**Setting up Project**

***Writing a Spec***

We won’t write a full project spec here, but we we’ll lay a few clear goals to keep development process focused.

***Creating a Virtual Environment***

Create a new directory for your project called pytest\_project, switch to that directory in a terminal, and enter the following code to create a virtual environment.

|  |
| --- |
| python -m venv pytest\_env |

***Activating the Virtual Environment***

Run the script *Activate* for the project.

|  |
| --- |
| pytest\_env\Scripts\Activate |

***Installing Packages with pip***

*pip* is the tool used to install Python packages, and it is installed as part of your python installation.

To check the version of *pip* and which version of Python it’s tied to, use the following script (on a virtual environment):

|  |
| --- |
| (pytest\_env) C:\ pytest\_project>pip --version  pip 21.1.1 from c:\ pytest\_project\pytest\_env\lib\site-packages\pip (python 3.9) |

To list the packages you have currently installed with pip, use pip list.

|  |
| --- |
| (pytest\_env) C:\ pytest\_project>pip list  Package Version  ---------- -------  pip 21.1.1  setuptools 56.0.0 |

***Installing pytest with pip***

Use *pip* to install *pytest* package as follows:

|  |
| --- |
| (pytest\_env) C:\ pytest\_project>pip install pytest  Collecting pytest  Downloading pytest-6.2.5-py3-none-any.whl (280 kB)  …  Installing collected packages: pyparsing, toml, py, pluggy, packaging, iniconfig, colorama, attrs, atomicwrites, pytest  Successfully installed atomicwrites-1.4.0 attrs-21.2.0 colorama-0.4.4 iniconfig-1.1.1 packaging-21.2 pluggy-1.0.0 py-1.11.0 pyparsing-2.4.7 pytest-6.2.5 toml-0.10.2 |

Now you can check the packages installed with *pip*:

|  |
| --- |
| (pytest\_env) C:\ pytest\_project>pip list  Package Version  ------------ -------  atomicwrites 1.4.0  attrs 21.2.0  colorama 0.4.4  iniconfig 1.1.1  packaging 21.2  pip 21.1.1  pluggy 1.0.0  py 1.11.0  pyparsing 2.4.7  pytest 6.2.5  setuptools 56.0.0  toml 0.10.2 |

**Getting Started with Pytest**

***Running pytest with a sample namedtuple***

Python supports a type of container like [dictionaries](https://www.geeksforgeeks.org/python-set-4-dictionary-keywords-python/) called “**namedtuple()**” present in the module, “[**collections**](https://www.geeksforgeeks.org/python-collections-module/)“. Like dictionaries, they contain keys that are hashed to a particular value. But on contrary, it supports both access from key-value and iteration, the functionality that dictionaries lack.

This is the first example on testing defaults (page 5 in book). I had to modify the code to better suit Python 3.7 and higher.

|  |  |
| --- | --- |
| *test\_three.py* | """Test the Task Data Type"""  from collections import namedtuple  fields = ('summary', 'owner', 'done', 'id' )  Task = namedtuple('Task', fields, defaults=(None, None, False, None))  def test\_defaults():  t1 = Task()  t2 = Task(None, None, False, None)  assert t1 == t2  def test\_member\_access():  """Check .field functionality of namedtuple."""  t = Task('buy milk', 'brian')  assert t.summary == 'buy milk'  assert t.owner == 'brian'  assert (t.done, t.id) == (False, None) |

***Running Only One Test***

Specify the file directory, and add a ::test\_name, like this:

|  |
| --- |
| (pytest\_env) C:\ pytest\_project\ch1\tasks> pytest test\_three.py::test\_defaults |

***Using Options***

**--collect-only**

The –collect-only option shows you which tests will be run with the given options and configuration.

|  |
| --- |
| (pytest\_env) C:\ pytest\_project\ch1\tasks>pytest --collect-only |

**-k Expression**

The –k option lets you use an expression to find what test functions to run. It can be used as shortcut to running an individual test if its name is unique, or running a set of tests that have a common prefix or suffix in their names.

|  |
| --- |
| (pytest\_env) C:\pytest\_project\ch1\tasks>pytest -k "member" --collect-only |

**Writing Test Functions**

Source code for Tasks project is located at: [Python Testing with pytest: Simple, Rapid, Effective, and Scalable by Brian Okken (pragprog.com)](https://pragprog.com/titles/bopytest/python-testing-with-pytest/#resources)

**Testing a package**

***Installing a Package Locally***

Here is the test\_task.py:

|  |
| --- |
| C:\...\pytest\_project\bopytest-code\code\tasks\_proj\tests\unit |
| """Test the Task data type."""  from tasks import Task  def test\_asdict():  """asdict() should return a dictionary."""  t\_task = Task('do something', 'okken', True, 21)  t\_dict = t\_task.\_asdict()  expected = {'summary': 'do something',  'owner': 'okken',  'done': True,  'id': 21}  assert t\_dict == expected  def test\_replace():  """replace() should change passed in fields."""  t\_before = Task('finish book', 'brian', False)  t\_after = t\_before.\_replace(id=10, done=True)  t\_expected = Task('finish book', 'brian', True, 10)  assert t\_after == t\_expected  def test\_defaults():  """Using no parameters should invoke defaults."""  t1 = Task()  t2 = Task(None, None, False, None)  assert t1 == t2  def test\_member\_access():  """Check .field functionality of namedtuple."""  t = Task('buy milk', 'brian')  assert t.summary == 'buy milk'  assert t.owner == 'brian'  assert (t.done, t.id) == (False, None) |

The best way to allow the tests to be able to import tasks is to install tasks locally using pip. This is possible because there’s a setup.py file present to direct pip.

Follow the installation instructions as listed in page 26 and 27: (Don’t forget to run the virtual environment first)

|  |
| --- |
| cd/path/to/code  pip install ./tasks\_proj  pip install –e ./tasks\_proj/  pip install pytest |

Now you can try running tests:

|  |
| --- |
| Cd/path/to/code/ch2/tasks\_proj/tests/unit  Pytest test\_task.py |

The result is:

|  |
| --- |
| ============================================================================================== test session starts ==================================================================  platform win32 -- Python 3.9.5, pytest-6.2.5, py-1.11.0, pluggy-1.0.0  rootdir: C:\Users\Windows 10\Desktop\python\_work\_2020\Learning\_python\pytest\_project\bopytest-code\code\tasks\_proj\tests, configfile: pytest.ini  collected 4 items  test\_task.py .... [100%]  =============================================================================================== 4 passed in 0.04s ===================================================================== |

The import worked! The rest of our tests can now safely use import tasks. Now let’s write some tests.

I started chapter 2. I had problems with importing from tasks while running pytest.

See this for more details on how to fix the problem:

[pytest cannot import module while python can - Stack Overflow](https://stackoverflow.com/questions/41748464/pytest-cannot-import-module-while-python-can)

**Partial Solution:**

I got to successfully test all functions except the first one (test\_asdict()).

To do so, I had to make some changes in the headers of src/tasks/api.py and tests/unit/test\_task:

**src/tasks/api.py:**

from collections import namedtuple

#from six import string\_types **# I had to comment out that line**

**tests/unit/test\_task:**

import sys

sys.path.append('C:/Users/Windows 10/Desktop/python\_work\_2020/Learning\_python/pytest\_project/bopytest-code/code/tasks\_proj/src/tasks')

import api

from api import Task

Now we can select the tests that work:

|  |
| --- |
| (pytest\_env) C:\Users\Windows 10\Desktop\python\_work\_2020\Learning\_python\pytest\_project\bopytest-code\code\tasks\_proj\tests\unit>pytest -k "replace or defaults or member" |

And the result is:

|  |
| --- |
| test\_task.py ... [100%]  ======================================================================================== 3 passed, 1 deselected in 0.02s =========================== |