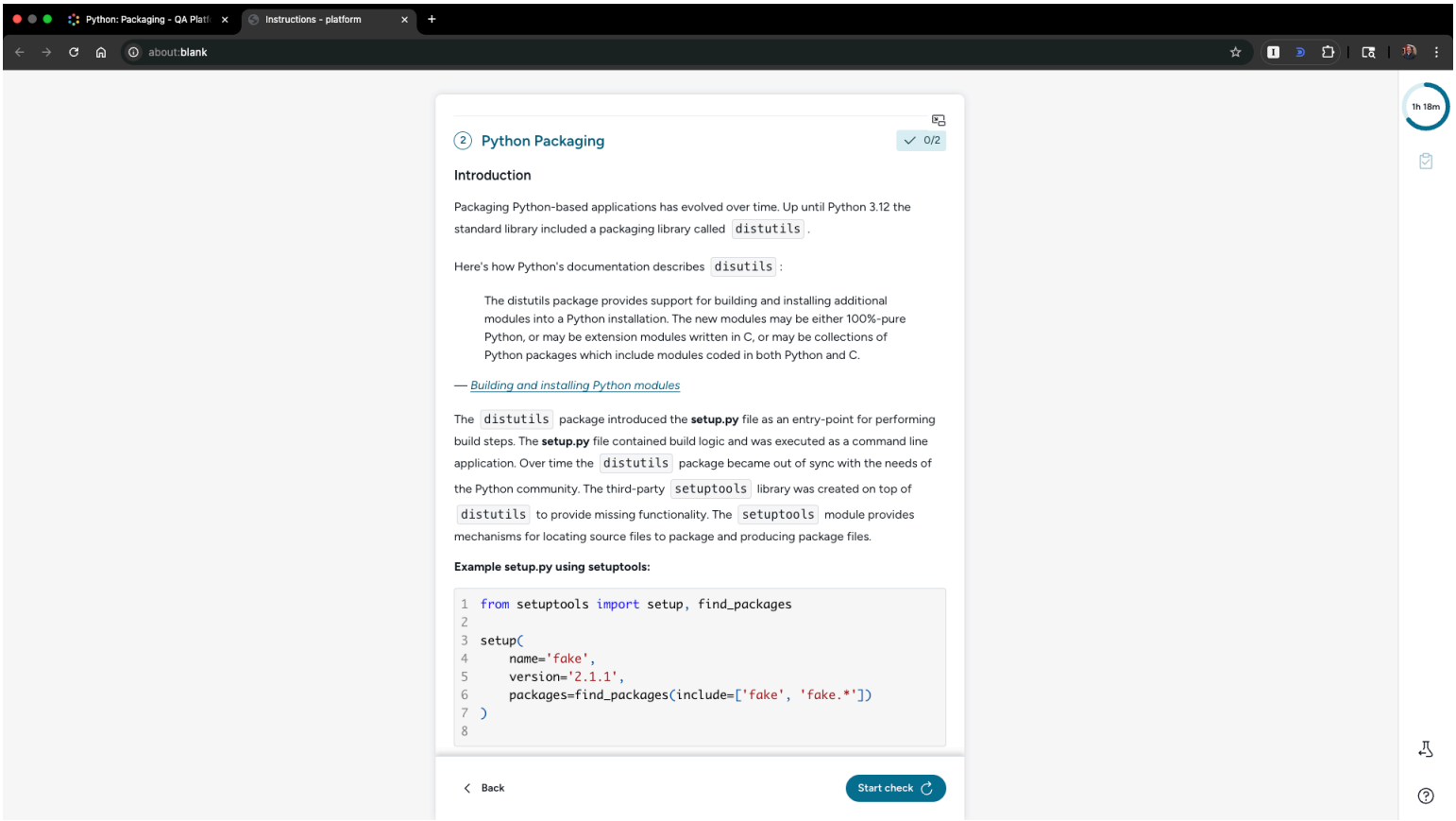
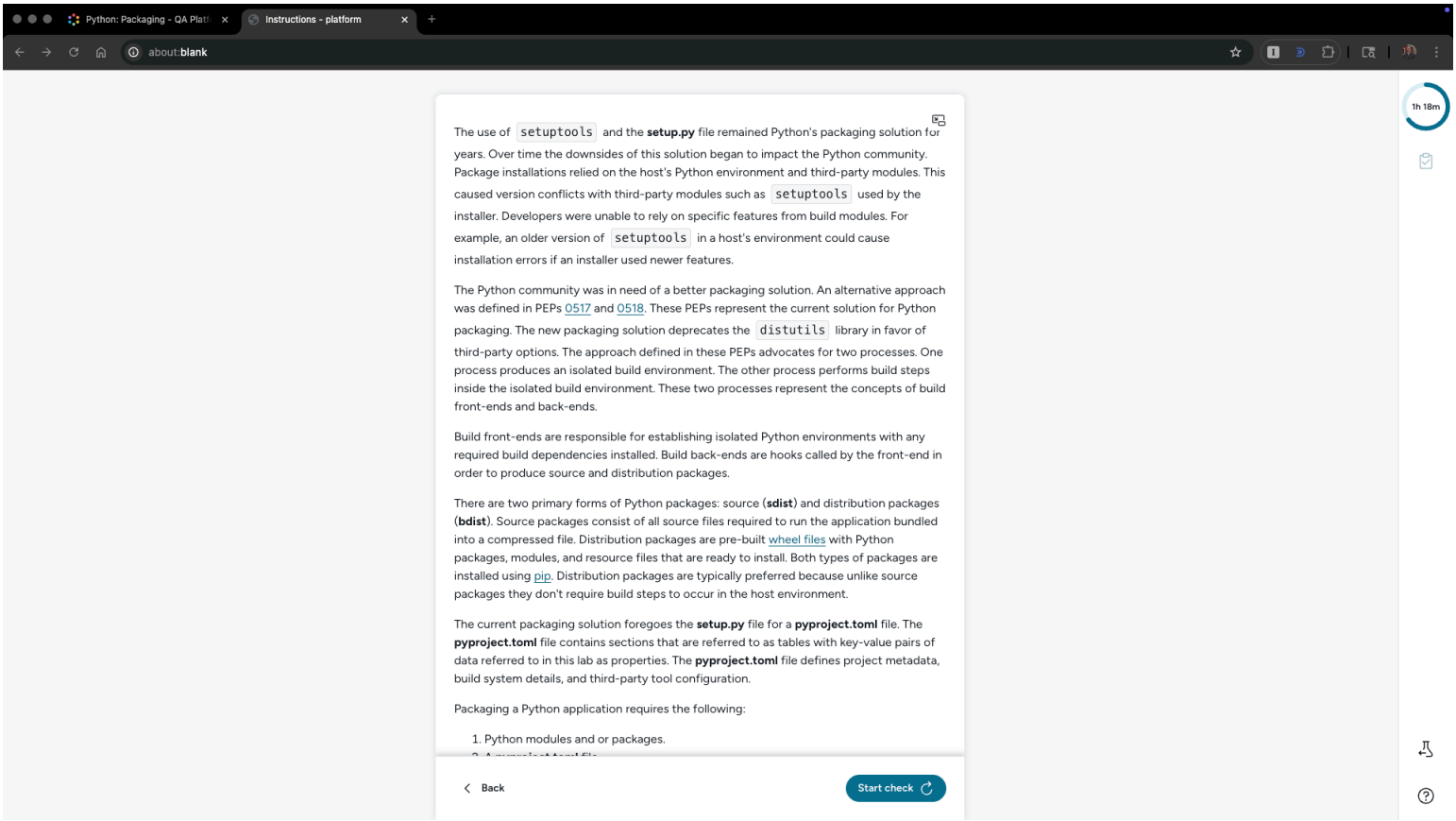

Python: Packaging - QA Platf
Instructions - platform
about:blank
1h 18m
2 Python Packaging
0/2
Introduction
Packaging Python-based applications has evolved over time. Up until Python 3.12 the standard library included a packaging library called distutils.
Here's how Python's documentation describes disutils:
The distutils package provides support for building and installing additional modules into a Python installation. The new modules may be either 100%-pure Python, or may be extension modules written in C, or may be collections of Python packages which include modules coded in both Python and C.
— Building and installing Python modules
The distutils package introduced the setup.py file as an entry-point for performing build steps. The setup.py file contained build logic and was executed as a command line application. Over time the distutils package became out of sync with the needs of the Python community. The third-party setuptools library was created on top of distutils to provide missing functionality. The setuptools module provides mechanisms for locating source files to package and producing package files.
Example setup.py using setuptools:
1 from setuptools import setup, find_packages
2
3 setup(
4 name='fake',
5 version='2.1.1',
6 packages=find_packages(include=['fake', 'fake.*'])
7)
8
Back
Start check


Python: Packaging - QA Platf
Instructions - platform
about:blank
1h 18m
The use of setuptools and the setup.py file remained Python's packaging solution for years. Over time the downsides of this solution began to impact the Python community. Package installations relied on the host's Python environment and third-party modules. This caused version conflicts with third-party modules such as setuptools used by the installer. Developers were unable to rely on specific features from build modules. For example, an older version of setuptools in a host's environment could cause installation errors if an installer used newer features.
The Python community was in need of a better packaging solution. An alternative approach was defined in PEPs 0517 and 0518. These PEPs represent the current solution for Python packaging. The new packaging solution deprecates the distutils library in favor of third-party options. The approach defined in these PEPs advocates for two processes. One process produces an isolated build environment. The other process performs build steps inside the isolated build environment. These two processes represent the concepts of build front-ends and back-ends.
Build front-ends are responsible for establishing isolated Python environments with any required build dependencies installed. Build back-ends are hooks called by the front-end in order to produce source and distribution packages.
There are two primary forms of Python packages: source (sdist) and distribution packages (bdist). Source packages consist of all source files required to run the application bundled into a compressed file. Distribution packages are pre-built wheel files with Python packages, modules, and resource files that are ready to install. Both types of packages are installed using pip. Distribution packages are typically preferred because unlike source packages they don't require build steps to occur in the host environment.
The current packaging solution foregoes the setup.py file for a pyproject.toml file. The pyproject.toml file contains sections that are referred to as tables with key-value pairs of data referred to in this lab as properties. The pyproject.toml file defines project metadata, build system details, and third-party tool configuration.
Packaging a Python application requires the following:
1. Python modules and or packages.
Back
Start check

1. Python modules and or packages.
2. A **pyproject.toml** file.
3. A build front-end.
4. A build back-end.

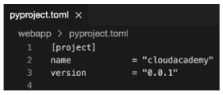
This lab walks through the process of creating source and distribution packages for an existing web application. Packages will be created for the application and installed into an isolated virtual environment.

Instructions

The **pyproject.toml** file allows project specific metadata to be specified under the `project` table. The allowed properties are defined in [PEP-0621(https://peps.python.org/pep-0621/). These properties are build-system independent.

1. Add the following code to the **webapp/pyproject.toml** file.

```
[project]
name            = "cloudacademy"
version         = "0.0.1"
```



When pip installs a package it also installs all required dependencies. The `dependencies` property accepts a list of requirement specifiers that define the runtime dependencies of the package.

2. Append the dependencies to the `project` table.

```
dependencies    = [
    "rich",
    "Flask",
```

Python: Packaging - QA Platf | Instructions - platform | about:blank

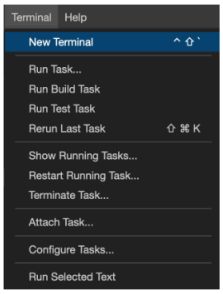
1h 17m

2. Append the dependencies to the `project` table.

```
dependencies    = [
    "rich",
    "Flask",
    "Pillow",
    "bcrypt"
]
```

```
[project]
name            = "cloudacademy"
version         = "0.0.1"
dependencies    = [
    "rich",
    "Flask",
    "Pillow",
    "bcrypt"
]
```

3. Open the built-in terminal pane **(Terminal > New Terminal)**.



Back

Start check

Python: Packaging - QA Platf | Instructions - platform

about:blank

1h 17m

A build front-end is required in order to read the **pyproject.toml** file and interact w a build back-end. Multiple build front-ends exist in the Python community. The build library is a minimalist build front-end.

4. Install the `build` front-end.

```
python3 -m pip install build
```

```
Collecting build
  Downloading build-0.9.0-py3-none-any.whl (17 kB)
Collecting tomli>=1.0.0
  Downloading tomli-2.0.1-py3-none-any.whl (12 kB)
Collecting packaging>=19.0
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.8/40.8 kB 13.0 MB/s eta 0:00:00
Collecting pep517>=0.9.1
  Downloading pep517-0.13.0-py3-none-any.whl (18 kB)
Collecting pyparsing!=3.0.5,>=2.0.2
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.3/98.3 kB 7.5 MB/s eta 0:00:00
Installing collected packages: tomli, pyparsing, pep517, packaging, build
Successfully installed build-0.9.0 packaging-21.3 pep517-0.13.0 pyparsing-3.0.9 tomli-2.0.1
```

Currently the **pyproject.toml** file only contains project related information and no build system details. Build front-ends use default settings for optional tables and properties. The `setuptools` library is used as the default build back-end.

Running the build process now will result in the front-end using the default `setuptools` back-end. The `setuptools` module includes its own default settings that determine which files to include in the packages.

5. Build the application.

```
python3 -m build ./webapp
```

```
* Creating venv isolated environment...
* Installing packages in isolated environment... (setuptools >= 40.8.0, wheel)
* Getting build dependencies for sdist...
running egg_info
creating cloudacademy.egg-info
writing cloudacademy.egg-info/PKG-INFO
writing dependency_links to cloudacademy.egg-info/dependency_links.txt
writing requirements to cloudacademy.egg-info/requires.txt
writing top-level names to cloudacademy.egg-info/top_level.txt
writing manifest file 'cloudacademy.egg-info/SOURCES.txt'
reading manifest file 'cloudacademy.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'cloudacademy.egg-info/SOURCES.txt'
```

Back

Start check

...

```
adding 'webapp/__init__.py'
adding 'webapp/effects.py'
adding 'webapp/userman.py'
adding 'webapp/wsgi.py'
adding 'cloudacademy-0.0.1.dist-info/METADATA'
adding 'cloudacademy-0.0.1.dist-info/WHEEL'
adding 'cloudacademy-0.0.1.dist-info/top_level.txt'
adding 'cloudacademy-0.0.1.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
Successfully built cloudacademy-0.0.1.tar.gz and cloudacademy-0.0.1-py3-none-any.whl
```

By default the `build` front-end creates both source and distribution packages into the **dist/** directory. However, they can be created independently and into other directories with the --outdir flag.

6. Observe the build artifacts.

```
ls -lash webapp/dist
```

```
total 20K
4.0K drwxr-xr-x 2 theia theia 4.0K Nov 11 17:45 .
4.0K drwxrwxrwx 5 root  root  4.0K Nov 11 17:45
8.0K -rw-r--r-- 1 theia theia 4.7K Nov 11 17:45 cloudacademy-0.0.1-py3-none-any.whl
4.0K -rw-r--r-- 1 theia theia 3.7K Nov 11 17:45 cloudacademy-0.0.1.tar.gz
```

The *.whl* file is the distribution package and the compressed *.tar.gz* file is the source distribution.

Build system details are defined in the `build-system` table. The `requires` property is used to install build system packages such as build back-ends. The property accepts a list of requirement specifiers as strings. The build-backend property accepts a string pointing to the build back-end module. The values defined below were found in the `setuptools` documentation.

Breaking the build process into a front-end and back-end allows build dependencies to be isolated from the host Python environment.

The following table and associated properties explicitly define the build system to use `setuptools`. These settings could be omitted since they match the default values. However, explicit settings for important tables such as `build-system` can

Back

Start check

1h 17m

Python: Packaging - QA Platf | Instructions - platform | about:blank

1h 17m

7. Append the following code to the **webapp/pyproject.toml** file.

```
[build-system]
requires        = ["setuptools"]
build-backend   = "setuptools.build_meta"
```

```
[project]
name            = "cloudacademy"
version         = "0.0.1"
dependencies    = [
    "rich",
    "Flask",
    "Pillow",
    "bcrypt"
]

[build-system]
requires        = ["setuptools"]
build-backend   = "setuptools.build_meta"
```

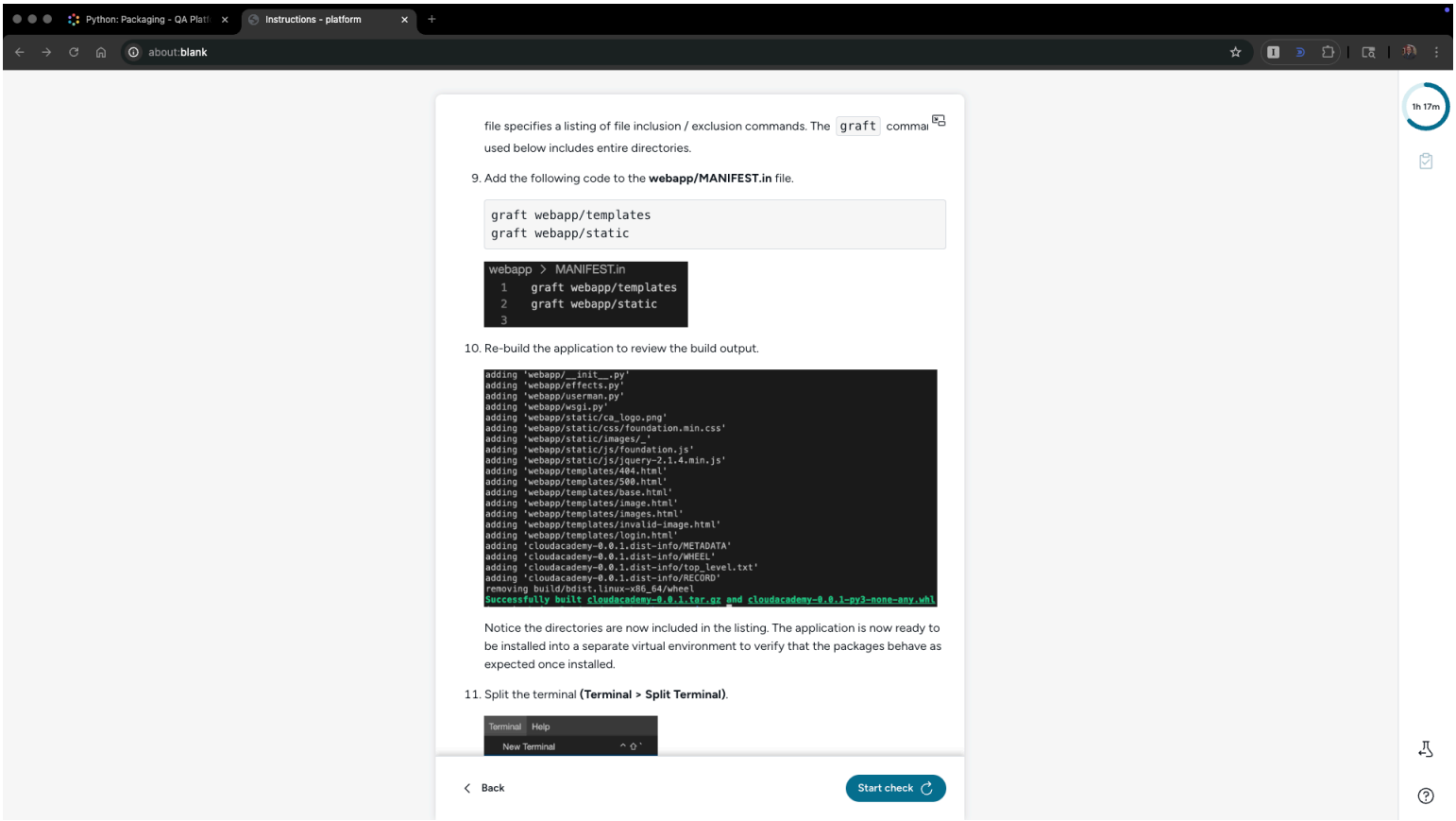
The default settings of `setuptools` is causing some directories/files to be omitted from the packages. Specifically the **static/** and **templates/** directories.

8. Re-build the application to review the build output.

```
adding 'webapp/__init__.py'
adding 'webapp/effects.py'
adding 'webapp/userman.py'
adding 'webapp/wsgi.py'
adding 'cloudacademy-0.0.1.dist-info/METADATA'
adding 'cloudacademy-0.0.1.dist-info/WHEEL'
adding 'cloudacademy-0.0.1.dist-info/top_level.txt'
adding 'cloudacademy-0.0.1.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
Successfully built cloudacademy-0.0.1.tar.gz and cloudacademy-0.0.1-py3-none-any.whl
```

Notice the listing of added files doesn't include files located in the missing directories. Each back-end defines its own configuration settings for details such as which files to include in the packages. The `setuptools` module includes rules for determining which files to package. The use of a MANIFEST.in file allows `setuptools` finer control over the included and excluded files. The **MANIFEST.in**

Back | Start check



file specifies a listing of file inclusion / exclusion commands. The `graft` comma used below includes entire directories.

9. Add the following code to the **webapp/MANIFEST.in** file.

```
graft webapp/templates
graft webapp/static
```

webapp > MANIFEST.in

```
1  graft webapp/templates
2  graft webapp/static
3
```

10. Re-build the application to review the build output.

```
adding 'webapp/__init__.py'
adding 'webapp/effects.py'
adding 'webapp/userman.py'
adding 'webapp/wsgi.py'
adding 'webapp/static/ca_logo.png'
adding 'webapp/static/css/foundation.min.css'
adding 'webapp/static/images/_'
adding 'webapp/static/js/foundation.js'
adding 'webapp/static/js/jquery-2.1.4.min.js'
adding 'webapp/templates/404.html'
adding 'webapp/templates/500.html'
adding 'webapp/templates/base.html'
adding 'webapp/templates/image.html'
adding 'webapp/templates/images.html'
adding 'webapp/templates/invalid-image.html'
adding 'webapp/templates/login.html'
adding 'cloudacademy-0.0.1.dist-info/METADATA'
adding 'cloudacademy-0.0.1.dist-info/WHEEL'
adding 'cloudacademy-0.0.1.dist-info/top_level.txt'
adding 'cloudacademy-0.0.1.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
Successfully built cloudacademy-0.0.1.tar.gz and cloudacademy-0.0.1-py3-none-any.whl
```

Notice the directories are now included in the listing. The application is now ready to be installed into a separate virtual environment to verify that the packages behave as expected once installed.

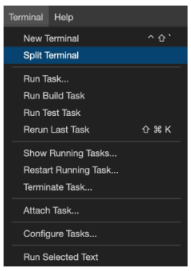
11. Split the terminal **(Terminal > Split Terminal)**.

Back

Start check

Python: Packaging - QA Platf | Instructions - platform | about:blank

11. Split the terminal **(Terminal > Split Terminal)**.



Going forward the left terminal window is designated the **build terminal** and the right terminal the **install terminal**.

12. Using the **install terminal**, create and activate a virtual environment.

```
python3 -m venv install_env
deactivate
source install_env/bin/activate
```

The **webapp/webapp/__init__.py** file includes a print statement at the top of the file which displays the text: *greetings from webapp*. Importing the module will result in the message being printed to the console. This is going to be used to determine the success of the package installation.

13. **(install terminal)** Observe that the *cloudacademy* package is currently uninstalled in this virtual environment.

```
python3 -m pip show cloudacademy
```

```
WARNING: Package(s) not found: cloudacademy
```

Back

Start check

1h 17m

Python: Packaging - QA Platf | Instructions - platform | about:blank

1h 17m

14. **(install terminal)** Install the distribution package.

```
python3 -m pip install webapp/dist/cloudacademy-0.0.1-py3-none-
```

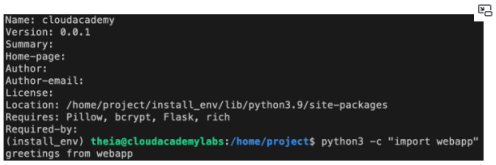
```
Processing ./webapp/dist/cloudacademy-0.0.1-py3-none-any.whl
Collecting bcrypt
  Downloading bcrypt-4.0.1-cp36-abi3-manylinux_2_28_x86_64.whl (593 kB)
     |████████████████████████████████| 593 kB 5.0 MB/s
Collecting Flask
  Downloading Flask-2.2.2-py3-none-any.whl (101 kB)
     |████████████████████████████████| 101 kB 31.2 MB/s
Collecting Pillow
  Downloading Pillow-9.3.0-cp39-cp39-manylinux_2_28_x86_64.whl (3.3 MB)
     |████████████████████████████████| 3.3 MB 184.4 MB/s
Collecting rich
  Downloading rich-12.6.0-py3-none-any.whl (237 kB)
     |████████████████████████████████| 237 kB 114.6 MB/s
Collecting click>=8.0
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
     |████████████████████████████████| 96 kB 17.0 MB/s
Collecting Werkzeug>=2.2.2
  Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)
     |████████████████████████████████| 232 kB 121.1 MB/s
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting importlib-metadata>=3.6.0
  Downloading importlib_metadata-5.0.0-py3-none-any.whl (21 kB)
Collecting Jinja2>=3.0
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
     |████████████████████████████████| 133 kB 134.6 MB/s
Collecting zipp>=0.5
  Downloading zipp-3.10.0-py3-none-any.whl (6.2 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
Collecting pygments<3.0.0,>=2.6.0
  Downloading Pygments-2.13.0-py3-none-any.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 123.2 MB/s
Collecting commonmark<0.10.0,>=0.9.0
  Downloading commonmark-0.9.1-py2.py3-none-any.whl (51 kB)
     |████████████████████████████████| 51 kB 19.7 MB/s
Installing collected packages: zipp, MarkupSafe, Werkzeug, pygments, Jinja2, itsdangerous, importlib-
metadata, commonmark, click, rich, Pillow, Flask, bcrypt, cloudacademy
Successfully installed Flask-2.2.2 Jinja2-3.1.2 MarkupSafe-2.1.1 Pillow-9.3.0 Werkzeug-2.2.2 bcrypt-4
.0.1 click-8.1.3 cloudacademy-0.0.1 commonmark-0.9.1 importlib-metadata-5.0.0 itsdangerous-2.1.2 pygm
ents-2.13.0 rich-12.6.0 zipp-3.10.0
WARNING: You are using pip version 21.2.4; however, version 22.3.1 is available.
You should consider upgrading via the '/home/project/install_env/bin/python3 -m pip install --upgrade
 pip' command.
```

15. **(install terminal)** Verify that the package was successfully installed.

```
python3 -m pip show cloudacademy
python3 -c "import webapp"
```

```
Name: cloudacademy
Version: 0.0.1
Summary:
Home-page:
Author:
Author-email:
License:
```

Back | Start check

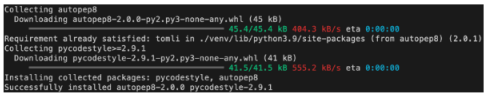


The **pyproject.toml** file includes a table named `tools` that allows supported Python packages from PyPI to be configured using a period to separate namespaces. This table commonly configures build back-ends, linters, code coverage modules, etc.

Libraries such as `setuptools`, `pytest`, `autopep8`, among others already support using the `tools` table as a configuration mechanism.

16. Using the **(build terminal)** Install the autopep8 package.

```
python3 -m pip install autopep8
```



17. Append the following table to the **webapp/pyproject.toml** file.

```
[tool.autopep8]
max_line_length = 120
in-place        = true
recursive       = true
aggressive      = 3
exclude         = "*/build/**,*.egg-info"
```

Back

Start check

1h 16m

Python: Packaging - QA Platf | Instructions - platform | about:blank

1h 16m

```
[project]
name            = "cloudacademy"
version         = "0.0.1"
dependencies    = [
    "rich",
    "Flask",
    "Pillow",
    "bcrypt"
]

[build-system]
requires        = ["setuptools"]
build-backend   = "setuptools.build_meta"

[tool.autopep8]
max_line_length = 120
in-place        = true
recursive       = true
aggressive      = 3
exclude         = "*/build/**,*.egg-info"
```

18. **(build terminal)** Run `autopep8` against the webapp Python package.

```
autopep8 -v webapp/
```

```
enable pyproject.toml config: key=max_line_length, value=120
enable pyproject.toml config: key=in-place, value=True
enable pyproject.toml config: key=recursive, value=True
enable pyproject.toml config: key=aggressive, value=3
enable pyproject.toml config: key=exclude, value=*/build/**,*.egg-info
```

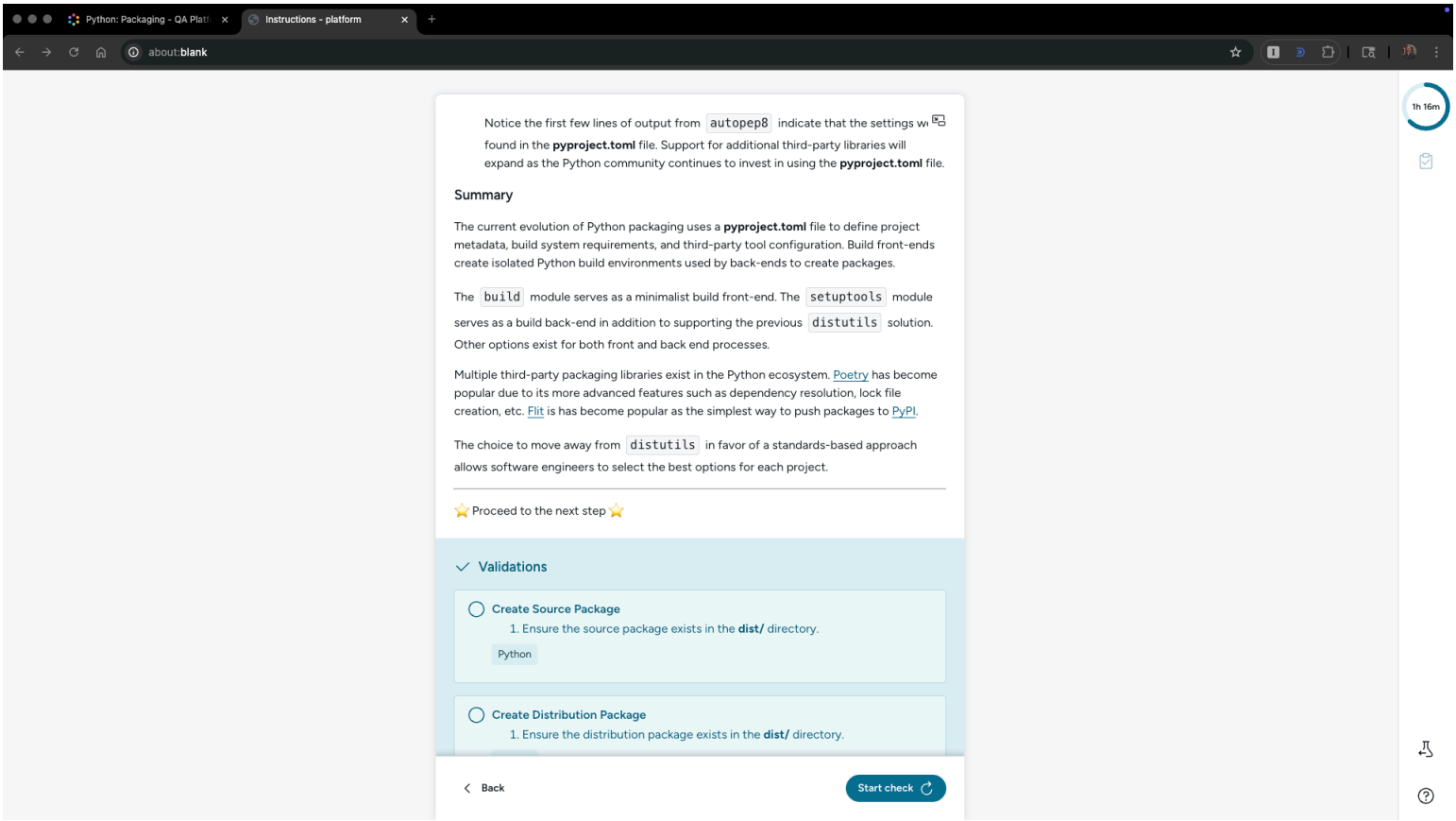
Notice the first few lines of output from `autopep8` indicate that the settings were found in the **pyproject.toml** file. Support for additional third-party libraries will expand as the Python community continues to invest in using the **pyproject.toml** file.

Summary

The current evolution of Python packaging uses a **pyproject.toml** file to define project metadata, build system requirements, and third-party tool configuration. Build front-ends create isolated Python build environments used by back-ends to create packages.

The `build` module serves as a minimalist build front-end. The `setuptools` module serves as a build back-end in addition to supporting the previous `distutils` solution.

Back | Start check


Python: Packaging - QA Platf
Instructions - platform
about:blank
Notice the first few lines of output from autopep8 indicate that the settings w
found in the pyproject.toml file. Support for additional third-party libraries will
expand as the Python community continues to invest in using the pyproject.toml file.
Summary
The current evolution of Python packaging uses a pyproject.toml file to define project metadata, build system requirements, and third-party tool configuration. Build front-ends create isolated Python build environments used by back-ends to create packages.
The build module serves as a minimalist build front-end. The setuptools module serves as a build back-end in addition to supporting the previous distutils solution. Other options exist for both front and back end processes.
Multiple third-party packaging libraries exist in the Python ecosystem. Poetry has become popular due to its more advanced features such as dependency resolution, lock file creation, etc. Flit is has become popular as the simplest way to push packages to PyPI.
The choice to move away from distutils in favor of a standards-based approach allows software engineers to select the best options for each project.
Proceed to the next step
Validations
Create Source Package
1. Ensure the source package exists in the dist/ directory.
Python
Create Distribution Package
1. Ensure the distribution package exists in the dist/ directory.
Back
Start check
1h 16m