Lab: Testing

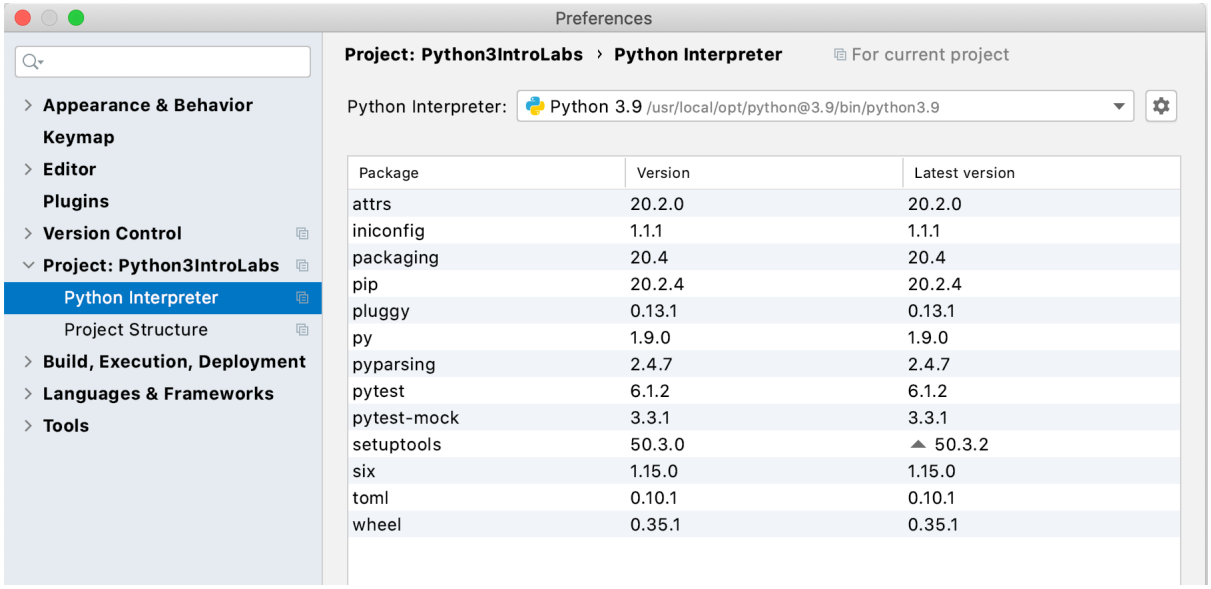
Write a set of tests for your **get user input yes or no function** class.

These tests should validate that the function operatesin the way you expect.

Don’t forget to add pytest and mock to the list of libraries in your environment.

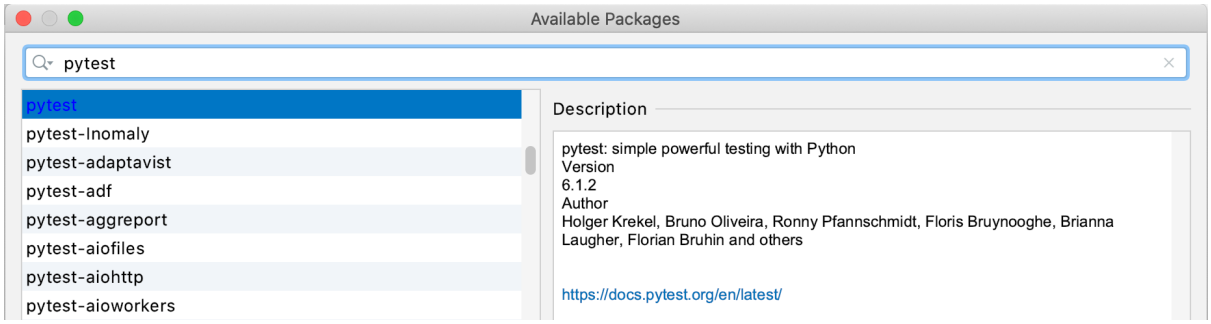
This can be done interactively through PyCharm.

Open your Preferences (on a Mac) Settings (on Windows) and select project and then the python interpreter:



You can then add a package to the current environment for the specified interpreter using the ‘+’ button at the bottom of the dialog.

To add pytest type in pytest to the resulting search bar:



Hints

The get function will take input from the user.

This input can be mocked out for testing purposes using the mock module and the with statement.

For example:

```
def test_get_user_yes_or_no_with_y():
    with mock.patch.object(builtins, 'input', lambda _: 'y'):
        response = get_user_yes_or_no('input name: ')
        assert response == 'y'
```

In the above mock.patch.object is used to mock out the builtin input function with another function (the lambda in this case) which will always return ‘y’. There is a parameter to this function, the prompt but we are not using it, and this is represented by the ‘_’.

The use of the with statement ensures that the effect of the mock version of the input function is limited to the lines within the with statement. Once the with statement completes then the input function is reset.

If you need the function to return different values depending on the number of times it is called, then you can use a named function that uses a global or a nonlocal variable, for example:

```
def test_get_user_yes_or_no_with_H():
    count = 0
    def generate_input(prompt):
        nonlocal count
        if count == 0:
            count = count + 1
            return 'H'
        else:
            return 'y'

    with mock.patch.object(builtins, 'input', generate_input):
        response = get_user_yes_or_no('input name: ')
        assert response == 'y'
```

Note that count above is a local variable to the test_get_user_yes_or_no_with_H function but the nested (inner) function generate_input needs to access it. This cvan be done by marking the count variable as nonlocal (which allows it to access the outter scope variable rather than a global variable).

If you have time, you could consider writing tests for other functions you have created.