the `conda-build` package as well. This package is required when building a package for Conda.

```
conda create --name build-pkg conda-build
```

Conda uses something called a “recipe” when building a package. A recipe contains three files in a dedicated folder. The files are called `bld.bat` (used for Windows machines), `build.sh` (used for macOS and Linux machines), and `meta.yaml` (metadata for the package and build process). These three files are located in the directory `shapes`.

```
shapes
├── bld.bat
├── build.sh
└── meta.yaml
```

The `bld.bat` script should look exactly like this:

```
"%PYTHON%" setup.py install
if errorlevel 1 exit 1
```

While the `build.sh` script should look exactly like this:

```
$PYTHON setup.py install     # Python command to install the
script.
```

These files are already created for you. You will use the same files with the same code in every Conda recipe. The `meta.yaml` file should look like the example below. This is the bare minimum of information needed to build a package. Notice the `git_url` is the URL for your package uploaded to GitHub. Update the `meta.yaml` file (bottom-left) with the appropriate information.

```yaml
package:
  name: shapes
  version: "0.1"

source:
  git_rev:
  git_url: https://github.com/codio-content/shapes-package

requirements:
  build:
    - python
    - setuptools

  run:
    - python

test:
  imports:
    - shapes

about:
  home: https://github.com/codio-content/shapes-package
```

Once your recipe is complete, use the following command in the terminal (top-left). Conda is going to look in the `shapes` directory and follow the build recipe. This can take several minutes and lots of text will appear in the terminal.

```
conda-build shapes
```

After the build process is finished, you should see the following text (you may have to scroll up a bit):

```
# To have conda build upload to anaconda.org automatically, use
# conda config --set anaconda_upload yes
anaconda upload /home/codio/anaconda3/conda-bld/linux-64/shapes-
0.1-py39_0.tar.bz2
```

The `anaconda upload` command will upload your package to Anaconda. This requires that you have an account for this to work. We are not going to upload our package. However, we do want to make sure the build process worked as expected. The path after `anaconda upload` is the location of the newly built package. **Important**, your location may be different than the one listed above. Be sure to copy ++your++ location and use it in the command below. Install this package with the `--use-local` flag.

```
conda install --use-local /home/codio/anaconda3/conda-bld/linux-
       64/shapes-0.1-py39_0.tar.bz2
```

Use `grep` to search the list of installed packages for any that contain `"shapes"`. You should see your package installed in the virtual environment; it is ready to be used.

```
conda list | grep "shapes"
```

Conda Forge

The Anaconda repository is divided into channels. Any packages maintained by the Anaconda company are in the official channel. If you create an account and upload your package, it will reside in your channel. `conda-forge` is a community driven channel where individuals come together to host their packages in a single channel. This is a very popular channel with high quality packages. You can read more about this channel here.

`conda-forge` has its own way of building and uploading packages. For starters, the whole process is built around GitHub. You fork and clone a repository with the recipe. After filling out the recipe (including other information like documentation), you send your package as a pull request. `conda-forge` maintainers review your package, send it back for more work, or host it on their channel.

If you are interested in building Conda packages, familiarizing yourself with `conda-forge` and their unique build process is a good idea.

Formative Assessment 1

Formative Assessment 2