



Python Repository Template - Documentation & Usage Instructions

Oliver Blagg

March 26, 2021

Contents

1	Python Respository Template Introduction	2
1.1	What is a Repository Template and why is it Necessary?	2
1.2	How to Create a Repository with This Template	3
2	Python Respository Template Features	3
2.1	Files	4
2.1.1	README.md	4
2.1.2	.gitignore	4
2.1.3	requirements.txt	4
2.1.4	LICENSE	5
2.2	Folders	5
2.2.1	docs/	5
2.2.2	src/ or sample/	5
3	Next Steps	6
3.1	Virtual Environments & Installing Requirements	6
3.1.1	Steps to create and use a virtual environment for the pip package manager	7
3.1.2	Steps to create and use a virtual environment for the conda package manager	7
3.2	Python Scripting Best Practices	8
3.3	Python Packages	8

1 Python Respository Template Introduction

Welcome to the user guide for the Python Template Repository, on the Mott MacDonald Global GitHub organisation. After reading this user guide, you should be able to understand what the Python Template Repository is, each of its features, and how it should be used.

Before reading further, please note that this user guide assumes that you already have a GitHub account and that you are a member of the Mott MacDonald Global GitHub Organisation. Furthermore a working knowledge of the Python language is assumed. If you do not yet have a GitHub account, or membership of the Mott MacDonald Global GitHub Organisation, please contact Kate Molony at Kate.Molony@mottmac.com (contact details accurate as of March 26, 2021).

1.1 What is a Repository Template and why is it Necessary?

A repository template provides users with the ability to create new repositories, that contain the same directory structure and files as the template. Using template repositories as a ‘boilerplate’ when creating a new repository can be a valuable exercise because it enables best practices in code maintenance, collaboration and curation to be clearly defined from the start of a programming project. By adhering to a common code of best practice, programming projects become easier to share, collaborate on and quality control/audit.

There is a growing abundance of repositories in the Mott MacDonald Global GitHub Organisation, many of which do not follow any best practices, and as a result are a challenge to maintain and curate. By using this repository template when beginning you Python project, you are giving

your project the best possible start and vastly extending the life of your scripts, as you are not beginning your project with avoidable technical debt.

1.2 How to Create a Repository with This Template

1. To begin creating your new Python project repository in the Mott MacDonald Global GitHub Organisation, select the 'New' button (as shown in figure 1).

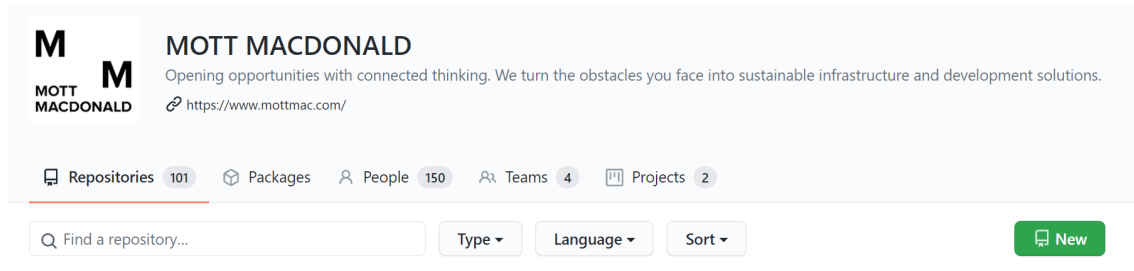


Figure 1: Screen-shot of the Mott MacDonald Global Organisation repository page.

2. Ensure that you have selected the correct repository owner, as well as a sensible repository name. Next, in the drop-down menu above, make sure to select 'Python_Template' from the available repository template options.

Regarding your repository name, it is important that your project is sensibly named to allow others to easily find your project if they are looking for it, or to give prospective organisation members the ability to easily understand the purpose of your code.

3. As usual, you can create a description about your repository and choose whether your project should be private or public. You can now select the 'Create Repository' button.

2 Python Repository Template Features

Name	Date modified	Type
docs	25/03/2021 11:38	File folder
src	25/03/2021 10:29	File folder
.gitignore	25/03/2021 10:29	Text Document
LICENSE	09/03/2021 10:22	File
main	25/03/2021 11:34	Python File
README.md	25/03/2021 10:29	MD File
requirements	25/03/2021 10:32	Text Document
Template_Instructions	25/03/2021 11:31	Microsoft Edge PDF Document

Figure 2: File Explorer view of the top level Python Repository Template directory structure.

2.1 Files

2.1.1 README.md

When an individual opens your repository on GitHub, below the top-level directory structure of your repository, the first thing they will see is the ‘compiled’ `README.md` file. This file contains information about your repository (such as description, set-up instructions, and author contact details), and these details will be shown on the ‘front page’ of your repository in GitHub. It is important that your `README.md` file is written in sufficient detail to give prospective users a clear understanding of the purpose of your project, and how it can be installed & run, as well as who to contact with any issues or questions.

As you can see from the file extension, the `README.md` document is a type of text file called a ‘markdown’ file. Markdown files are created and saved using plain text, however using specific plain text tags, users can define how this text should be formatted when the contents of the file are displayed in GitHub on your repository front page.

The `README.md` file in the Python Template Repository has been started for you, and the sections you should include have already been defined. You can add to these sections as well as create any new ones that you find necessary. For more information about how to write markdown files, you can follow this link: <https://guides.github.com/features/mastering-markdown/>.

2.1.2 .gitignore

The `.gitignore` file is a simple text file that tells Git what files or folders to leave untracked in a project. To elaborate, when using GitHub as your project source control, you can push or pull changes that you have made to files on your personal machine. There may be some files and folders that you do not want to be added to the GitHub repository, such as temporary files, log files or environment files.

The `.gitignore` file in the Python Template Repository has already been filled with the standard items that would remain untracked in a regular Python project. There should be no need for you to make any changes to this to start with, unless there are additional items that you know should remain untracked by Git.

2.1.3 requirements.txt

The `requirements.txt` file is an industry standard way of communicating to users the technical specification of your project (what packages your project uses, and hence what packages users must install for your code to run on their machine). When formatted correctly, the `requirements.txt` file can be used to install all requisite Python packages through one command line command.

The `requirements.txt` should be manually kept up to date by the project developer(s). An example `requirements.txt` file is shown in figure 2.1.3. You can see that next to the package name separated by two equals signs, the version of the package is also shown.

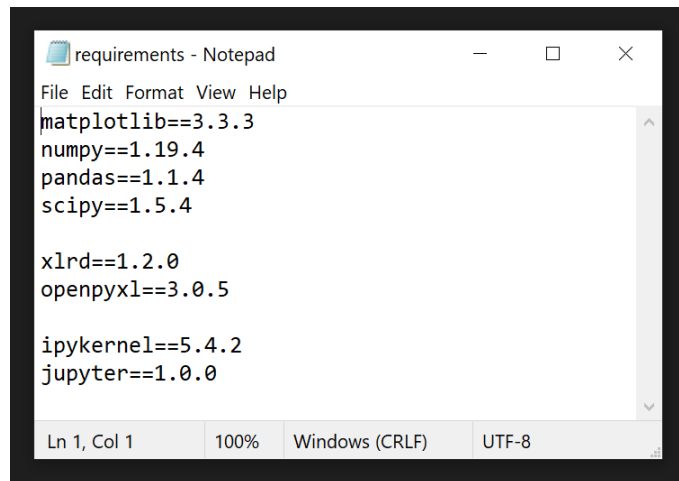
Please note that some packages used by your code will not need to be included in your `requirements.txt` file: some packages are ‘standard’ packages that come with your Python installation (for example `os`, `datetime` and `tkinter`). The following resources may be of use:

- List of Python built-in modules:

<https://www.tutorialsteacher.com/python/python-builtin-modules>

- How to find out which modules have been installed (and their versions). The method you choose should depend on what Python package manager you are using (pip or conda):

<https://www.activestate.com/resources/quick-reads/how-to-list-installed-python-packages/>



```
requirements - Notepad
File Edit Format View Help
matplotlib==3.3.3
numpy==1.19.4
pandas==1.1.4
scipy==1.5.4

xlrd==1.2.0
openpyxl==3.0.5

ipykernel==5.4.2
jupyter==1.0.0

Ln 1, Col 1    100%    Windows (CRLF)    UTF-8
```

Figure 3: Example requirements text file.

Note that by using the `> pip freeze` method, the command line output is of the correct format and can be copied straight into your `requirements.txt` file.

Please ensure that you read about virtual environments in the ‘Next Steps’ section for more information about how the `requirements.txt` file can be used.

2.1.4 LICENSE

The LICENSE file requires no interaction from the developer, this is simply a legal liability statement that protects the development team and the business from the consequences of improper or unauthorised use of the project/project code.

It is important that this file is not edited in any way unless explicitly told to do so by the Mott MacDonald Global GitHub Organisation administration team. For any questions about the license, please contact Kate Molony at Kate.Molony@mottmac.com (contact details accurate as of March 26, 2021).

2.2 Folders

2.2.1 docs/

It is important that you document your code well, both within the code itself (proper use of commenting and documentation strings) but also outside of your code. Good documentation is key because it facilitates understanding of your project/code for those wishing to use it (through user guides), as well as for those collaborating with the development process (through more detailed technical documentation).

Please note that Git does not track empty directories, so this template includes a `.deleteme` file in each of these directories to ensure that the template directory structure is reproduced correctly in the template.

2.2.2 src/ or sample/

As projects grow, code can spill into many different Python files and modules, and over time this can make the code-base hard to navigate. A common practice is for your code to be initiated

by called some ‘main’ Python file in the top level of your directory structure, which in turn can call modules and additional functions which are well organised inside another folder. The rules here however are not strict, and the important feature of this section is simply to emphasise that the code for large projects should be clearly organised and structured within a subdirectory of the repository.

In web development projects (that leverage languages such as JavaScript or TypeScript as opposed to Python), it is common best practice to name the folder in which the bulk of your code is stored `src/`. However, this rule is not so strictly adhered to in Python projects. Therefore, you may choose to rename this folder if appropriate, but it is highly recommended that the bulk of your project code remains well organised inside a dedicated folder regardless of its naming.

Additional common names of the source folder are `sample/`, or your package name if you are designing a Python package.

3 Next Steps

After you have completed all steps summarised in Section 1 of this user guide (e.g creating your new repository/project area using this template), you should look to cover the following steps.

1. Complete the necessary sections of your `README.md` file. These sections will include providing a clear description of the purpose of your repository, as well as ownership and contact details. The section on how your code should be run may well need to evolve over time.
2. Ensure that you keep your `requirements.txt` file up to date as time goes by. There are some useful resources for doing this in the ‘requirements.txt’ section. Make sure to familiarise yourself with the concept of a virtual environment.
3. Keep a copy of this user guide for future reference.
4. Ensure that you keep your documentation folder up to date, and ensure that as your code develops, it uses a sensible architecture and is organised coherently in its designated module folder. Note that the name of this folder need not be `src/`, there are other names you could give.

Furthermore, you should also familiarise yourself with the resources in this SharePoint folder, for more guidance on how to manage your repositories and collaborate with teams in the Mott MacDonald Global GitHub Organisation:

<https://mottmac.sharepoint.com/sites/AutomationComputationalDesign/Shared%20Documents/Forms/AllItems.aspx?id=%2Fsites%2FAutomationComputationalDesign%2FShared%20Documents%2FScriptHub%20Revival%2FGitHub>

3.1 Virtual Environments & Installing Requirements

A great feature of Python is that which allows the creation of an isolated development environment for a single Python application, whose dependencies are themselves independent of any wider dependencies on your machine (or in other projects). This feature is useful because different projects may require different versions of the same package, which can be problematic for projects if there are unreconcilable differences between each version of the package of interest.

By creating a virtual environment, you can develop a project using known versions of packages, and these can be shared with collaborators to ensure that they too are also using the correct package

versions. It is highly recommended that when you are working on multiple Python projects, you create a virtual environment for each one.

Follow this link for more information on virtual environments <https://realpython.com/python-virtual-environments-a-primer/>.

3.1.1 Steps to create and use a virtual environment for the pip package manager

1. Navigate into the root working directory of your local copy of the Python project using your command line.
2. Create a new directory in which to store the files that govern the virtual environment using the command line.

```
> mkdir venv
```

3. Create the virtual environment in the `venv` folder with the following command.

```
> python -m venv .\venv
```

4. Create a document that records the dependencies of the project (skip this step if such a document already exists). As outlined previously, the industry standard way of doing this is to create a text file called `requirements.txt` starting with the following command:

```
> notepad requirements.txt
```

Populate the text file using the `package==version` convention (see the ‘requirements.txt’ section for further guidance and resources):

```
matplotlib==3.3.3
numpy==1.19.4
pandas==1.1.4
```

5. Once the requirements text file has been filled, install the dependencies using the following command.

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

You can activate and deactivate your virtual environment using the following commands:

```
> venv\scripts\activate
> venv\scripts\deactivate
```

3.1.2 Steps to create and use a virtual environment for the conda package manager

1. Navigate into the root working directory of your local copy of the Python project using your command line.
2. Create your `requirements.txt` file using the same method as that of the pip package manager.
3. Enter the following command into your command line:

```
> conda create --name venv --file requirements.txt
> conda activate
```

3.2 Python Scripting Best Practices

As you begin your Python project, beyond adopting the directory structure best practices outlined in this user guide, there is another step you can take to maximise the professionalism of your project. Writing your code according to a style guide is a very beneficial exercise, because it maximises the readability of your code to others and hence vastly improves the lifetime of your code. Even a beginner at programming can write code of similar quality to an expert by adopting the industry best practices suggested in a style guide.

The following resources may be of use:

- ‘Pep 8’ Official Style Guide for Python Code : <https://www.python.org/dev/peps/pep-0008/>
- ‘Pep 257’ Official Docstring Conventions for Python Code : <https://www.python.org/dev/peps/pep-0257/>
- Python style guide used by Google : <https://github.com/google/styleguide/blob/master/pyguide.md>

3.3 Python Packages

The Python Template Repository discussed in this user guide is a relatively simple template that should give smaller scripting projects a great start. For larger projects that aim to deliver new modules and packages, there are additional features that you will need to add in the future. To get started on creating Python packages, you may find the following resources helpful:

- Basics of creating a package, and importing modules from it: <https://www.tutorialspoint.com/create-and-access-a-python-package>
- Advanced/professional package creation; industry standard practices: <https://realpython.com/pypi-publish-python-package/>

(Note that much of the above article is geared toward publishing an open source package to PyPI, which is unlikely to fall within your scope of works - the Mott MacDonald Global Github Organisation is where your project would be hosted, but there is still a lot of useful information in this article about how your repository should be structured.)

- Examples of famous Python packages, as well as links to their codebases. It may be helpful to see how large scale Python packages have structured their repositories: <https://vegibit.com/24-popular-python-repositories/>