

ECSE-428: Software Engineering Practices

Assignment B Test Driven Development

Name: Eric Vuong

ID: 260630559

Name: Victor Vuong

ID: 260742996

Due: March 04, 2018

Hyper links

[Introduction:](#)

[Test 1: No arguments](#)

[Test 2: Too few arguments](#)

[Test 3: Too many arguments](#)

[Test 4: Invalid starting postal code](#)

[Test 5: Invalid destination postal code](#)

[Test 6: Invalid length format](#)

[Test 7: Length too short](#)

[Test 8: Length too long](#)

[Test 9: Invalid width format](#)

[Test 10: Width too short](#)

[Test 11: Width too long](#)

[Test 12: Invalid height format](#)

[Test 13: Height too short](#)

[Test 14: Height too big](#)

[Test 15: Correct volume price](#)

[Test 16: Invalid weight format](#)

[Test 17: Weight too low](#)

[Test 18: Weight too high](#)

[Test 19: Correct weight price](#)

[Test 20: Invalid post-type](#)

[Test 21: Correct Regular post-type cost](#)

[Test 22: Correct Xpress post-type cost](#)

[Test 23: Correct Priority post-type cost](#)

[Test 24: Correct post rate cost from calculator \(program\)](#)

[Source code](#)

[Source Code: Test](#)

Introduction:

The purpose of this assignment is to develop a postal rate calculator, which computes postal rates for sending parcels in Canada using Test Driven Development software development process. The development process will be performed in PyCharm IDE for Python programming language.

Pytest framework will be used to write and run unit tests throughout the development. All unit tests are placed in the file : **test.py**. The application source code is located in the **postal_calc.py** file.

For the postal rate application, there are a few assumptions made in order to simplify the development process. First, a list of postal codes will be generated from a simple CSV file, where the format will only consist of existing Canadian postal codes. Second, the weight price of the parcel is also linearly dependant on the weight of the parcel with a rate of 0.5\$ / kg and the volume price of the parcel will have a rate of 5\$ / m^3 . Finally, the total rate will be calculated as follows:

$$\text{postal parcel price} = \text{weight price} + \text{volume price} + \text{post type price}$$

The dimensions are inspired from the actual Canada Post parcel rate rules:

<https://www.canadapost.ca/cpptools/apps/far/business/findARate?execution=e1s1>

<https://www.canadapost.ca/tools/pg/prices/SBParcels-e.pdf>

However, for the purpose of this assignment, some of the assumptions are simplified, such as calculations for the girth of a parcel, weight and dimension prices, etc.

Assumptions and input variables:

A valid postal code is solely dependant on the lookup table found in the .csv file. Thus, it is assumed that the program is contingent with the .csv file being in the same directory.

The program will take 7 input variables:

- Source Postal Code LNL NLN (letter, N = number)
- Destination Postal Code LNL NLN (L = letter, N = number)
- Length: length of the parcel in cm, between 10-200 cm.
- Width: width of the parcel in cm, between 10-200 cm.
- Height: height of the parcel in cm, between 10-200 cm.
- Weight: weight of the parcel in kg, between 0.1-30 kg.
- Post Type: [Regular, Xpress, Priority]

The possible outputs are separated into 2 categories: error conditions and valid conditions:

Error conditions:

- Invalid source postal code
- Invalid destination postal code
- Invalid length
- Invalid width
- Invalid height
- Invalid weight
- Invalid post-type

Valid Condition(s):

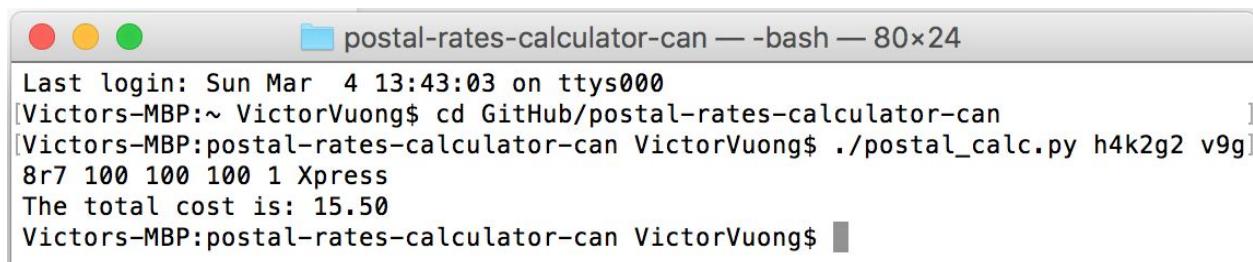
- Postal rate price based on Canada Post parcel Rate rules with slight modifications.

The CSV file: ***postal-code-generator.csv*** has been filled in the following and will be used to look up valid postal code entries from user.

```
1   code
2   h4k2g2
3   v9g8r7
4   g3h0j9
5   g1k8n2
6   e9cm1j
7   t8rl6b
8   p7k6b2
9   t7sh4h
10  v1r7b0
11  j5k5n1
12  e5n9m1
13  t9v7m1
14  k7h3v1
15  g7b1y7
16  e6g2k2
17  t1w8n6
18  s4a9m3
19  g8j4o2
20  s9v8c3
21  l3m0j2
22  k7v0e8
23  t4l3j8
24  v9g0n2
25  e5s3j8
26  j5x0n1
27  v3a6k0
28  v3a7v3
29  n3y9y3
30  g5x0m8
```

Executing the final application:

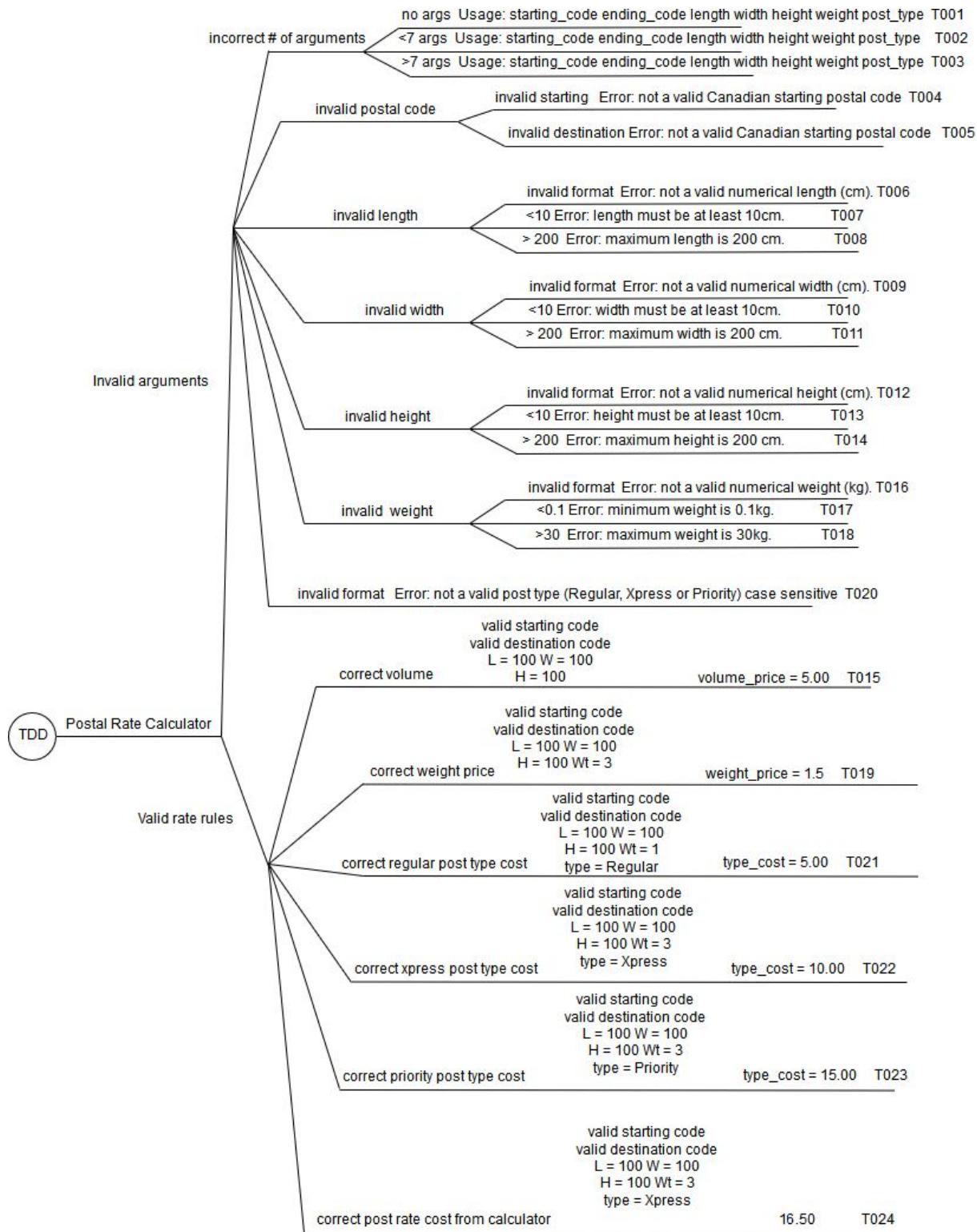
Note that this is developed in Python, which is a scripting language. Thus, executing this application needs to have python 3.6 installed like running any other scripts on a command line. The aforementioned .csv file must be in the same directory as the script. The sha-bang is `#!/usr/bin/env python3.6`. The following screenshot is an example of an executable of the resulting application:



A screenshot of a macOS terminal window titled "postal-rates-calculator-can — -bash — 80x24". The window shows the following command-line session:

```
Last login: Sun Mar  4 13:43:03 on ttys000
[Victors-MBP:~ VictorVuong$ cd GitHub/postal-rates-calculator-can
[Victors-MBP:postal-rates-calculator-can VictorVuong$ ./postal_calc.py h4k2g2 v9g
8r7 100 100 100 1 Xpress
The total cost is: 15.50
Victors-MBP:postal-rates-calculator-can VictorVuong$ ]
```

The following diagram is a visual representation of all test cases throughout the TDD development



List of tests

Test 1: No arguments
Test 2: Too few arguments
Test 3: Too many arguments
Test 4: Invalid starting postal code
Test 5: Invalid destination postal code
Test 6: Invalid length format
Test 7: Length too short
Test 8: Length too long
Test 9: Invalid width format
Test 10: Width too short
Test 11: Width too long
Test 12: Invalid height format
Test 13: Height too short
Test 14: Height too big
Test 15: Correct volume price
Test 16: Invalid weight format
Test 17: Weight too low
Test 18: Weight too high
Test 19: Correct weight price
Test 20: Invalid post-type
Test 21: Correct Regular post-type cost
Test 22: Correct Xpress post-type cost
Test 23: Correct Priority post-type cost
Test 24: Correct post rate cost from calculator (program)

Development Process:

For each step of the development processes, no code will be written until a written test fails, unless the code is written to make a failing unit test. New code can be only written when a unit test is proven to fail and that code will only serve the purpose of making the new unit test pass. Lastly, refactoring may occur, but all prior tests must pass before creating other failing unit tests.

Screenshots and Implementations

Every test below is a step towards the final postal rate calculator program

The TDD process is shown in this order for each test:

- 1- Description of the test (purposes, parameters, syntax and notes)
- 2- Screenshot of the failing test
- 3- Screenshot of the current code making the test fail
- 4- Screenshot of the succeeding test
- 5- Screenshot of the current code making the test succeed (with changes)

Test 1: No arguments

Purpose of the test: Make sure that the user inputs arguments to run the program

Test Name: test_no_args

Call setup: postal_calc.py

Expected Result: Usage: starting_code ending_code length width height weight post_type

Note: Handles case when user thinks the function doesn't take any arguments; therefore, it is assumed that this test is different from the following one (too few arguments) and is not redundant.

Test 1 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains a single failing test case `test_no_args`. The bottom window shows the `Test Results` tool window, indicating 1 test failed. The terminal output shows the assertion error and the command used to run the test.

```
import pytest
import postal_calc

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

test.py:9: AssertionError
=====
 FAILURES =====
test_no_args

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
E     AssertionError: assert None == 'Usage: starting_code ending_code length width height weight post_type'

test.py:9: AssertionError
=====
 1 failed in 0.04 seconds =====
Process finished with exit code 0
```

Test 1 Code:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which contains the `main` function. The bottom window shows the `Test Results` tool window, indicating 1 test failed. The terminal output shows the assertion error and the command used to run the test.

```
import sys
def main(argc, argv):
    return None

main()
```

```
import pytest
import postal_calc

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

test.py:9: AssertionError
=====
 FAILURES =====
test_no_args

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
E     AssertionError: assert None == 'Usage: starting_code ending_code length width height weight post_type'

test.py:9: AssertionError
=====
 1 failed in 0.04 seconds =====
Process finished with exit code 0
```

Test 1 Success:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains a single test function `test_no_args`. The bottom window shows the `Run` tool window with the output of the pytest run. The output indicates 1 test passed in 0.01 seconds. The command used was `py.test in test.py`.

```
import pytest
import postal_calc

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
```

```
Testing started at 12:16 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can/test.py
Launching py.test with arguments /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can/test.py in /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can, inifile: test.py .
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can, inifile: test.py .
collected 1 item [100%]
test.py .                                         [100%]
=====
1 passed in 0.01 seconds
Process finished with exit code 0
```

Test 1 Code with changes:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include a `main` function. The bottom window shows the `Run` tool window with the output of the pytest run. The output indicates 1 test passed in 0.01 seconds. The command used was `py.test in test.py`.

```
import sys

def main(argc, argv):
    display_text = "Usage: starting_code ending_code length width height weight post_type"
    return display_text
```

```
Testing started at 12:16 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can/test.py
Launching py.test with arguments /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can/test.py in /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can, inifile: test.py .
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/PycharmProjects/postal-rates-calculator-can, inifile: test.py .
collected 1 item [100%]
test.py .                                         [100%]
=====
1 passed in 0.01 seconds
Process finished with exit code 0
```

Test 2: Too few arguments

Purpose of the test: Make sure that the number of arguments that the user inputs is not less than 7

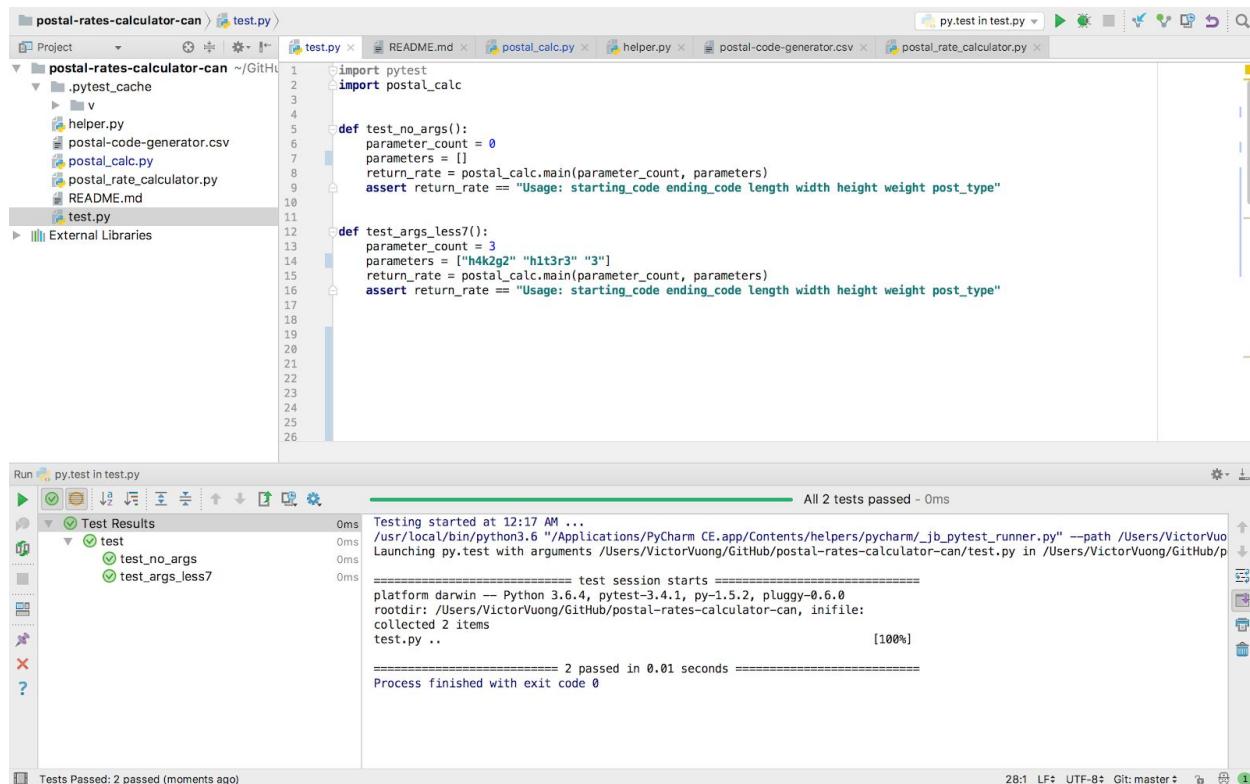
Test Name: test_args_less7

Call setup: postal_calc.py h4k2g2 h1th3r3 3

Expected Result: Usage: starting_code ending_code length width height weight post_type

Note: this test passes immediately, because prior TDD code outputs the expected result already. Thus, no additional code is required in the main.

Test 2 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains two test functions: `test_no_args()` and `test_args_less7()`. The bottom window shows the "Run" tool window with the output of the pytest run. The output indicates that all 2 tests passed in 0ms. The log shows the test session starting, platform details (darwin, Python 3.6.4), and collecting 2 items (test.py). It also shows the 2 tests passed in 0.01 seconds.

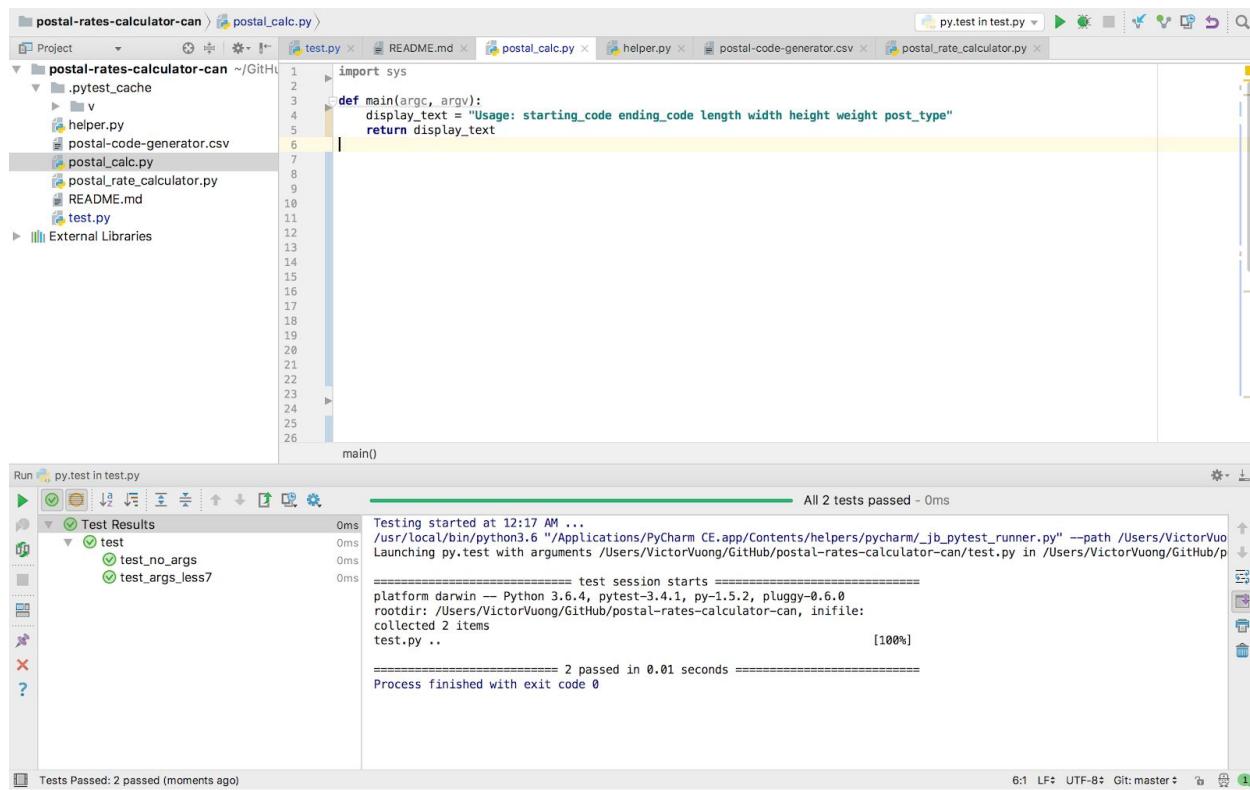
```
import pytest
import postal_calc

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_less7():
    parameter_count = 3
    parameters = ["h4k2g2", "h1t3r3", "3"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
```

```
All 2 tests passed - 0ms
Testing started at 12:17 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 2 items
test.py .. [100%]
=====
2 passed in 0.01 seconds
Process finished with exit code 0
```

Test 2 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, specifically the `main` function which returns a usage string. The bottom window shows the "Run" tool window with the output of the pytest run. The output indicates that all 2 tests passed in 0ms. The log shows the test session starting, platform details (darwin, Python 3.6.4), and collecting 2 items (test.py). It also shows the 2 tests passed in 0.01 seconds.

```
import sys

def main(argc, argv):
    display_text = "Usage: starting_code ending_code length width height weight post_type"
    return display_text
```

```
All 2 tests passed - 0ms
Testing started at 12:17 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 2 items
test.py .. [100%]
=====
2 passed in 0.01 seconds
Process finished with exit code 0
```

Test 3: Too many arguments

Purpose of the test: Make sure that the number of arguments inputted by the user is not more than 7

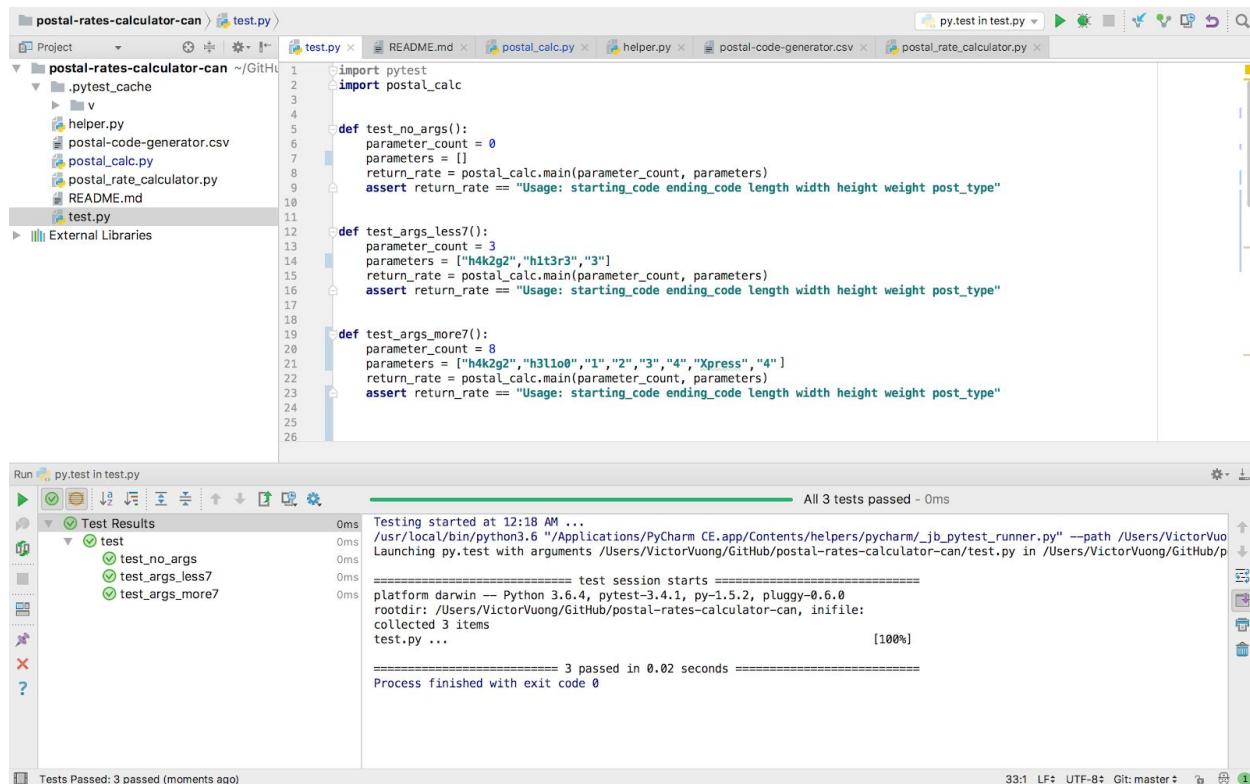
Test Name: test_args_more7

Call setup: postal_calc.py h4k2g2 h3l1o0 1 2 3 4 Xpress 4

Expected Result: Usage: starting_code ending_code length width height weight post_type

Note: this test passes immediately, because prior TDD code outputs the desired code already. Thus, no additional code is required in the main.

Test 3 Succeed:



The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'test.py' file is selected and contains three test functions: 'test_no_args', 'test_args_less7', and 'test_args_more7'. The 'Run' tool window at the bottom shows the output of the pytest run, indicating 'All 3 tests passed - 0ms'. The terminal pane displays the test session log, including the command used ('py.test in test.py'), the platform ('platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0'), and the collected items ('collected 3 items'). The log also shows the test results ('3 passed in 0.02 seconds') and the process finishing with exit code 0.

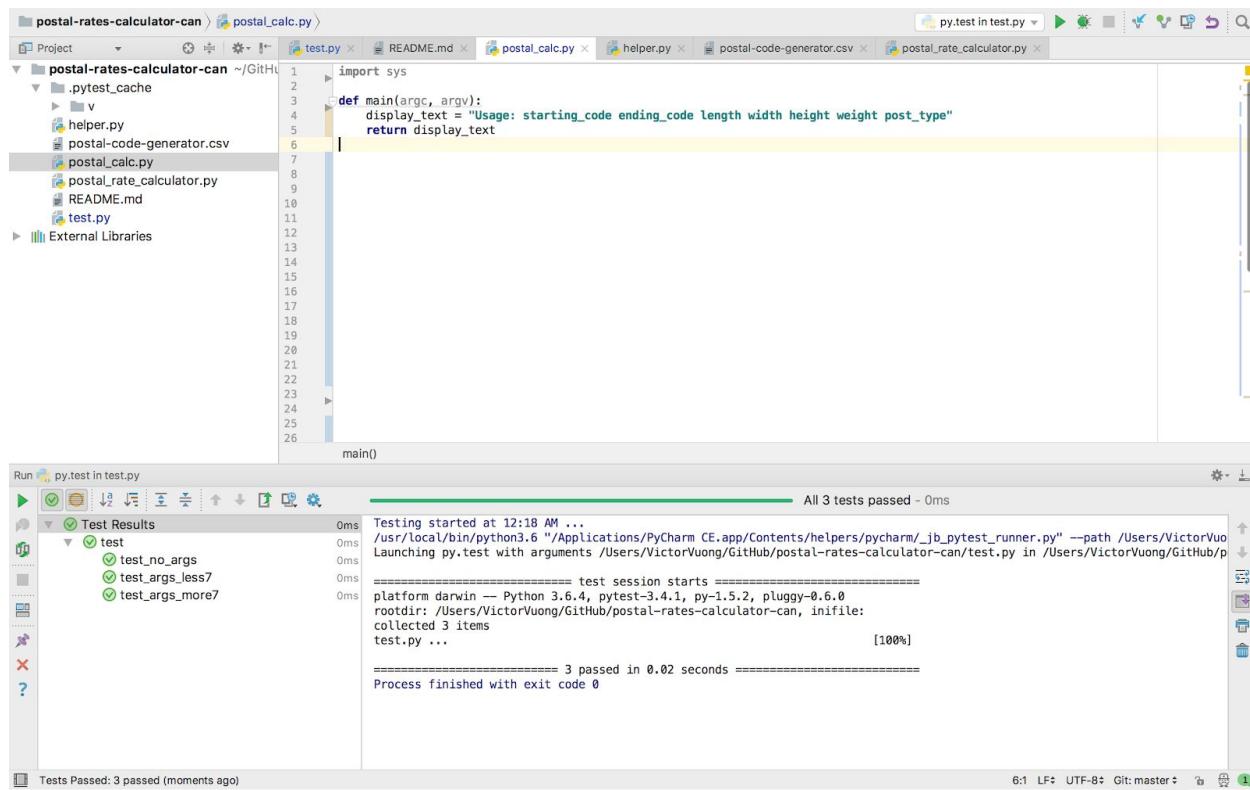
```
import pytest
import postal_calc

def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_less7():
    parameter_count = 3
    parameters = ['h4k2g2', 'h1t3r3', '3']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_more7():
    parameter_count = 8
    parameters = ['h4k2g2', 'h3l1o0', '1', '2', '3', '4', 'Xpress', '4']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
```

Test 3 Code with changes:



The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'postal_calc.py' file is selected and contains a single function 'main'. The 'Run' tool window at the bottom shows the output of the pytest run, indicating 'All 3 tests passed - 0ms'. The terminal pane displays the test session log, including the command used ('py.test in test.py'), the platform ('platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0'), and the collected items ('collected 3 items'). The log also shows the test results ('3 passed in 0.02 seconds') and the process finishing with exit code 0.

```
import sys

def main(argc, argv):
    display_text = "Usage: starting_code ending_code length width height weight post_type"
    return display_text
```

Test 4: Invalid starting postal code

Purpose of the test: Make sure that the starting postal code is a valid existing canadian postal code

Test Name: test_invalid_startcode

Call setup: postal_calc.py 123456 654321 1 2 3 4 Xpress

Expected Result: Error: not a valid canadian starting postal code.

Note: The validity of the postal code is dependent on the .csv file as aforementioned in the introduction.

As this is taken from a lookup table, it is not necessary to check for the length or format of the first argument.

Test 4 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the `test.py` file with several test cases for a postal rate calculator. The bottom window shows the `Test Results` tool window, which indicates 4 tests done, 1 failed, and a duration of 3ms. The failed test is `test_invalid_startcode`, which asserts that the function returns an error message for invalid starting postal codes.

```
return_rate = postal_calc.main(parameter_count, parameters)
assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_less7():
    parameter_count = 3
    parameters = ["h4k2g2" "h1t3r3" "3"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_more7():
    parameter_count = 8
    parameters = ["h4k2g2" "h3l1o0" "1" "2" "3" "4" "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_invalid_startcode():
    parameter_count = 7
    parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."
```

==== FAILURES =====
_____ test_invalid_startcode _____
def test_invalid_startcode():
 parameter_count = 7
 parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
 return_rate = postal_calc.main(parameter_count, parameters)
 assert return_rate == "Error: not a valid canadian starting postal code."
E AssertionError: assert 'Usage: start..ght post_type' == 'Error: not a ... postal code.'
E - Usage: starting_code ending_code length width height weight post_type
E + Error: not a valid canadian starting postal code.

test.py:30: AssertionError
===== 1 failed, 3 passed in 0.07 seconds ======

Test 4 Code:

The screenshot shows the PyCharm IDE interface. The top window displays the `postal_calc.py` file, which contains the `main` function. The bottom window shows the `Test Results` tool window, which indicates 4 tests done, 1 failed, and a duration of 3ms. The failed test is `test_invalid_startcode`, which asserts that the function returns an error message for invalid starting postal codes.

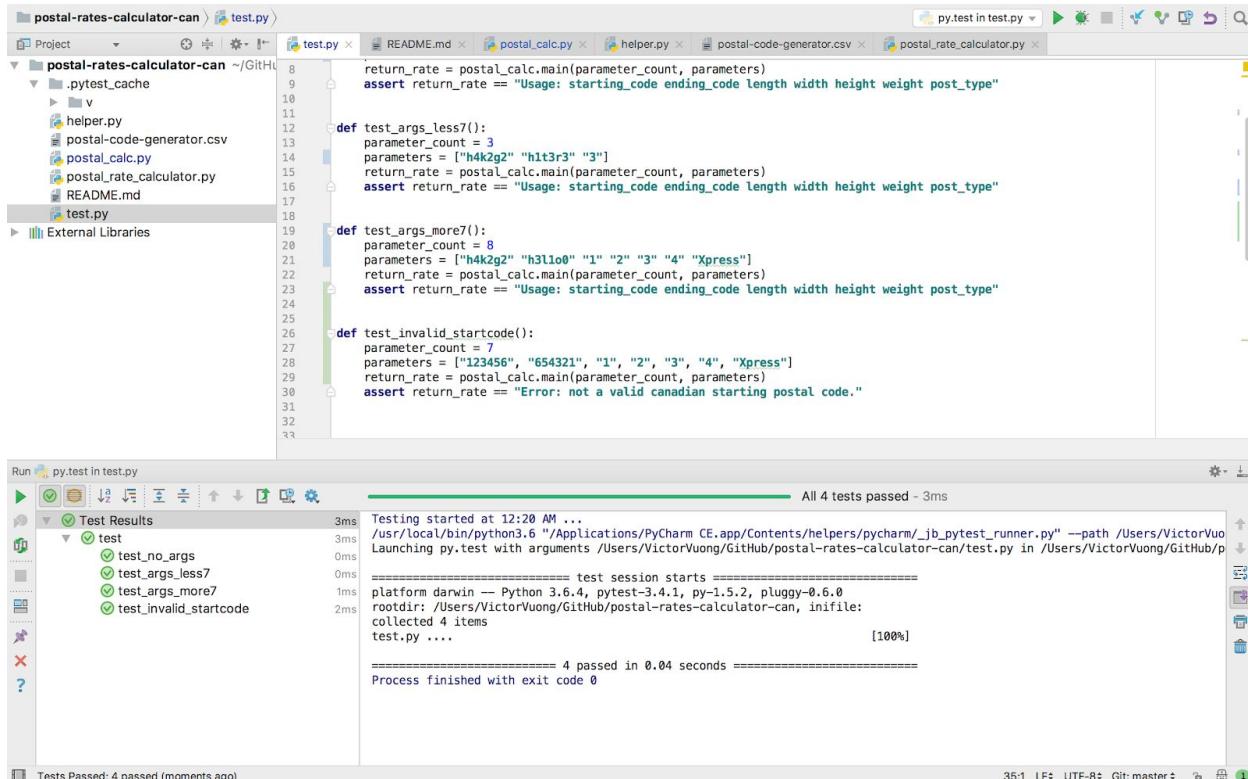
```
import sys

def main(argc, argv):
    display_text = "Usage: starting_code ending_code length width height weight post_type"
    return display_text
```

==== FAILURES =====
_____ test_invalid_startcode _____
def test_invalid_startcode():
 parameter_count = 7
 parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
 return_rate = postal_calc.main(parameter_count, parameters)
 assert return_rate == "Error: not a valid canadian starting postal code."
E AssertionError: assert 'Usage: start..ght post_type' == 'Error: not a ... postal code.'
E - Usage: starting_code ending_code length width height weight post_type
E + Error: not a valid canadian starting postal code.

test.py:30: AssertionError
===== 1 failed, 3 passed in 0.07 seconds ======

Test 4 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for a postal rate calculator. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 4 tests passed in 0.04 seconds.

```
def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_less7():
    parameter_count = 3
    parameters = ['h4k2g2', 'h1t3r3', '3']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_more7():
    parameter_count = 8
    parameters = ['h4k2g2', 'h3l1o0', '1', '2', '3', '4', 'Xpress']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

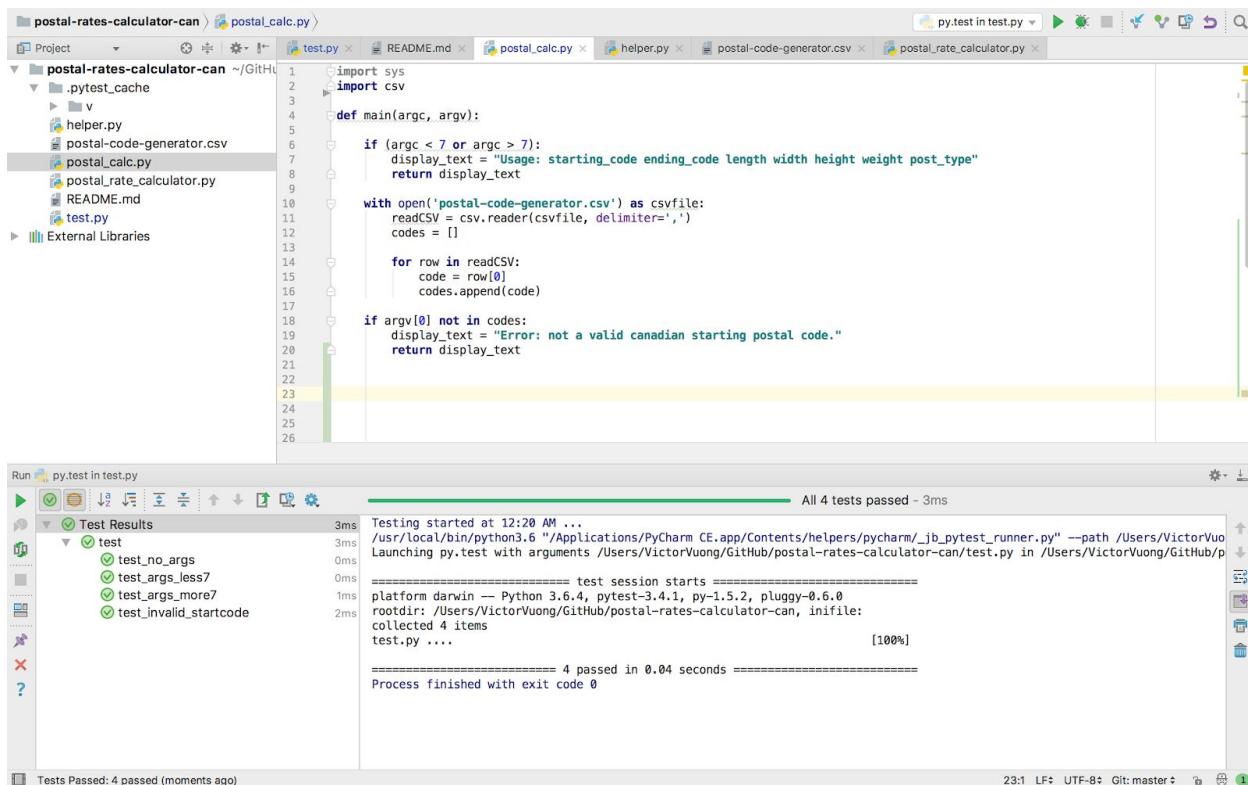
def test_invalid_startcode():
    parameter_count = 7
    parameters = ['123456', '654321', '1', '2', '3', '4', 'Xpress']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode

All 4 tests passed - 3ms

Test 4 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to handle command-line arguments and read from a CSV file. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 4 tests passed in 0.04 seconds.

```
import sys
import csv

def main(argv, argc):
    if (argc < 7 or argc > 7):
        display_text = "Usage: starting_code ending_code length width height weight post_type"
        return display_text

    with open('postal-code-generator.csv') as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        codes = []

        for row in readCSV:
            code = row[0]
            codes.append(code)

    if argv[0] not in codes:
        display_text = "Error: not a valid canadian starting postal code."
        return display_text
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode

All 4 tests passed - 3ms

Test 5: Invalid destination postal code

Purpose of the test: Make sure that the destination postal code is a valid existing canadian postal code

Test Name: test_invalid_destinationcode

Call setup: postal_calc.py h4k2g2 654321 1 2 3 4 Xpress

Expected Result: Error: not a valid canadian destination postal code.

Test 5 Fail:

The screenshot shows the PyCharm IDE interface with the project 'postal-rates-calculator-can' open. The code editor displays a test file 'test.py' containing several test cases for validating postal codes. The 'Run' tool window at the bottom shows the results of running 'py.test in test.py'. It indicates 5 tests done, 1 failed, and a duration of 10ms. The failed test is 'test_invalid_destinationcode', which failed with an AssertionError. The error message shows the assertion 'assert return_rate == "Error: not a valid canadian destination postal code."' failing because the actual return value was None.

```
parameter_count = 7
parameters = ["h4k2g2", "h1t3r3", "3"]
return_rate = postal_calc.main(parameter_count, parameters)
assert return_rate == "Usage: starting_code ending_code length width height post_type"

def test_args_more7():
    parameter_count = 8
    parameters = ["h4k2g2", "h3l1o0", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height post_type"

def test_invalid_startcode():
    parameter_count = 7
    parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."

def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian destination postal code."
```

Test Results

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 0ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 1ms |
| test_invalid_destinationcode | Failed | 9ms |

```
===== FAILURES =====
test_invalid_destinationcode
=====
def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid canadian destination postal code."
E     AssertionError: assert None == 'Error: not a valid canadian destination postal code.'
```

test.py:37: AssertionError

```
===== 1 failed, 4 passed in 0.09 seconds =====
```

Test 5 Code:

The screenshot shows the PyCharm IDE interface with the project 'postal-rates-calculator-can' open. The code editor displays the main calculation logic in 'postal_calc.py'. The 'Run' tool window at the bottom shows the results of running 'py.test in test.py'. It indicates 5 tests done, 1 failed, and a duration of 10ms. The failed test is 'test_invalid_destinationcode', which failed with an AssertionError. The error message shows the assertion 'assert return_rate == "Error: not a valid canadian destination postal code."' failing because the actual return value was None.

```
import sys
import csv

def main(argc, argv):
    if (argc < 7 or argc > 7):
        display_text = "Usage: starting_code ending_code length width height post_type"
        return display_text

    with open('postal-code-generator.csv') as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        codes = []

        for row in readCSV:
            code = row[0]
            codes.append(code)

    if argv[0] not in codes:
        display_text = "Error: not a valid canadian starting postal code."
    return display_text
```

Test Results

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 0ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 1ms |
| test_invalid_destinationcode | Failed | 9ms |

```
===== FAILURES =====
test_invalid_destinationcode
=====
def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid canadian destination postal code."
E     AssertionError: assert None == 'Error: not a valid canadian destination postal code.'
```

test.py:37: AssertionError

```
===== 1 failed, 4 passed in 0.09 seconds =====
```

Test 5 Succeed:

Project: postal-rates-calculator-can

File: test.py

```
parameter_count = 3
parameters = ["h4k2g2" "h1t3r3" "3"]
return_rate = postal_calc.main(parameter_count, parameters)
assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_args_more7():
    parameter_count = 8
    parameters = ["h4k2g2" "h3l1o0" "1" "2" "3" "4" "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Usage: starting_code ending_code length width height weight post_type"

def test_invalid_startcode():
    parameter_count = 7
    parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."

def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian destination postal code."
```

Run: py.test in test.py

Test Results: All 5 tests passed - 2ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode

Tests Passed: 5 passed (moments ago)

Test 5 Code with changes:

Project: postal-rates-calculator-can

File: postal_calc.py

```
def main(argc, argv):
    if (argc < 7 or argc > 7):
        display_text = "Usage: starting_code ending_code length width height weight post_type"
        return display_text

    with open('postal-code-generator.csv') as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        codes = []

        for row in readCSV:
            code = row[0]
            codes.append(code)

    if argv[0] not in codes:
        display_text = "Error: not a valid canadian starting postal code."
        return display_text

    if argv[1] not in codes:
        display_text = "Error: not a valid canadian destination postal code."
        return display_text
```

Run: py.test in test.py

Test Results: All 5 tests passed - 2ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode

Tests Passed: 5 passed (moments ago)

Test 6: Invalid length format

Purpose of the test: Make sure that the user inputs a numerical value for the length

Test Name: test_invalid_length_format

Call setup: postal_calc.py h4k2g2 v9g8r7 abc 2 3 4 Xpress

Expected Result: Error: not a valid numerical length (cm).

Note: It was decided that the input format for the length of the parcel is in decimal format.

Test 6 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for postal code validation. One specific test, `test_invalid_length_format()`, has failed with an `AssertionError`. The bottom window shows the `Run` tool window with the output of the test run, indicating 1 failed test out of 6.

```
def test_invalid_startcode():
    parameter_count = 7
    parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."

def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian destination postal code."

def test_invalid_length_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical length (cm)."
```

Run py.test in test.py
6 tests done: 1 failed - 8ms

Test Results

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format

===== FAILURES =====

test_invalid_length_format

```
def test_invalid_length_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical length (cm)."
E     AssertionError: assert None == 'Error: not a valid numerical length (cm).'

test.py:44: AssertionError
===== 1 failed, 5 passed in 0.09 seconds =====
```

Process finished with exit code 0

Test 6 Code:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which contains logic for parsing command-line arguments and reading from a CSV file to validate postal codes. The bottom window shows the `Run` tool window with the output of the test run, indicating 1 failed test out of 6.

```
if (argc < 7 or argc > 7):
    display_text = "Usage: starting_code ending_code length width height weight post_type"
    return display_text

with open('postal-code-generator.csv') as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    codes = []

    for row in readCSV:
        code = row[0]
        codes.append(code)

if argv[0] not in codes:
    display_text = "Error: not a valid canadian starting postal code."
    return display_text

if argv[1] not in codes:
    display_text = "Error: not a valid canadian destination postal code."
    return display_text
```

Run py.test in test.py
6 tests done: 1 failed - 8ms

Test Results

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format

===== FAILURES =====

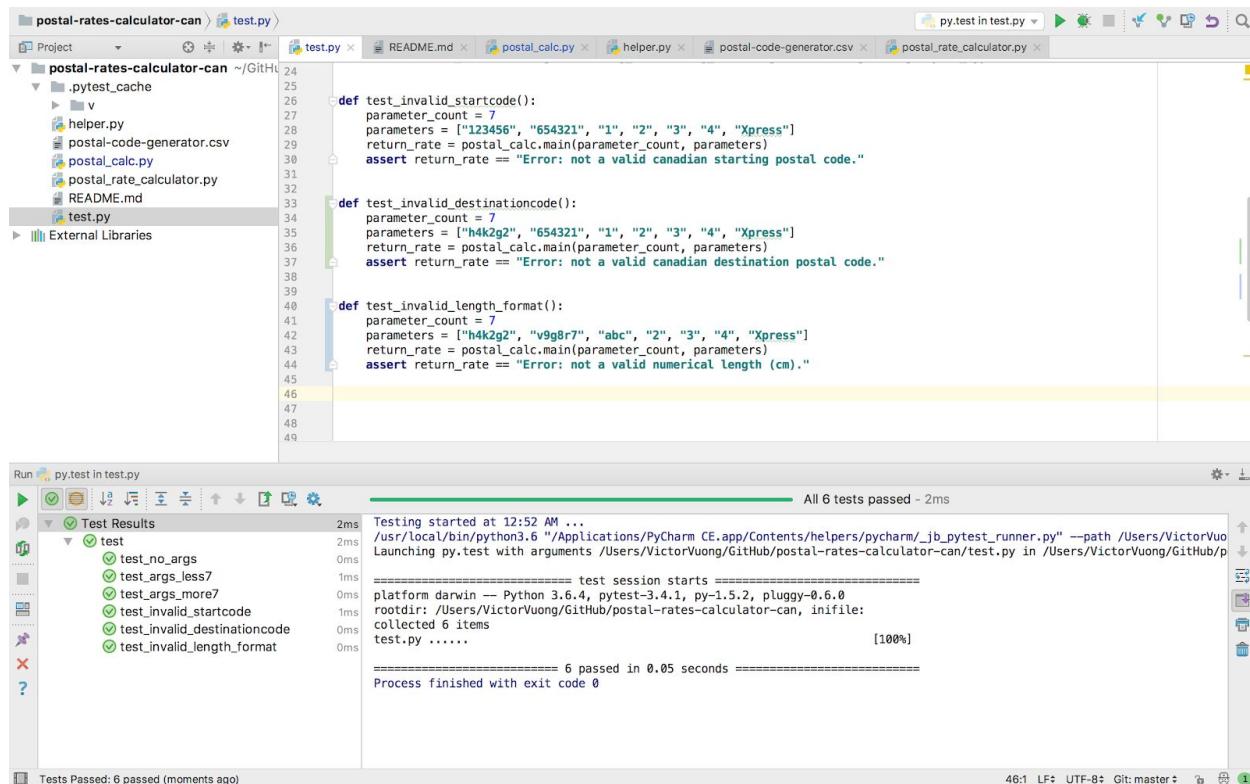
test_invalid_length_format

```
def test_invalid_length_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical length (cm)."
E     AssertionError: assert None == 'Error: not a valid numerical length (cm).'

test.py:44: AssertionError
===== 1 failed, 5 passed in 0.09 seconds =====
```

Process finished with exit code 0

Test 6 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains three test functions: `test_invalid_startcode`, `test_invalid_destinationcode`, and `test_invalid_length_format`. The bottom window shows the `Run` tool window with the output of the pytest run. The output indicates "All 6 tests passed - 2ms". The test results list shows six green checkmarks, each corresponding to one of the three test functions defined in `test.py`.

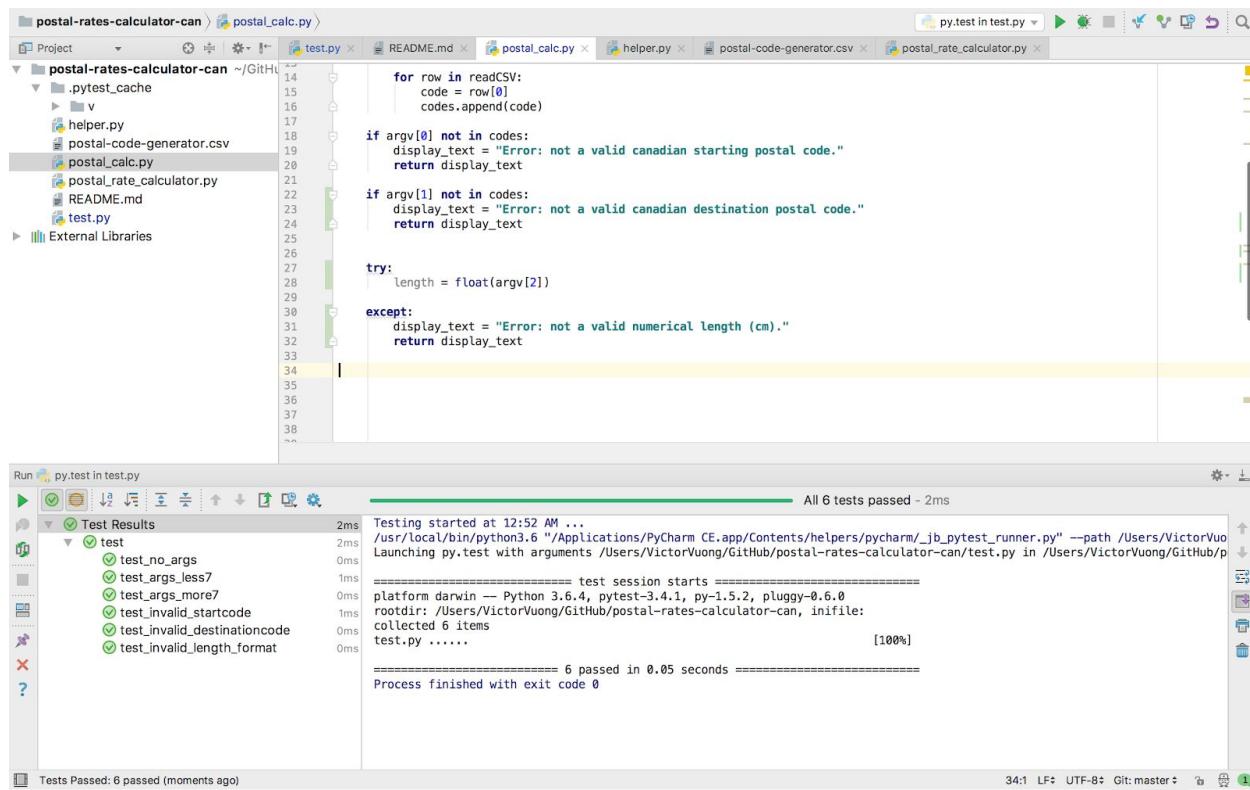
```
def test_invalid_startcode():
    parameter_count = 7
    parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."

def test_invalid_destinationcode():
    parameter_count = 7
    parameters = ["H4K2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian destination postal code."

def test_invalid_length_format():
    parameter_count = 7
    parameters = ["H4K2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical length (cm)."
```

```
All 6 tests passed - 2ms
Testing started at 12:52 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile: test.py .....
collected 6 items [100%]
=====
6 passed in 0.05 seconds
Process finished with exit code 0
```

Test 6 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include error handling for invalid input. The bottom window shows the `Run` tool window with the output of the pytest run. The output indicates "All 6 tests passed - 2ms". The test results list shows six green checkmarks, each corresponding to one of the three test functions defined in `test.py`.

```
for row in readCSV:
    code = row[0]
    codes.append(code)

if argv[0] not in codes:
    display_text = "Error: not a valid canadian starting postal code."
    return display_text

if argv[1] not in codes:
    display_text = "Error: not a valid canadian destination postal code."
    return display_text

try:
    length = float(argv[2])
except:
    display_text = "Error: not a valid numerical length (cm)."
    return display_text
```

```
All 6 tests passed - 2ms
Testing started at 12:52 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
=====
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile: test.py .....
collected 6 items [100%]
=====
6 passed in 0.05 seconds
Process finished with exit code 0
```

Test 7: Length too short

Purpose of the test: Make sure that the user inputs a length that is not less than the minimum value accepted (10cm)

Test Name: test_length_less10

Call setup: postal_calc.py h4k2g2 v9g8r7 -5 2 3 4 Xpress

Expected Result: Error: length must be at least 10cm.

Note: It was decided that the minimum length of a parcel is 10 cm for it to be eligible for shipping. This test also covers cases when the user inputs a negative number.

Test 7 Fail:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'test.py' file is selected in the left navigation bar. The code editor displays several test functions: `test_invalid_destinationcode()`, `test_invalid_length_format()`, and `test_length_less10()`. The `test_length_less10()` function is highlighted with a yellow selection bar. The run tool window at the bottom shows the test results: 7 tests done, 1 failed. The failed test is `test_length_less10` with an `AssertionError`. The error message is: `assert return_rate == "Error: length must be at least 10cm."`. The stack trace shows the code execution path from the test function down to the assertion.

Test 7 Code:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'postal_rate_calculator.py' file is selected in the left navigation bar. The code editor displays the implementation of the `test_length_less10()` function. The run tool window at the bottom shows the test results: 7 tests done, 1 failed. The failed test is `test_length_less10` with an `AssertionError`. The error message is: `assert return_rate == "Error: length must be at least 10cm."`. The stack trace shows the code execution path from the test function down to the assertion.

Test 7 Succeed:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `test.py`, which contains several test cases for a postal rate calculator. The bottom part shows the `Run` tool window with the output of the pytest run, indicating that all 7 tests passed in 0.04 seconds.

```
def test_no_args():
    parameter_count = 0
    parameters = []
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian starting postal code."
```

```
def test_args_less7():
    parameter_count = 7
    parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid canadian destination postal code."
```

```
def test_args_more7():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical length (cm)."
```

```
def test_length_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "-5", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: length must be at least 10cm."
```

Run py.test in test.py
All 7 tests passed - 2ms

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10

Testing started at 1:20 PM ...
Launching py.test with arguments /Users/VictorVuong/Github/postal-rates-calculator-can/test.py in /Users/VictorVuong/Github/postal-rates-calculator-can, inifile: test.py [100%]
===== 7 passed in 0.04 seconds =====
Process finished with exit code 0

Test 7 Code with changes:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `postal_calc.py`, which has been modified to handle command-line arguments and validate input. The bottom part shows the `Run` tool window with the output of the pytest run, indicating that all 7 tests passed in 0.04 seconds.

```
if argv[1] not in codes:
    display_text = "Error: not a valid canadian starting postal code."
    return display_text
```

```
try:
    length = float(argv[2])
except:
    display_text = "Error: not a valid numerical length (cm)."
    return display_text
```

```
if length < 10:
    display_text = "Error: length must be at least 10cm."
    return display_text
```

Run py.test in test.py
All 7 tests passed - 2ms

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10

Testing started at 1:20 PM ...
Launching py.test with arguments /Users/VictorVuong/Github/postal-rates-calculator-can/test.py in /Users/VictorVuong/Github/postal-rates-calculator-can, inifile: test.py [100%]
===== 7 passed in 0.04 seconds =====
Process finished with exit code 0

Test 8: Length too long

Purpose of the test: Make sure that the user inputs a length that is not more than the maximum value accepted (200cm)

Test Name: test_length_more200

Call setup: postal_calc.py h4k2g2 v9g8r7 244 2 3 4 Xpress

Expected Result: Error: maximum length is 200cm.

Note: It was decided that the range of valid length inputs would be extended up to 200 cm.

Test 8 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for length validation. The bottom window shows the `Test Results` tool window with the following output:

```
8 tests done: 1 failed - 4ms
=====
FAILURES =====
test_length_more200
=====
test.py:58: AssertionError
[100%]
=====
1 failed, 7 passed in 0.09 seconds =====
Process finished with exit code 0
```

Tests Failed: 7 passed, 1 failed (moments ago)

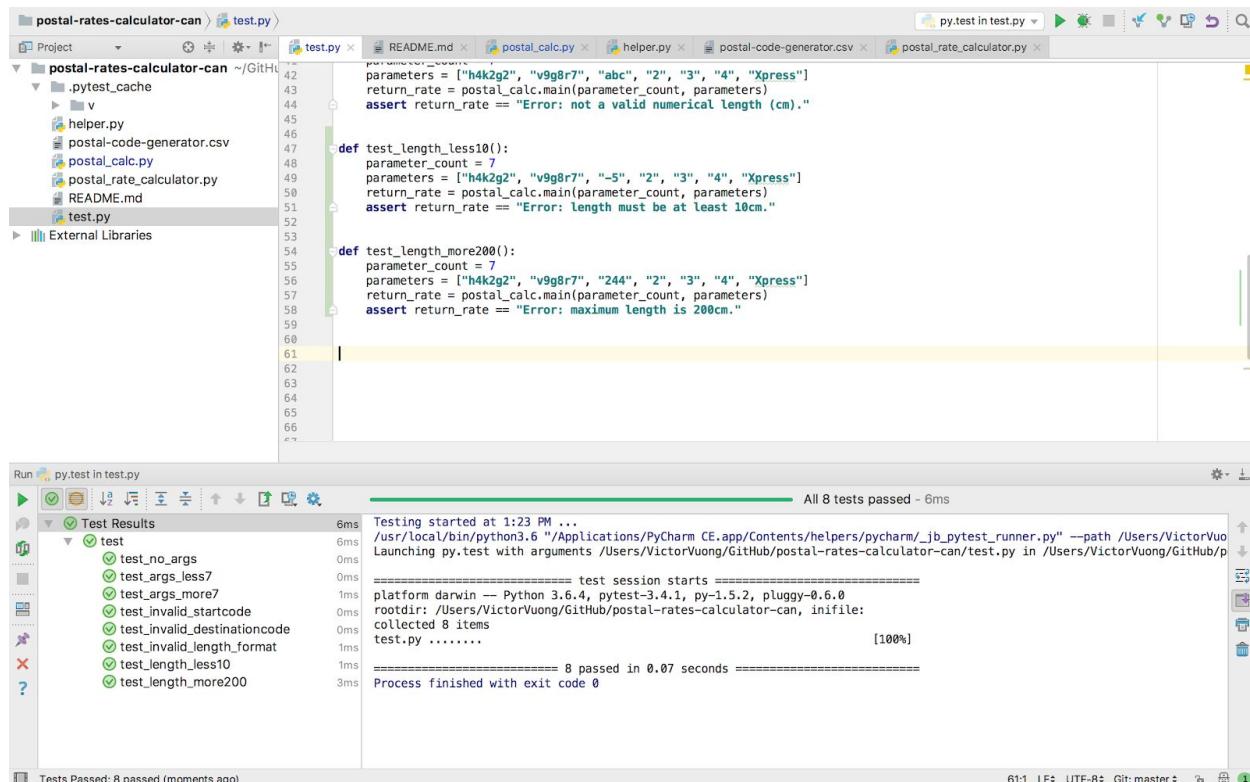
Test 8 Code:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which includes logic for handling invalid length inputs. The bottom window shows the `Test Results` tool window with the following output:

```
8 tests done: 1 failed - 4ms
=====
FAILURES =====
test_length_more200
=====
test.py:58: AssertionError
[100%]
=====
1 failed, 7 passed in 0.09 seconds =====
Process finished with exit code 0
```

Tests Failed: 7 passed, 1 failed (moments ago)

Test 8 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for a postal rate calculator. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 8 tests passed in 0.07 seconds.

```
def test_length_less10():
    parameter_count = 7
    parameters = ['h4k2g2', 'v9g8r7', '-5', '2', '3', '4', 'Xpress']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: length must be at least 10cm."
```

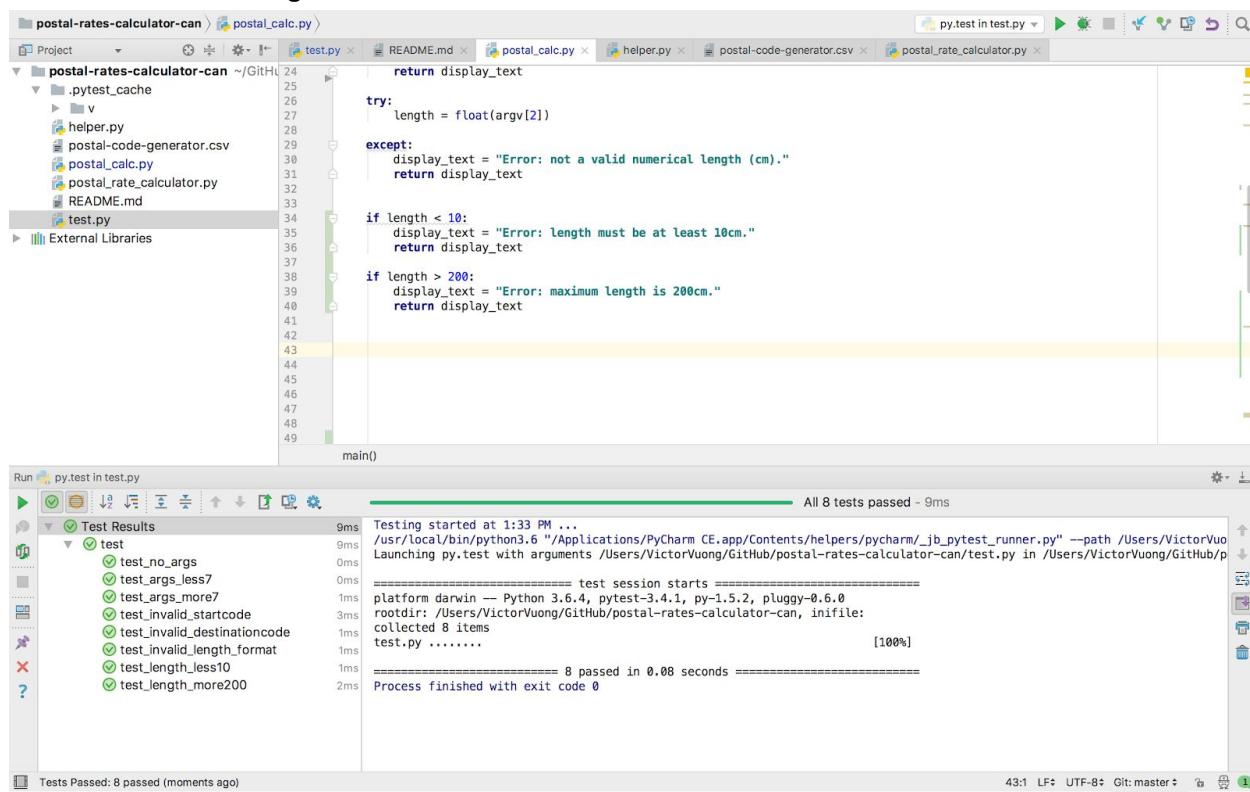
```
def test_length_more200():
    parameter_count = 7
    parameters = ['h4k2g2', 'v9g8r7', '244', '2', '3', '4', 'Xpress']
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum length is 200cm."
```

Test Results: All 8 tests passed - 6ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200

Testing started at 1:23 PM ...
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 8 items
test.py [100%]
===== 8 passed in 0.07 seconds ======

Test 8 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include validation for length. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 8 tests passed in 0.08 seconds.

```
def main():
    if len(sys.argv) < 2:
        display_text = "Usage: postal_calc.py length"
        return display_text

    try:
        length = float(argv[2])
    except:
        display_text = "Error: not a valid numerical length (cm)."
        return display_text

    if length < 10:
        display_text = "Error: length must be at least 10cm."
        return display_text

    if length > 200:
        display_text = "Error: maximum length is 200cm.."
        return display_text
```

Test Results: All 8 tests passed - 9ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200

Testing started at 1:33 PM ...
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 8 items
test.py [100%]
===== 8 passed in 0.08 seconds ======

Test 9: Invalid width format

Purpose of the test: Make sure that the user inputs a numerical value for the width

Test Name: test_invalid_width_format

Call setup: postal_calc.py h4k2g2 v9g8r7 50 abc 3 4 Xpress

Expected Result: Error: not a valid numerical width (cm).

Note: It was decided that the input format for the width of the parcel is in decimal format.

Test 9 Fail:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `test.py` with three failing test cases: `test_length_less10`, `test_length_more200`, and `test_invalid_width_format`. The bottom part shows the `Test Results` tool window with the failure details for `test_invalid_width_format`.

```
def test_length_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "-5", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: length must be at least 10cm."

def test_length_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "244", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum length is 200cm."

def test_invalid_width_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical width (cm.)"
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format

===== FAILURES =====

```
def test_invalid_width_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical width (cm.)"
E     AssertionError: assert None == 'Error: not a valid numerical width (cm.)'

test.py:65: AssertionError
===== 1 failed, 8 passed in 0.11 seconds =====
```

Test 9 Code:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `postal_calc.py` with logic for handling invalid width input. The bottom part shows the `Test Results` tool window with the failure details for `test_invalid_width_format`.

```
display_text = "Error: not a valid numerical destination postal code."
return display_text

try:
    length = float(argv[2])
except:
    display_text = "Error: not a valid numerical length (cm.)."
    return display_text

if length < 10:
    display_text = "Error: length must be at least 10cm."
    return display_text

if length > 240:
    display_text = "Error: maximum length is 200cm."
    return display_text
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format

===== FAILURES =====

```
def test_invalid_width_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical width (cm.)"
E     AssertionError: assert None == 'Error: not a valid numerical width (cm.)'

test.py:65: AssertionError
===== 1 failed, 8 passed in 0.11 seconds =====
```

Test 9 Succeed:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `test.py` with three test cases: `test_length_less10()`, `test_length_more200()`, and `test_invalid_width_format()`. The bottom part shows the test results in the "Run" tool window, indicating "All 9 tests passed - 10ms". The test results list includes: `test_no_args`, `test_args_less7`, `test_args_more7`, `test_invalid_startcode`, `test_invalid_destinationcode`, `test_invalid_length_format`, `test_length_less10`, `test_length_more200`, and `test_invalid_width_format`.

Test 9 Code with changes:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `postal_calc.py` with a modified `main()` function. The bottom part shows the test results in the "Run" tool window, indicating "All 9 tests passed - 10ms". The test results list includes: `test_no_args`, `test_args_less7`, `test_args_more7`, `test_invalid_startcode`, `test_invalid_destinationcode`, `test_invalid_length_format`, `test_length_less10`, `test_length_more200`, and `test_invalid_width_format`.

Test 10: Width too short

Purpose of the test: Make sure that the user inputs a width that is not less than the minimum value accepted (10cm)

Test Name: test_width_less10

Call setup: postal_calc.py h4k2g2 v9g8r7 50 2 3 4 Xpress

Expected Result: Error: width must be at least 10cm.

Note: It was decided that the minimum width required for the parcel to be eligible for shipping is 10 cm.

Test 10 Fail:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'test.py' file is selected in the editor. The code contains three test functions: `test_length_more200()`, `test_invalid_width_format()`, and `test_width_less10()`. The `test_width_less10()` function fails with an `AssertionError`.

```
def test_width_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: width must be at least 10cm."
    E AssertionError: assert None == 'Error: width must be at least 10cm.'
```

In the 'Run' tool window, the test results show 10 tests done, 1 failed, and 11ms execution time. The failed test is `test_width_less10` with an `AssertionError`.

Test Results:

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10

10 tests done: 1 failed - 11ms

Test 10 Code:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'postal_calc.py' file is selected in the editor. The code contains a `main()` function with validation logic for length and width.

```
def main(parameter_count, parameters):
    if length > 240:
        display_text = "Error: length must be at least 200cm."
        return display_text
    if length < 10:
        display_text = "Error: maximum length is 200cm."
        return display_text
    try:
        width = float(argv[3])
    except:
        display_text = "Error: not a valid numerical width (cm)."
        return display_text
```

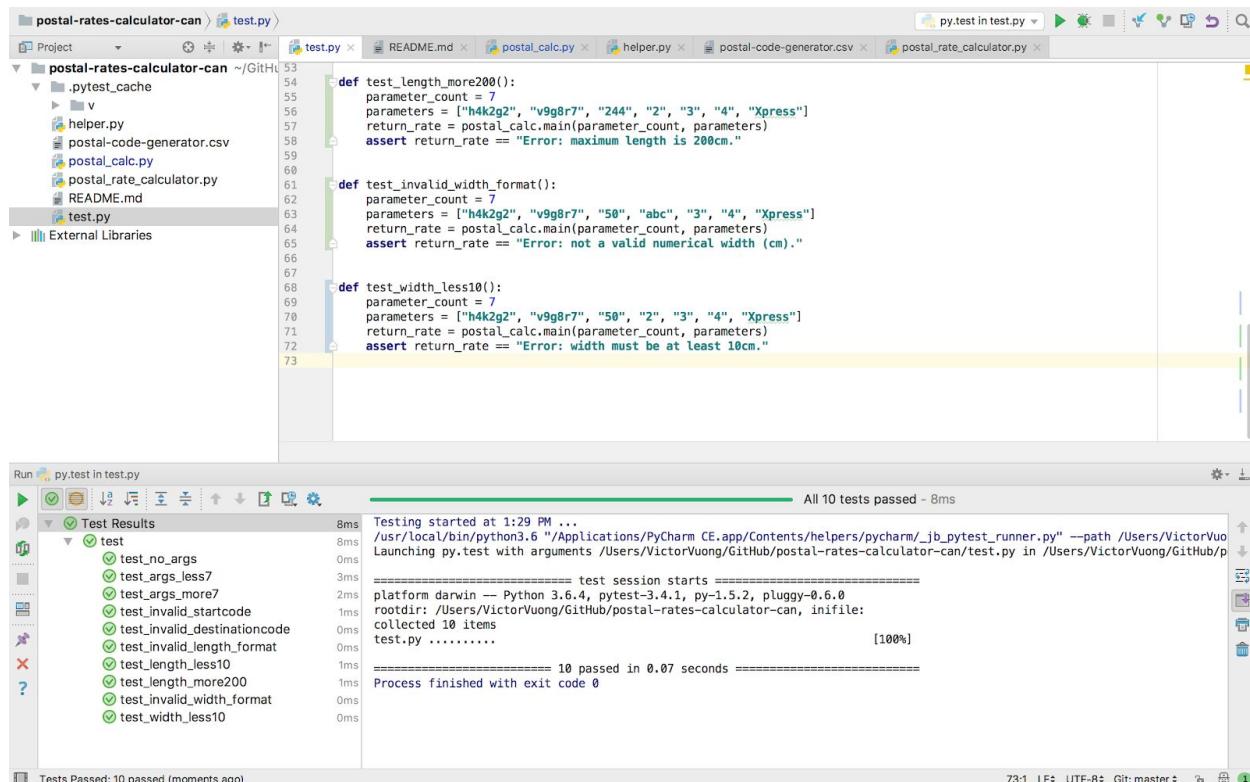
In the 'Run' tool window, the test results show 10 tests done, 1 failed, and 11ms execution time. The failed test is `test_width_less10` with an `AssertionError`.

Test Results:

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10

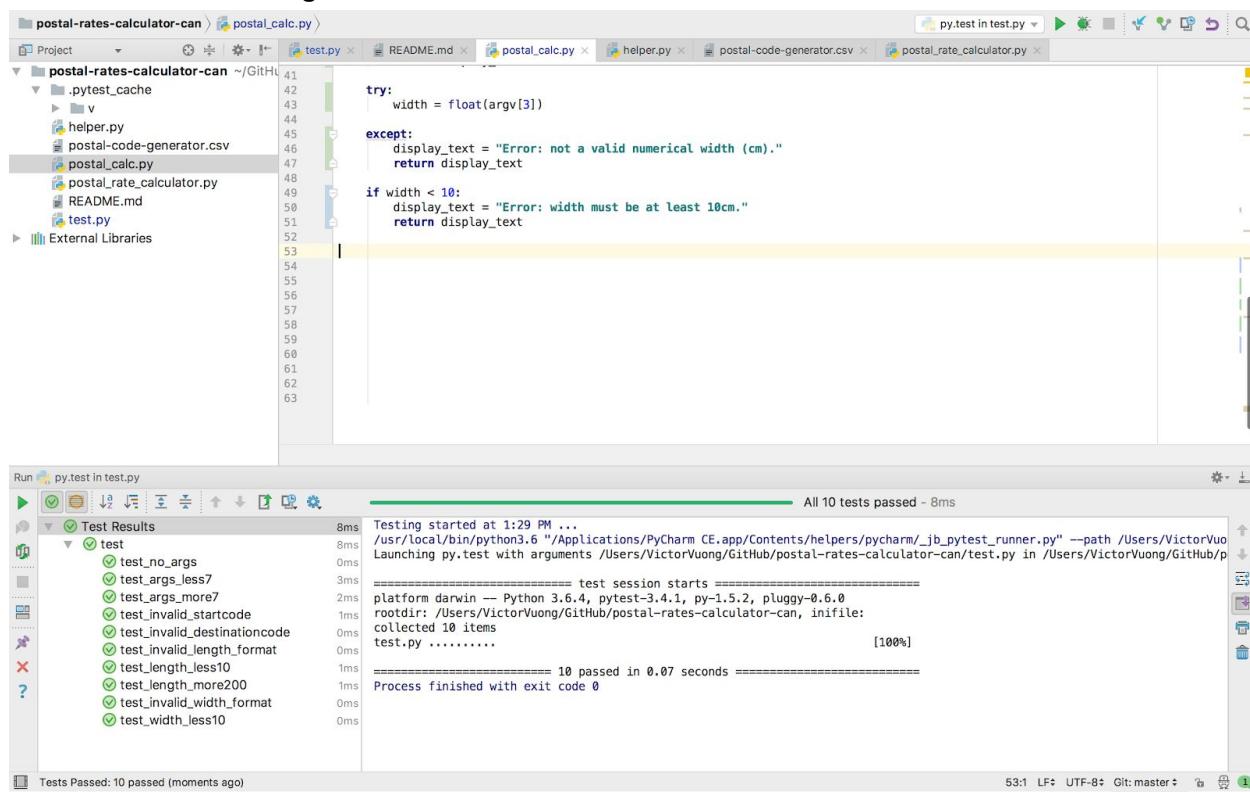
10 tests done: 1 failed - 11ms

Test 10 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains three test functions: `test_length_more200`, `test_invalid_width_format`, and `test_width_less10`. The bottom window shows the `Test Results` panel with a summary: "All 10 tests passed - 8ms". The results list includes all ten test cases from `test`, each marked with a green checkmark and a duration of 0ms.

Test 10 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, specifically focusing on the `width` parameter handling. The bottom window shows the `Test Results` panel with a summary: "All 10 tests passed - 8ms". The results list includes all ten test cases from `test`, each marked with a green checkmark and a duration of 0ms.

Test 11: Width too long

Purpose of the test: Make sure that the user inputs a width that is not more than the maximum value accepted (200cm)

Test Name: test_width_more200

Call setup: postal_calc.py h4k2g2 v9g8r7 50 201 3 4 Xpress

Expected Result: Error: maximum width is 200cm.

Note: It was decided that the range of valid inputs for the width would be extended up to 200 cm.

Test 11 Fail:

PyCharm IDE screenshot showing the project structure and the contents of `test.py`. The code contains three test functions: `test_invalid_width_format`, `test_width_less10`, and `test_width_more200`. The `test_width_more200` function fails with an `AssertionError`.

```
def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."
```

The 'Test Results' panel shows the following output:

```
11 tests done: 1 failed - 25ms
=====
  FAILURES =====
  test_width_more200
=====
  def test_width_more200():
      parameter_count = 7
      parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
      return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: maximum width is 200cm."
E     AssertionError: assert None == 'Error: maximum width is 200cm.'
```

Process finished with exit code 0

Test 11 Code:

PyCharm IDE screenshot showing the project structure and the contents of `postal_calc.py`. The code contains a function `test_width_more200` that fails with an `AssertionError`.

```
try:
    width = float(argv[3])
except:
    display_text = "Error: not a valid numerical width (cm)."
    return display_text

if width < 10:
    display_text = "Error: width must be at least 10cm."
    return display_text
```

The 'Test Results' panel shows the following output:

```
11 tests done: 1 failed - 25ms
=====
  FAILURES =====
  test_width_more200
=====
  def test_width_more200():
      parameter_count = 7
      parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
      return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: maximum width is 200cm."
E     AssertionError: assert None == 'Error: maximum width is 200cm.'
```

Process finished with exit code 0

Test 11 Succeed:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains 11 test cases for the `postal_calc` module. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 11 tests passed in 0.11 seconds.

```
def test_invalid_width_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "abc", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical width (cm.)"

def test_width_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: width must be at least 10cm."

def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "201", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200

All 11 tests passed - 12ms

Test 11 Code with changes:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include additional validation logic for width. The bottom window shows the `Run` tool window with the output of the pytest run, indicating that all 11 tests passed in 0.11 seconds.

```
display_text = "Error: length must be at least 10cm."
return display_text

if length > 200:
    display_text = "Error: maximum length is 200cm.."
    return display_text

try:
    width = float(argv[3])
except:
    display_text = "Error: not a valid numerical width (cm.)."
    return display_text

if width < 10:
    display_text = "Error: width must be at least 10cm."
    return display_text

if width > 200:
    display_text = "Error: maximum width is 200cm.."
    return display_text
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200

All 11 tests passed - 12ms

Test 12: Invalid height format

Purpose of the test: Make sure that the user inputs a numerical value for the height

Test Name: test_invalid_height_format

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 abc 4 Xpress

Expected Result: Error: not a valid numerical height (cm).

Note: It was decided that the input format for the height of the parcel is in decimal format.

Test 12 Fail:

```

def test_width_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: width must be at least 10cm."

def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "201", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical height (cm)."

```

Run py.test in test.py
12 tests done: 1 failed - 15ms

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format

===== FAILURES =====

test_invalid_height_format

```

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical height (cm)."
E     AssertionError: assert None == 'Error: not a valid numerical height (cm).'

test.py:86: AssertionError
===== 1 failed, 11 passed in 0.16 seconds =====
Process finished with exit code 0

```

Test 12 Code:

```

try:
    width = float(argv[3])
except:
    display_text = "Error: not a valid numerical width (cm)."
    return display_text

if width < 10:
    display_text = "Error: width must be at least 10cm."
    return display_text

if width > 200:
    display_text = "Error: maximum width is 200cm."
    return display_text

```

Run py.test in test.py
12 tests done: 1 failed - 15ms

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format

===== FAILURES =====

test_invalid_height_format

```

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>     assert return_rate == "Error: not a valid numerical height (cm)."
E     AssertionError: assert None == 'Error: not a valid numerical height (cm).'

test.py:86: AssertionError
===== 1 failed, 11 passed in 0.16 seconds =====
Process finished with exit code 0

```

Tests Failed: 11 passed, 1 failed (moments ago)

Test 12 Succeed:

Project: postal-rates-calculator-can ~GitHub

test.py

```

def test_width_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "2", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: width must be at least 10cm."

def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "201", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical height (cm)."

```

Run: py.test in test.py

All 12 tests passed - 11ms

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format

PEP 8: blank line at end of file

Test 12 Code with changes:

Project: postal-rates-calculator-can ~GitHub

test.py

```

display_text = "Error: width must be at least 10cm."
return display_text

if width > 200:
    display_text = "Error: maximum width is 200cm."
    return display_text

try:
    height = float(argv[4])
except:
    display_text = "Error: not a valid numerical height (cm)."
    return display_text

```

Run: py.test in test.py

All 12 tests passed - 11ms

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format

Tests Passed: 12 passed (moments ago)

Test 13: Height too short

Purpose of the test: Make sure that the user inputs a height that is not less than the minimum value accepted (10cm)

Test Name: test_height_less10

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 0 4 Xpress

Expected Result: Error: height must be at least 10cm.

Note: It was decided that the minimum height required for the parcel to be eligible for shipping is 10.

Test 13 Fail:

Project: postal-rates-calculator-can

File: test.py

```

def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "201", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical height (cm)."

def test_height_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: height must be at least 10cm."

```

Run: py.test in test.py

Test Results: 13 tests done: 1 failed - 18ms

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10**

AssertionError: assert None == 'Error: height must be at least 10cm.'

Process finished with exit code 0

Tests Failed: 12 passed, 1 failed (moments ago)

Test 13 Code:

Project: postal-rates-calculator-can

File: postal_calc.py

```

if width > 200:
    display_text = "Error: maximum width is 200cm."
    return display_text

try:
    height = float(argv[4])
except:
    display_text = "Error: not a valid numerical height (cm)."
    return display_text

```

Run: py.test in test.py

Test Results: 13 tests done: 1 failed - 18ms

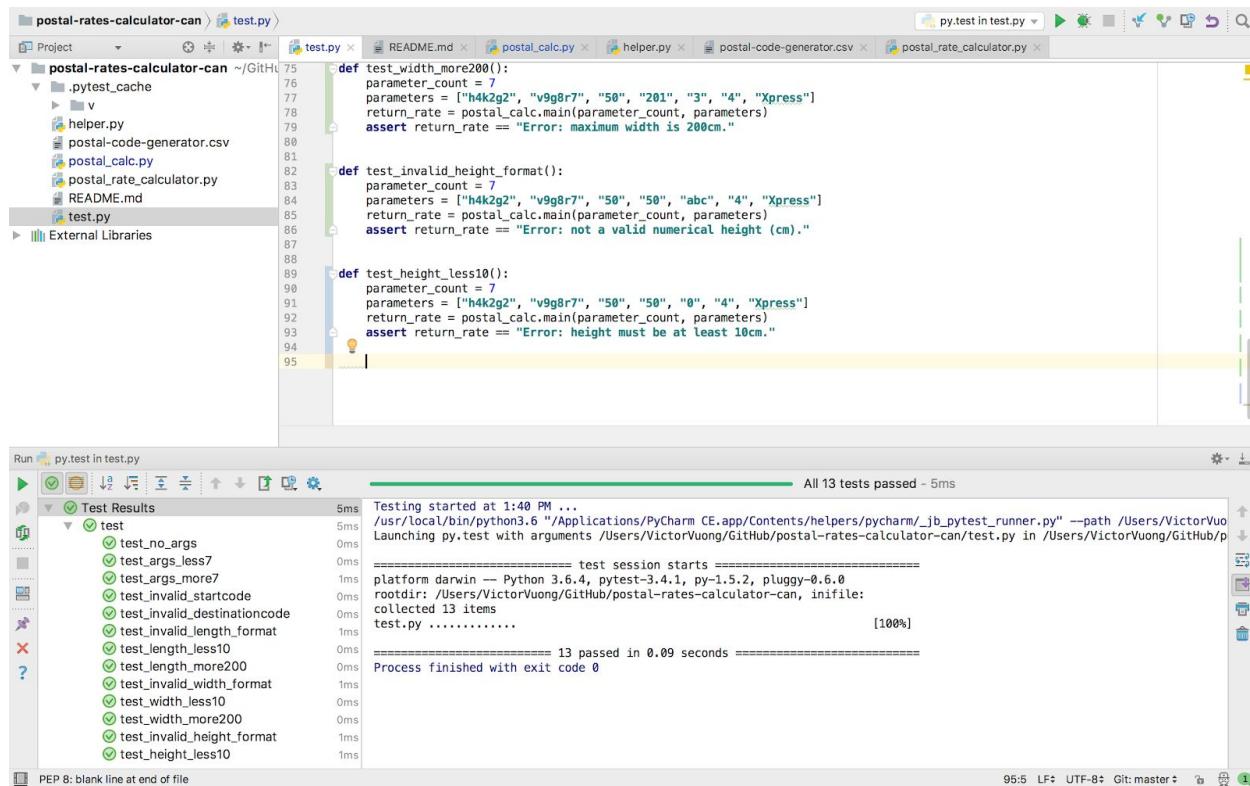
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10**

AssertionError: assert None == 'Error: height must be at least 10cm.'

Process finished with exit code 0

Tests Failed: 12 passed, 1 failed (moments ago)

Test 13 Succeed:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for postal rate calculations. The bottom window shows the `Test Results` tool window with a green checkmark next to each test name, indicating they all passed. The output pane shows the command-line logs for the pytest run, confirming 13 tests passed in 0.09 seconds.

```
def test_width_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "201", "3", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum width is 200cm."

def test_invalid_height_format():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid numerical height (cm)."

def test_height_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: height must be at least 10cm."
```

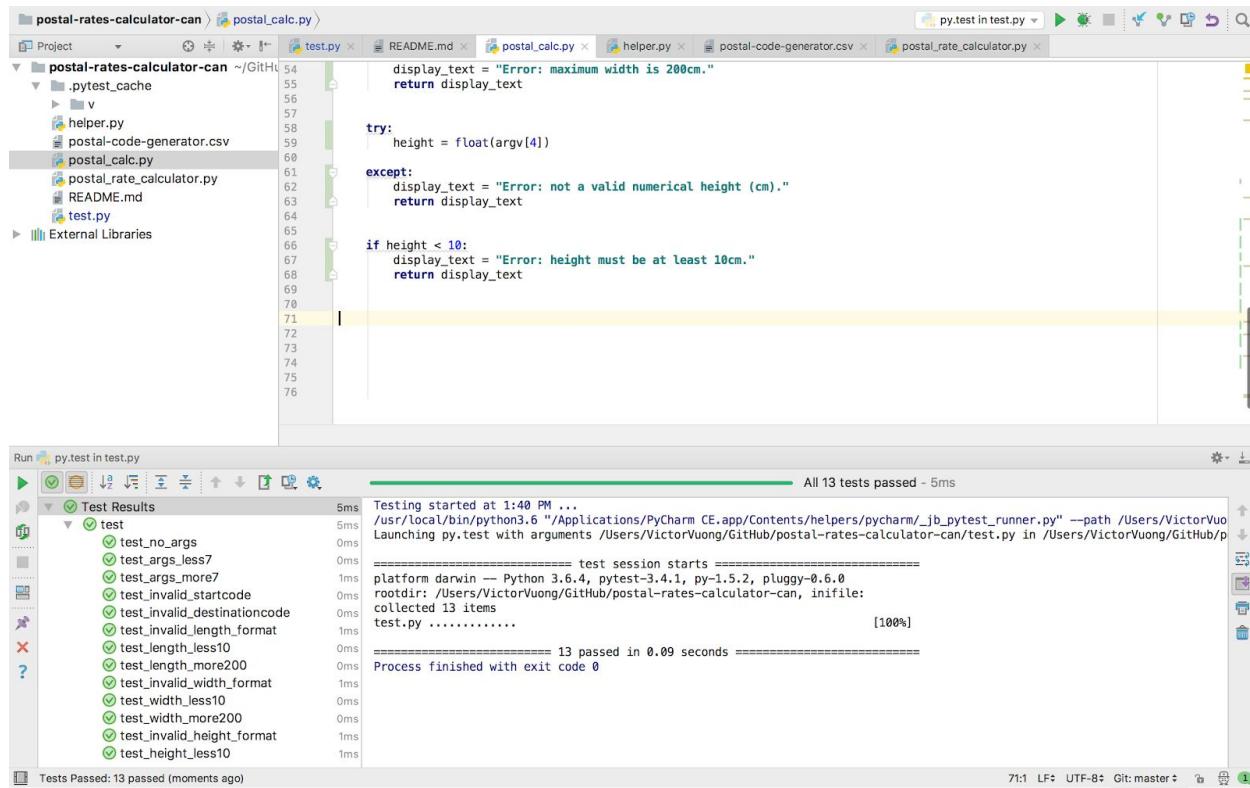
Test Results

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10

All 13 tests passed - 5ms

```
Testing started at 1:40 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 13 items
test.py ..... [100%]
=====
13 passed in 0.09 seconds
Process finished with exit code 0
```

Test 13 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which includes additional logic to handle invalid height inputs. The bottom window shows the `Test Results` tool window with a green checkmark next to each test name, indicating they all passed. The output pane shows the command-line logs for the pytest run, confirming 13 tests passed in 0.09 seconds.

```
display_text = "Error: maximum width is 200cm."
return display_text

try:
    height = float(argv[4])
except:
    display_text = "Error: not a valid numerical height (cm)."
    return display_text

if height < 10:
    display_text = "Error: height must be at least 10cm."
    return display_text
```

Test Results

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10

All 13 tests passed - 5ms

```
Testing started at 1:40 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 13 items
test.py ..... [100%]
=====
13 passed in 0.09 seconds
Process finished with exit code 0
```

Test 14: Height too big

Purpose of the test: Make sure that the user inputs a height that is not more than the maximum value accepted (200cm)

Test Name: test_height_more200

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 400 4 Xpress

Expected Result: Error: maximum height is 200cm.

Note: It was decided the range of valid inputs for the height would be extended up to 200.

Test 14 Fail:

```

parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
return_rate = postal_calc.main(parameter_count, parameters)
assert return_rate == "Error: not a valid numerical height (cm.)"

def test_height_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: height must be at least 10cm."

def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum height is 200cm."

```

Run py.test in test.py

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format
 - test_height_less10
 - test_height_more200

14 tests done: 1 failed - 24ms

E Assertion error: assert None == 'Error: maximum height is 200cm.'

test.py:100: Assertion error [100%]

===== FAILURES =====

test_height_more200

```

def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>   assert return_rate == "Error: maximum height is 200cm."
E   Assertion error: assert None == 'Error: maximum height is 200cm.'

```

test.py:100: Assertion error

===== 1 failed, 13 passed in 0.21 seconds =====

Process finished with exit code 0

Test 14 Code:

```

try:
    height = float(argv[4])
except:
    display_text = "Error: not a valid numerical height (cm.)."
    return display_text

if height < 10:
    display_text = "Error: height must be at least 10cm."
    return display_text

```

Run py.test in test.py

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format
 - test_height_less10
 - test_height_more200

14 tests done: 1 failed - 24ms

E Assertion error: assert None == 'Error: maximum height is 200cm.'

test.py:100: Assertion error [100%]

===== FAILURES =====

test_height_more200

```

def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>   assert return_rate == "Error: maximum height is 200cm."
E   Assertion error: assert None == 'Error: maximum height is 200cm.'

```

test.py:100: Assertion error

===== 1 failed, 13 passed in 0.21 seconds =====

Process finished with exit code 0

Test 14 Succeed:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test cases for height validation. The bottom window shows the `Test Results` panel with 14 tests passed, indicating successful execution.

```
parameters = ["h4k2g2", "v9g8r7", "50", "50", "abc", "4", "Xpress"]
return_rate = postal_calc.main(parameter_count, parameters)
assert return_rate == "Error: not a valid numerical height (cm)."

def test_height_less10():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: height must be at least 10cm."

def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum height is 200cm."
```

Test Results: All 14 tests passed - 8ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200

Process finished with exit code 0

Test 14 Code with changes:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to handle height validation. The bottom window shows the `Test Results` panel with 14 tests passed, indicating successful execution.

```
height = float(argv[4])

except:
    display_text = "Error: not a valid numerical height (cm)."
    return display_text

if height < 10:
    display_text = "Error: height must be at least 10cm."
    return display_text

if height > 200:
    display_text = "Error: maximum height is 200cm."
    return display_text
```

Test Results: All 14 tests passed - 8ms

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200

Process finished with exit code 0

Test 15: Correct volume price

Purpose of the test: Make sure that the computed volume price is correct (5\$ per meters cubed)

Test Name: test_volume_price

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 4 Xpress

Expected Result: 5.00

Note: It was decided that the price of the parcel is linearly dependant on the volume of the parcel with a rate of 5\$ / meters cubed.

Test 15 Fail:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `test.py`, which contains two failing test cases: `test_height_more200` and `test_volume_price`. The bottom part shows the `Run` tool window with the output of the test run, indicating 15 tests done, 1 failed, and a duration of 28ms. The failure for `test_volume_price` is highlighted with a yellow bar.

```
def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum height is 200cm."

def test_volume_price():
    length = 100
    width = 100
    height = 100
    price = postal_calc.calcVolumePrice(length, width, height)
    assert price == 5.00
```

Run py.test in test.py
15 tests done: 1 failed - 28ms

Test Results

| Test | Status | Time |
|------------------------------|--------|------|
| test_no_args | Passed | 28ms |
| test_args_less7 | Passed | 3ms |
| test_args_more7 | Passed | 1ms |
| test_invalid_startcode | Passed | 1ms |
| test_invalid_destinationcode | Passed | 1ms |
| test_invalid_length_format | Passed | 1ms |
| test_length_less10 | Passed | 1ms |
| test_length_more200 | Passed | 1ms |
| test_invalid_width_format | Passed | 1ms |
| test_width_less10 | Passed | 1ms |
| test_width_more200 | Passed | 1ms |
| test_invalid_height_format | Passed | 1ms |
| test_height_less10 | Passed | 2ms |
| test_height_more200 | Failed | 4ms |
| test_volume_price | Failed | 7ms |

===== FAILURES =====

```
def test_volume_price():
    length = 100
    width = 100
    height = 100
    price = postal_calc.calcVolumePrice(length, width, height)
>     assert price == 5.00
E     assert None == 5.0

test.py:108: AssertionError
[100%]
```

===== 1 failed, 14 passed in 0.22 seconds =====

Process finished with exit code 0

Test 15 Code:

The screenshot shows the PyCharm IDE interface. The top part displays the code for `postal_calc.py`, which contains logic for calculating volume price and handling height restrictions. The bottom part shows the `Run` tool window with the output of the test run, indicating 15 tests done, 1 failed, and a duration of 28ms. The failure for `test_volume_price` is highlighted with a yellow bar.

```
if height > 200:
    display_text = "Error: maximum height is 200cm."
    return display_text

def calcVolumePrice(length, width, height):
    return None
```

Run py.test in test.py
15 tests done: 1 failed - 28ms

Test Results

| Test | Status | Time |
|------------------------------|--------|------|
| test_no_args | Passed | 28ms |
| test_args_less7 | Passed | 3ms |
| test_args_more7 | Passed | 1ms |
| test_invalid_startcode | Passed | 1ms |
| test_invalid_destinationcode | Passed | 1ms |
| test_invalid_length_format | Passed | 1ms |
| test_length_less10 | Passed | 1ms |
| test_length_more200 | Passed | 1ms |
| test_invalid_width_format | Passed | 1ms |
| test_width_less10 | Passed | 1ms |
| test_width_more200 | Passed | 1ms |
| test_invalid_height_format | Passed | 1ms |
| test_height_less10 | Passed | 2ms |
| test_height_more200 | Passed | 4ms |
| test_volume_price | Failed | 7ms |

===== FAILURES =====

```
def test_volume_price():
    length = 100
    width = 100
    height = 100
    price = postal_calc.calcVolumePrice(length, width, height)
>     assert price == 5.00
E     assert None == 5.0

test.py:108: AssertionError
[100%]
```

===== 1 failed, 14 passed in 0.22 seconds =====

Process finished with exit code 0

Test 15 Succeed:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test functions for postal rate calculations. The bottom window shows the test results, indicating that all 15 tests have passed in 10ms.

```
def test_height_more200():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum height is 200cm."

def test_volume_price():
    length = 100
    width = 100
    height = 100
    price = postal_calc.calcVolumePrice(length, width, height)
    assert price == 5.00
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price

All 15 tests passed ~ 10ms

Test 15 Code with changes:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include a height validation and a different volume calculation logic. The bottom window shows the test results, indicating that all 15 tests have passed in 10ms.

```
if height > 200:
    display_text = "Error: maximum height is 200cm."
    return display_text

def calcVolumePrice(length, width, height):
    price = (0.01*length * 0.01*width * 0.01*height) * 5
    return price
```

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price

All 15 tests passed ~ 10ms

Test 16: Invalid weight format

Purpose of the test: Make sure that the user inputs a numerical value for the weight

Test Name: test_invalid_weight_format

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 50 abc Xpress

Expected Result: Error: not a valid numerical weight (kg).

Note: It was decided that the input format for the weight of the parcel is in decimal format.

Test 16 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, specifically the `test_invalid_weight_format` function. The bottom window shows the `Run` tool window with the output of the test run. The output indicates 16 tests were done, with 1 failing. The failing test is `test_invalid_weight_format`, which asserts that the return rate is an error message for non-numerical weight. The assertion fails because the function returns `None` instead of the expected error string.

```
assert return_rate == "Error: not a valid numerical weight (kg)."
```

```
test.py:115: AssertionError
```

```
===== FAILURES =====
```

```
def test_invalid_weight_format():
    parameter_count = 7
    parameters = ["h4K2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    > assert return_rate == "Error: not a valid numerical weight (kg)."
E     AssertionError: assert None == 'Error: not a valid numerical weight (kg).'
```

```
test.py:115: AssertionError
```

```
===== 1 failed, 15 passed in 0.14 seconds =====
```

```
Process finished with exit code 0
```

Test 16 Code:

The screenshot shows the Pycharm IDE interface. The top window displays the code for `postal_calc.py`, specifically the `calcVolumePrice` function. The bottom window shows the `Run` tool window with the output of the test run. The output indicates 16 tests were done, with 1 failing. The failing test is `test_invalid_weight_format`, which asserts that the return rate is an error message for non-numerical weight. The assertion fails because the function returns `None` instead of the expected error string.

```
def calcVolumePrice(length, width, height):
```

```
except:
    display_text = "Error: not a valid numerical height (cm)."
    return display_text
```

```
if height < 10:
    display_text = "Error: height must be at least 10cm."
    return display_text
```

```
if height > 200:
    display_text = "Error: maximum height is 200cm."
    return display_text
```

```
parameters = ["h4K2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
```

```
> assert return_rate == "Error: not a valid numerical weight (kg)."
E     AssertionError: assert None == 'Error: not a valid numerical weight (kg).'
```

```
test.py:115: AssertionError
```

```
===== FAILURES =====
```

```
def test_invalid_weight_format():
    parameter_count = 7
    parameters = ["h4K2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    > assert return_rate == "Error: not a valid numerical weight (kg)."
E     AssertionError: assert None == 'Error: not a valid numerical weight (kg).'
```

```
test.py:115: AssertionError
```

```
===== 1 failed, 15 passed in 0.14 seconds =====
```

```
Process finished with exit code 0
```

Test 16 Succeed:

Project structure:

- postal-rates-calculator-can
- test.py
- External Libraries

Code in test.py:

```

100     return_rate = postal_calc.main(parameter_count, postal_code_generator.csv)
101     assert return_rate == "Error: maximum height is 200cm."
102
103 def test_volume_price():
104     length = 100
105     width = 100
106     height = 100
107     price = postal_calc.calcVolumePrice(length, width, height)
108     assert price == 5.00
109
110 def test_invalid_weight_format():
111     parameter_count = 7
112     parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
113     return_rate = postal_calc.main(parameter_count, parameters)
114     assert return_rate == "Error: not a valid numerical weight (kg)."
115

```

Run Results:

All 16 tests passed - 4ms

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 4ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 0ms |
| test_invalid_destinationcode | Passed | 0ms |
| test_invalid_length_format | Passed | 0ms |
| test_length_less10 | Passed | 0ms |
| test_length_more200 | Passed | 0ms |
| test_invalid_width_format | Passed | 0ms |
| test_width_less10 | Passed | 0ms |
| test_width_more200 | Passed | 0ms |
| test_invalid_height_format | Passed | 1ms |
| test_height_less10 | Passed | 1ms |
| test_height_more200 | Passed | 0ms |
| test_volume_price | Passed | 0ms |
| test_invalid_weight_format | Passed | 0ms |

PEP 8: no newline at end of file

Test 16 Code with changes:

Project structure:

- postal-rates-calculator-can
- postal_calc.py
- test.py
- External Libraries

Code in postal_calc.py:

```

67     display_text = "Error: height must be at least 10cm."
68     return display_text
69
70 if height > 200:
71     display_text = "Error: maximum height is 200cm."
72     return display_text
73
74 try:
75     weight = float(argv[5])
76
77 except:
78     display_text = "Error: not a valid numerical weight (kg)."
79     return display_text
80
81
82 def calcVolumePrice(length, width, height):
83
84
85
86
87

```

Run Results:

All 16 tests passed - 4ms

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 4ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 0ms |
| test_invalid_destinationcode | Passed | 0ms |
| test_invalid_length_format | Passed | 0ms |
| test_length_less10 | Passed | 0ms |
| test_length_more200 | Passed | 0ms |
| test_invalid_width_format | Passed | 0ms |
| test_width_less10 | Passed | 0ms |
| test_width_more200 | Passed | 0ms |
| test_invalid_height_format | Passed | 1ms |
| test_height_less10 | Passed | 1ms |
| test_height_more200 | Passed | 0ms |
| test_volume_price | Passed | 0ms |
| test_invalid_weight_format | Passed | 0ms |

Tests Passed: 16 passed (moments ago)

Test 17: Weight too low

Purpose of the test: Make sure that the user inputs a weight that is not less than the minimum value accepted (0.1kg)

Test Name: test_weight_less_minimum

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 50 0 Xpress

Expected Result: Error: minimum weight is 0.1kg.

Note: It was decided that the minimum weight for the parcel to be eligible for shipping is 0.1

Test 17 Fail:

```

107     price = postal_calc.calcVolumePrice(length, width, height)
108     assert price == 5.00
109
110     def test_invalid_weight_format():
111         parameter_count = 7
112         parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
113         return_rate = postal_calc.main(parameter_count, parameters)
114         assert return_rate == "Error: not a valid numerical weight (kg)."
115
116     def test_weight_less_minimum():
117         parameter_count = 7
118         parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "Xpress"]
119         return_rate = postal_calc.main(parameter_count, parameters)
120         assert return_rate == "Error: minimum weight is 0.1kg."
121
122
123     test_weight_less_minimum()

```

Run py.test in test.py

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format
 - test_height_less10
 - test_height_more200
 - test_volume_price
 - test_invalid_weight_format
 - test_weight_less_minimum

```

parameter_count = 7
parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "0", "Xpress"]
> assert return_rate == "Error: minimum weight is 0.1kg."
E   AssertionError: assert None == 'Error: minimum weight is 0.1kg.'

test.py:121: AssertionError
[100%]

===== FAILURES =====
----- test_weight_less_minimum -----
def test_weight_less_minimum():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    > assert return_rate == "Error: minimum weight is 0.1kg."
    E   AssertionError: assert None == 'Error: minimum weight is 0.1kg.'

test.py:121: AssertionError
=====
1 failed, 16 passed in 0.17 seconds =====
Process finished with exit code 0

```

Tests Failed: 16 passed, 1 failed (moments ago)

Test 17 Code:

```

70
71     if height > 200:
72         display_text = "Error: maximum height is 200cm."
73         return display_text
74
75     try:
76         weight = float(argv[5])
77     except:
78         display_text = "Error: not a valid numerical weight (kg)."
79         return display_text
80
81
82
83
84
85
86
87
88
89     def calcVolumePrice(length, width, height):

```

Run py.test in test.py

Test Results

- test
 - test_no_args
 - test_args_less7
 - test_args_more7
 - test_invalid_startcode
 - test_invalid_destinationcode
 - test_invalid_length_format
 - test_length_less10
 - test_length_more200
 - test_invalid_width_format
 - test_width_less10
 - test_width_more200
 - test_invalid_height_format
 - test_height_less10
 - test_height_more200
 - test_volume_price
 - test_invalid_weight_format
 - test_weight_less_minimum

```

parameter_count = 7
parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "0", "Xpress"]
return_rate = postal_calc.main(parameter_count, parameters)
> assert return_rate == "Error: minimum weight is 0.1kg."
E   AssertionError: assert None == 'Error: minimum weight is 0.1kg.'

test.py:121: AssertionError
[100%]

===== FAILURES =====
----- test_weight_less_minimum -----
def test_weight_less_minimum():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    > assert return_rate == "Error: minimum weight is 0.1kg."
    E   AssertionError: assert None == 'Error: minimum weight is 0.1kg.'

test.py:121: AssertionError
=====
1 failed, 16 passed in 0.17 seconds =====
Process finished with exit code 0

```

Tests Failed: 16 passed, 1 failed (moments ago)

Test 17 Succeed:

The screenshot shows the PyCharm IDE interface. The top navigation bar includes tabs for 'postal-rates-calculator-can' and 'test.py'. The left sidebar displays the project structure under 'postal-rates-calculator-can' with files like 'helper.py', 'postal_code-generator.csv', 'test.py', and 'README.md'. The main code editor shows the 'test.py' file with several test cases for postal calculations. The bottom 'Run' tool window shows the results of a recent run: 'All 17 tests passed - 62ms'. The 'Test Results' section lists 17 individual test cases, each with a green checkmark indicating success. The terminal output at the bottom shows the command used to run the tests and the resulting log message.

```
187     price = postal_calc.calcVolumePrice(length, width, height)
188     assert price == 5.00
189
190
191     def test_invalid_weight_format():
192         parameter_count = 7
193         parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
194         return_rate = postal_calc.main(parameter_count, parameters)
195         assert return_rate == "Error: not a valid numerical weight (kg)."
196
197
198     def test_weight_less_minimum():
199         parameter_count = 7
200         parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "0", "Xpress"]
201         return_rate = postal_calc.main(parameter_count, parameters)
202         assert return_rate == "Error: minimum weight is 0.1kg."
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
```

test_weight_less_minimum()

Run py.test in test.py

All 17 tests passed - 62ms

Test Results

- test
 - ✓ test_no_args
 - ✓ test_args_less7
 - ✓ test_args_more7
 - ✓ test_invalid_startcode
 - ✓ test_invalid_destinationcode
 - ✓ test_invalid_length_format
 - ✓ test_length_less10
 - ✓ test_length_more200
 - ✓ test_invalid_width_format
 - ✓ test_width_less10
 - ✓ test_width_more200
 - ✓ test_invalid_height_format
 - ✓ test_height_less10
 - ✓ test_height_more200
 - ✓ test_volume_price
 - ✓ test_invalid_weight_format
 - ✓ test_weight_less_minimum

Testing started at 2:12 PM ...

/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can

Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can

platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0

rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:

collected 17 items

test.py [100%]

===== 17 passed in 0.33 seconds =====

Process finished with exit code 0

Test 17 Code with changes:

The screenshot shows the PyCharm IDE interface. The top navigation bar includes tabs for 'test.py', 'README.md', 'postal_calc.py', 'helper.py', 'postal_code-generator.csv', and 'postal_rate_calculator.py'. The left sidebar displays the project structure under 'postal-rates-calculator-can' with files like '.pytest_cache', 'v', 'helper.py', 'postal_code-generator.csv', 'postal_calc.py', 'postal_rate_calculator.py', 'README.md', and 'test.py'. Below the sidebar is an 'External Libraries' section. The main code editor area contains Python code for 'test.py', specifically handling arguments and calculating volume price. The bottom panel shows the 'Run' tool window with a green 'All 17 tests passed - 62ms' status bar. The 'Test Results' section lists 17 test cases under the 'test' group, all marked with a green checkmark, indicating they passed. The log output shows the test session starting at 2:12 PM, running on Darwin with Python 3.6.4, and collecting 17 items from 'test.py'. The total execution time was 0.33 seconds.

```
display_text = "Error: maximum height is 200cm."
    return display_text

try:
    weight = float(argv[5])

except:
    display_text = "Error: not a valid numerical weight (kg)."
    return display_text

if weight < 0.1:
    display_text = "Error: minimum weight is 0.1kg."
    return display_text

def calcVolumePrice(length, width, height):
```

Run py.test in test.py

All 17 tests passed - 62ms

Test Results

- test
 - ✓ test_no_args
 - ✓ test_args_less7
 - ✓ test_args_more7
 - ✓ test_invalid_startcode
 - ✓ test_invalid_destinationcode
 - ✓ test_invalid_length_format
 - ✓ test_length_less10
 - ✓ test_length_more200
 - ✓ test_invalid_width_format
 - ✓ test_width_less10
 - ✓ test_width_more200
 - ✓ test_invalid_height_format
 - ✓ test_height_less10
 - ✓ test_height_more200
 - ✓ test_volume_price
 - ✓ test_invalid_weight_format
 - ✓ test_weight_less_minimum

Testing started at 2:12 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 17 items
test.py [100%]
===== 17 passed in 0.33 seconds ======

Process finished with exit code 0

Test 18: Weight too high

Purpose of the test: Make sure that the user inputs a weight that is not more than the maximum value accepted (30kg)

Test Name: test_weight_more_maximum

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 50 31 Xpress

Expected Result: Error: maximum weight is 30kg.

Note: It was decided that the range of valid inputs for the weight would be extended up to 30.

Test 18 Fail:

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "postal-rates-calculator-can". It contains files like `helper.py`, `postal_code-generator.csv`, and `test.py`.
- Code Editor:** The `test.py` file is open, containing Python test cases for the `postal_calc` module. The code includes assertions for minimum and maximum weight limits.
- Run Tab:** Shows the results of 18 tests, with 1 failure in 22ms.
- Test Results:** A detailed breakdown of the test results, including the failure for the `test_weight_more_maximum` test.
- Status Bar:** Shows "Tests Failed: 17 passed, 1 failed (moments ago)" and "Process finished with exit code 0".

Test 18 Code:

The screenshot shows the PyCharm IDE interface. The top navigation bar includes tabs for 'test.py' and 'py.test in test.py'. The left sidebar displays the project structure for 'postal-rates-calculator-can' with files like 'helper.py', 'postal_code-generator.csv', 'postal_calc.py', 'README.md', 'test.py', and 'External Libraries'. The main code editor window shows Python code for 'postal_calc.py' with several code review annotations (green highlights and checkmarks) on lines 75 through 92. Below the code editor is a 'Run' toolbar with various icons. The bottom panel displays the 'Test Results' tool window. It lists 18 tests completed, with one failure: 'test_weight_more_maximum' failed with an AssertionError. The failure message is: 'AssertionError: assert None == 'Error: maximum weight is 30kg.''. The test results table includes columns for test name, duration, and status. The status for the failing test is 'FAILURES'.

| Test | Duration | Status |
|------------------------------|----------|----------|
| test | 22ms | PASSED |
| test_no_args | 22ms | PASSED |
| test_args_less7 | 0ms | PASSED |
| test_args_more7 | 0ms | PASSED |
| test_invalid_startcode | 1ms | PASSED |
| test_invalid_destinationcode | 4ms | PASSED |
| test_invalid_length_format | 1ms | PASSED |
| test_length_less10 | 0ms | PASSED |
| test_length_more200 | 3ms | PASSED |
| test_invalid_width_format | 0ms | PASSED |
| test_width_less10 | 1ms | PASSED |
| test_width_more200 | 0ms | PASSED |
| test_invalid_height_format | 0ms | PASSED |
| test_height_less10 | 1ms | PASSED |
| test_height_more200 | 4ms | PASSED |
| test_volume_price | 1ms | PASSED |
| test_invalid_weight_format | 1ms | PASSED |
| test_weight_less_minimum | 1ms | PASSED |
| test_weight_more_maximum | 4ms | FAILURES |

Test 18 Succeed:

PyCharm Project Structure:

- postal-rates-calculator-can
- .pytest_cache
- v
- helper.py
- postal-code-generator.csv
- postal_calc.py
- postal_rate_calculator.py
- README.md
- test.py
- External Libraries

Code in test.py:

```
114     return_rate = postal_calc.main(parameter_count, parameters)
115     assert return_rate == "Error: not a valid numerical weight (kg)."
116
117
118     def test_weight_less_minimum():
119         parameter_count = 7
120         parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "0", "Xpress"]
121         return_rate = postal_calc.main(parameter_count, parameters)
122         assert return_rate == "Error: minimum weight is 0.kg."
123
124
125     def test_weight_more_maximum():
126         parameter_count = 7
127         parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "31", "Xpress"]
128         return_rate = postal_calc.main(parameter_count, parameters)
129         assert return_rate == "Error: maximum weight is 30kg."
130
```

Run Results:

All 18 tests passed - 10ms

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 0ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 0ms |
| test_invalid_destinationcode | Passed | 0ms |
| test_invalid_length_format | Passed | 0ms |
| test_length_less10 | Passed | 1ms |
| test_length_more200 | Passed | 1ms |
| test_invalid_width_format | Passed | 1ms |
| test_width_less10 | Passed | 1ms |
| test_width_more200 | Passed | 1ms |
| test_invalid_height_format | Passed | 0ms |
| test_height_less10 | Passed | 2ms |
| test_height_more200 | Passed | 0ms |
| test_volume_price | Passed | 0ms |
| test_invalid_weight_format | Passed | 0ms |
| test_weight_less_minimum | Passed | 0ms |
| test_weight_more_maximum | Passed | 0ms |

Tests Passed: 18 passed (moments ago)

Test 18 Code with changes:

PyCharm Project Structure:

- postal-rates-calculator-can
- .pytest_cache
- v
- helper.py
- postal-code-generator.csv
- postal_calc.py
- postal_rate_calculator.py
- README.md
- test.py
- External Libraries

Code in postal_calc.py:

```
80     display_text = "Error: not a valid numerical weight (kg)."
81     return display_text
82
83
84     if weight < 0.1:
85         display_text = "Error: minimum weight is 0.1kg."
86         return display_text
87
88
89     if weight > 30:
90         display_text = "Error: maximum weight is 30kg."
91         return display_text
92
93
94
95
96
97
98     def calcVolumePrice(length, width, height):
```

Run Results:

All 18 tests passed - 10ms

| Test | Status | Time (ms) |
|------------------------------|--------|-----------|
| test_no_args | Passed | 0ms |
| test_args_less7 | Passed | 0ms |
| test_args_more7 | Passed | 0ms |
| test_invalid_startcode | Passed | 0ms |
| test_invalid_destinationcode | Passed | 0ms |
| test_invalid_length_format | Passed | 0ms |
| test_length_less10 | Passed | 1ms |
| test_length_more200 | Passed | 1ms |
| test_invalid_width_format | Passed | 1ms |
| test_width_less10 | Passed | 2ms |
| test_width_more200 | Passed | 1ms |
| test_invalid_height_format | Passed | 0ms |
| test_height_less10 | Passed | 2ms |
| test_height_more200 | Passed | 0ms |
| test_volume_price | Passed | 0ms |
| test_invalid_weight_format | Passed | 0ms |
| test_weight_less_minimum | Passed | 0ms |
| test_weight_more_maximum | Passed | 0ms |

Tests Passed: 18 passed (moments ago)

Test 19: Correct weight price

Purpose of the test: Make sure that the computed weight price is correct (0.5\$ per kg)

Test Name: test_weight_price

Call setup: postal_calc.py h4k2g2 v9g8r7 50 50 50 3 Xpress

Expected Result: 1.5

Note: It was decided that the price of the parcel is linearly dependant on the weight of the parcel with a rate of 0.5\$ / kg.

Test 19 Fail:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'test.py' file is selected in the left sidebar. The code editor displays two test cases: `test_weight_more_maximum` and `test_weight_price`. The `test_weight_price` test fails with an assertion error. The run tool window at the bottom shows the test results: 19 tests done, 1 failed. The failed test is `test_weight_price`, which expected 1.5 but got None.

```
def test_weight_more_maximum():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "31", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum weight is 30kg."

def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
    assert price == 1.5
```

```
19 tests done: 1 failed - 8ms
Test Results
  test
    ✓ test_no_args
    ✓ test_args_less7
    ✓ test_args_more7
    ✓ test_invalid_startcode
    ✓ test_invalid_destinationcode
    ✓ test_invalid_length_format
    ✓ test_length_less10
    ✓ test_length_more200
    ✓ test_length_width200
    ✓ test_invalid_height_format
    ✓ test_height_less10
    ✓ test_height_more200
    ✓ test_volume_price
    ✓ test_invalid_weight_format
    ✓ test_weight_less_minimum
    ✓ test_weight_more_maximum
    ⚡ test_weight_price
```

```
Actual :None
<Click to see difference>
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
>     assert price == 1.5
E     assert None == 1.5

test.py:135: AssertionError
=====
===== FAILURES =====
_____ test_weight_price _____
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
>     assert price == 1.5
E     assert None == 1.5

test.py:135: AssertionError
=====
1 failed, 18 passed in 0.15 seconds
Process finished with exit code 0
```

Test 19 Code:

The screenshot shows the PyCharm IDE interface with the project structure for 'postal-rates-calculator-can' open. The 'postal_calc.py' file is selected in the left sidebar. The code editor displays the implementation of the `calcWeightPrice` function, which returns None if the weight is greater than 30. The run tool window at the bottom shows the test results: 19 tests done, 1 failed. The failed test is `test_weight_price`, which expected 1.5 but got None.

```
if weight > 30:
    display_text = "Error: maximum weight is 30kg."
    return display_text

def calcVolumePrice(length, width, height):
    price = (0.01*length * 0.01*width * 0.01*height) * 5
    return price

def calcWeightPrice(weight):
    return None
```

```
19 tests done: 1 failed - 8ms
Test Results
  test
    ✓ test_no_args
    ✓ test_args_less7
    ✓ test_args_more7
    ✓ test_invalid_startcode
    ✓ test_invalid_destinationcode
    ✓ test_invalid_length_format
    ✓ test_length_less10
    ✓ test_length_more200
    ✓ test_length_width200
    ✓ test_invalid_height_format
    ✓ test_height_less10
    ✓ test_height_more200
    ✓ test_volume_price
    ✓ test_invalid_weight_format
    ✓ test_weight_less_minimum
    ✓ test_weight_more_maximum
    ⚡ test_weight_price
```

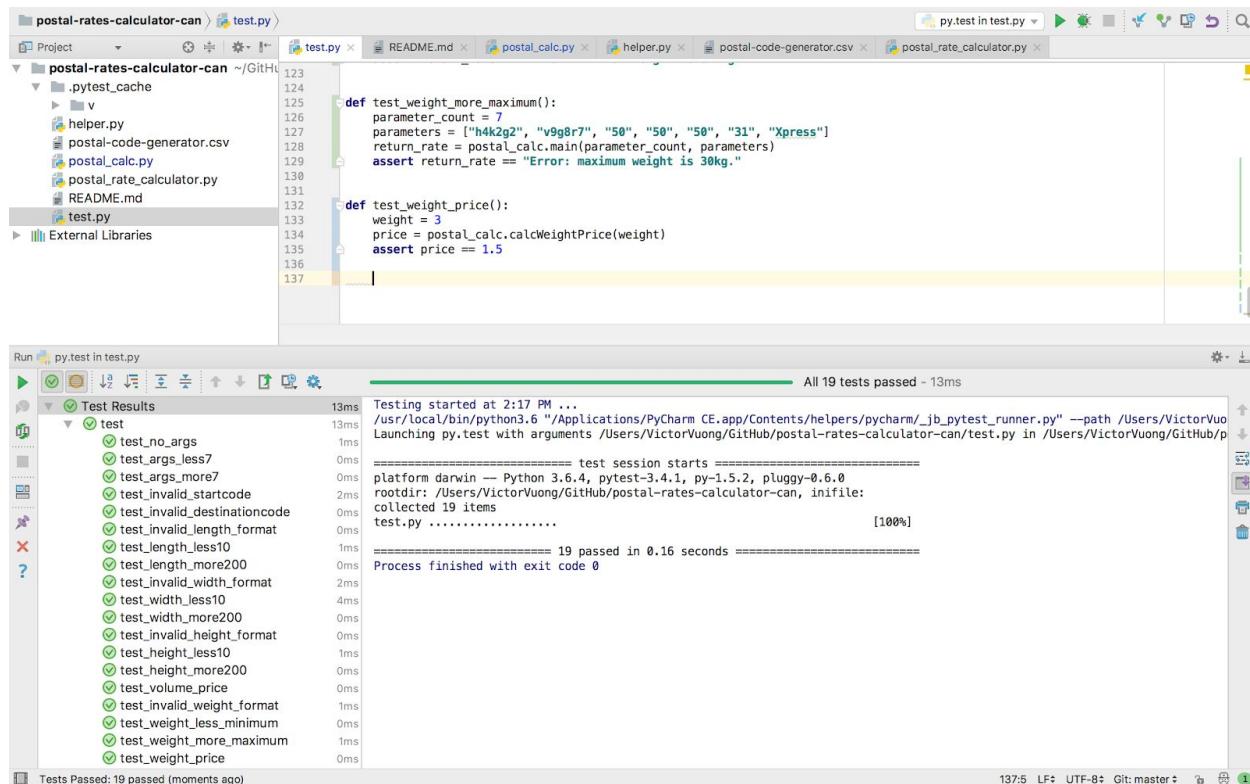
```
Actual :None
<Click to see difference>
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
>     assert price == 1.5
E     assert None == 1.5

test.py:135: AssertionError
=====
===== FAILURES =====
_____ test_weight_price _____
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
>     assert price == 1.5
E     assert None == 1.5

test.py:135: AssertionError
=====
1 failed, 18 passed in 0.15 seconds
Process finished with exit code 0
```

Tests Failed: 18 passed, 1 failed (moments ago)

Test 19 Succeed:



Project: postal-rates-calculator-can ~GitHub
File: test.py

```
def test_weight_more_maximum():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "31", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: maximum weight is 30kg."
```

```
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
    assert price == 1.5
```

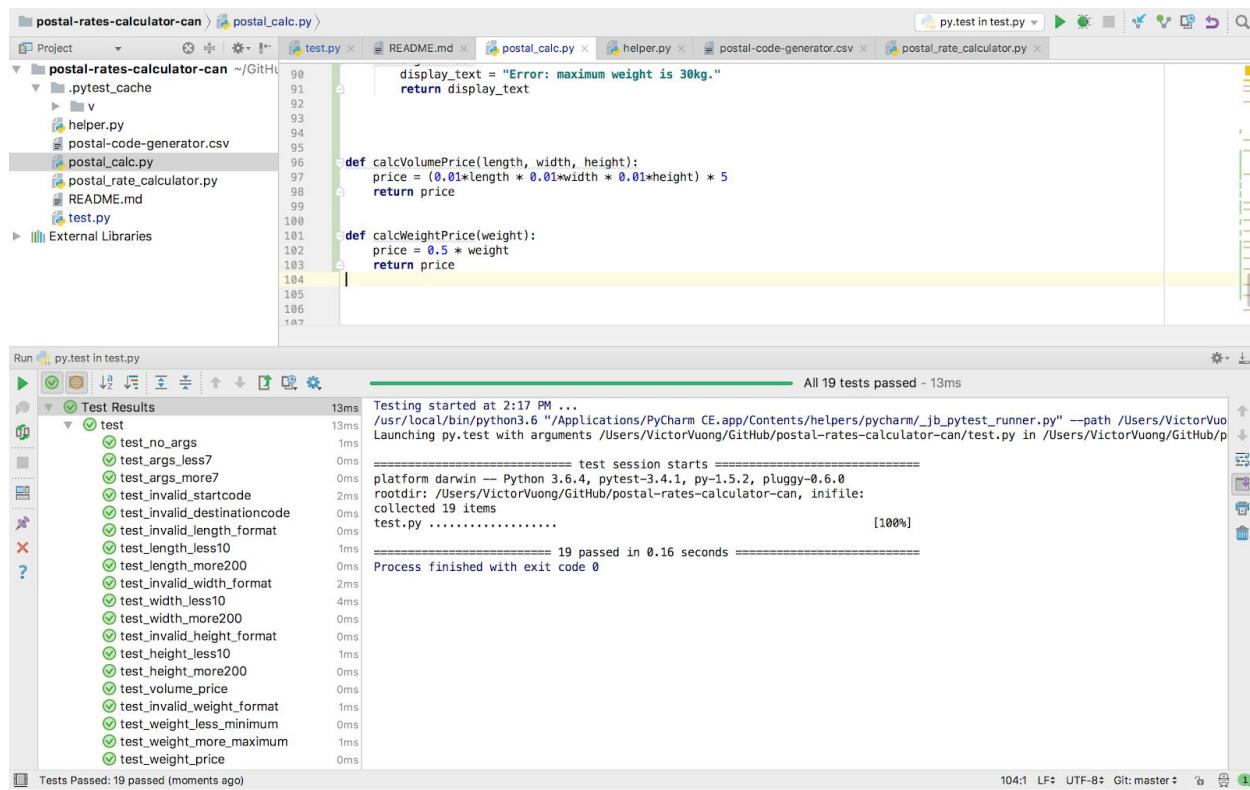
Run: py.test in test.py
All 19 tests passed - 13ms

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_length_width200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price
- test_invalid_weight_format
- test_weight_less_minimum
- test_weight_more_maximum
- test_weight_price

Tests Passed: 19 passed (moments ago)

Test 19 Code with changes:



Project: postal-rates-calculator-can ~GitHub
File: postal_calc.py

```
display_text = "Error: maximum weight is 30kg."
return display_text
```

```
def calcVolumePrice(length, width, height):
    price = (0.01*length * 0.01*width * 0.01*height) * 5
    return price
```

```
def calcWeightPrice(weight):
    price = 0.5 * weight
    return price
```

Run: py.test in test.py
All 19 tests passed - 13ms

Test Results

- test
- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_length_width200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price
- test_invalid_weight_format
- test_weight_less_minimum
- test_weight_more_maximum
- test_weight_price

Tests Passed: 19 passed (moments ago)

Test 20: Invalid post-type

Purpose of the test: Make sure that the user inputs a valid post type (Regular, Xpress or Priority) case sensitive

Test Name: test_invalid_post_type

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 1 hello

Expected Result: Error: not a valid post type (Regular, Xpress or Priority) case sensitive.

Test 20 Fail:

```

def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
    assert price == 1.5

def test_invalid_post_type():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "hello"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."

```

Run py.test in test.py
20 tests done: 1 failed - 19ms
<Click to see differences>
test
test_no_args
test_args_less7
test_args_more7
test_invalid_startcode
test_invalid_destinationcode
test_invalid_length_format
test_length_less10
test_length_more200
test_invalid_width_format
test_width_less10
test_width_more200
test_invalid_height_format
test_height_less10
test_height_more200
test_volume_price
test_invalid_weight_format
test_weight_less_minimum
test_weight_more_maximum
test_weight_price
test_invalid_post_type
[100%]
===== FAILURES =====
test_invalid_post_type
===== test_invalid_post_type =====
test.py:142: AssertionError
1 failed, 19 passed in 0.23 seconds
Process finished with exit code 0

Test 20 Code:

```

def main():
    if weight < 0.1:
        display_text = "Error: minimum weight is 0.1kg."
        return display_text

    if weight > 30:
        display_text = "Error: maximum weight is 30kg."
        return display_text

```

Run py.test in test.py
20 tests done: 1 failed - 19ms
<Click to see differences>
test
test_no_args
test_args_less7
test_args_more7
test_invalid_startcode
test_invalid_destinationcode
test_invalid_length_format
test_length_less10
test_length_more200
test_invalid_width_format
test_width_less10
test_width_more200
test_invalid_height_format
test_height_less10
test_height_more200
test_volume_price
test_invalid_weight_format
test_weight_less_minimum
test_weight_more_maximum
test_weight_price
test_invalid_post_type
[100%]
===== FAILURES =====
test_invalid_post_type
===== test_invalid_post_type =====
test.py:142: AssertionError
1 failed, 19 passed in 0.23 seconds
Process finished with exit code 0

Test 20 Succeed:

The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'test.py' and 'py.test in test.py'. The code editor displays a portion of 'test.py' with two test functions: 'test_weight_price' and 'test_invalid_post_type'. The 'Run' tool window at the bottom shows the output of the test run: 'All 20 tests passed - 9ms'. The 'Test Results' section on the left lists 20 test cases, all of which have passed, indicated by green checkmarks.

```
def test_weight_price():
    weight = 3
    price = postal_calc.calcWeightPrice(weight)
    assert price == 1.5

def test_invalid_post_type():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "hello"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
```

Test 20 Code with changes:

The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'test.py' and 'py.test in test.py'. The code editor displays a portion of 'test.py' with the same two test functions as before. The 'Run' tool window at the bottom shows the output of the test run: 'All 20 tests passed - 8ms'. The 'Test Results' section on the left lists 20 test cases, all of which have passed, indicated by green checkmarks.

```
if weight > 30:
    display_text = "Error: maximum weight is 30kg."
    return display_text

post_type = argv[6]

if post_type != "Regular" and post_type != "Xpress" and post_type != "Priority":
    display_text = "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
    return display_text
```

Test 21: Correct Regular post-type cost

Purpose of the test: Make sure that the “Regular” post type returns the correct additional cost

Test Name: test_regular_type_price

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 1 Regular

Expected Result: 5.00

Note: this is a test for the calcPostTypePrice method, which returns a correct fixed price. It was decided that the the Regular post-type has a fixed price of \$5.00.

Test 21 Fail:

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "postal-rates-calculator-can". It contains files: .pytest_cache, v, helper.py, postal-code-generator.csv, postal_calc.py, postal_rate_calculator.py, README.md, and test.py.
- Code Editor:** The code editor shows a portion of the test.py file. A specific line of code is highlighted in yellow:

```
assert price == 5.0
```
- Run Tool Window:** The "Run" tool window shows the results of the test run:
 - Total tests: 21
 - Failed tests: 1
 - Time taken: 9ms
- Test Results:** The "Test Results" panel displays the failure details for the test_regular_type_price() test. It shows the expected value (None) and the actual value (5.0). The error message is:

```
test.py:148: AssertionError
```
- Output:** The terminal output shows the failure:

```
===== FAILURES =====
test_regular_type_price
=====
def test_regular_type_price():
    post_type = "Regular"
    price = postal_calc.calcPostTypePrice(post_type)
>   assert price == 5.0
E   assert None == 5.0
test.py:148: AssertionError
=====
1 failed, 20 passed in 0.16 seconds
=====
Process finished with exit code 0
```

Test 21 Code:

The screenshot shows the PyCharm IDE interface with the following details:

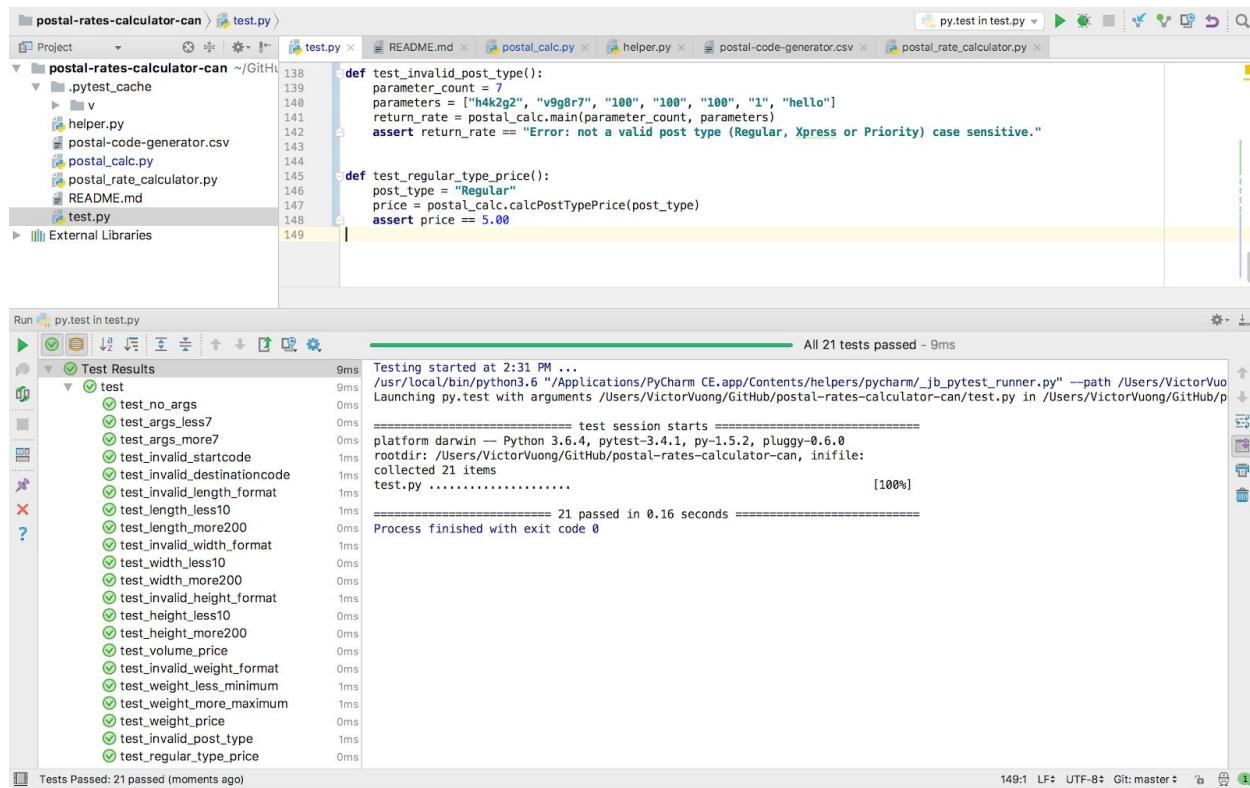
- Project Structure:** The project is named "postal-rates-calculator-can". It contains files: .pytest_cache, v, helper.py, postal-code-generator.csv, postal_calc.py, postal_rate_calculator.py, README.md, and test.py.
- Code Editor:** The code editor shows a portion of the postal_calc.py file. A specific line of code is highlighted in yellow:

```
return None
```
- Run Tool Window:** The "Run" tool window shows the results of the test run:
 - Total tests: 21
 - Failed tests: 1
 - Time taken: 9ms
- Test Results:** The "Test Results" panel displays the failure details for the test_regular_type_price() test. It shows the expected value (None) and the actual value (5.0). The error message is:

```
test.py:148: AssertionError
```
- Output:** The terminal output shows the failure:

```
===== FAILURES =====
test_regular_type_price
=====
def test_regular_type_price():
    post_type = "Regular"
    price = postal_calc.calcPostTypePrice(post_type)
>   assert price == 5.0
E   assert None == 5.0
test.py:148: AssertionError
=====
1 failed, 20 passed in 0.16 seconds
=====
Process finished with exit code 0
```

Test 21 Succeed:

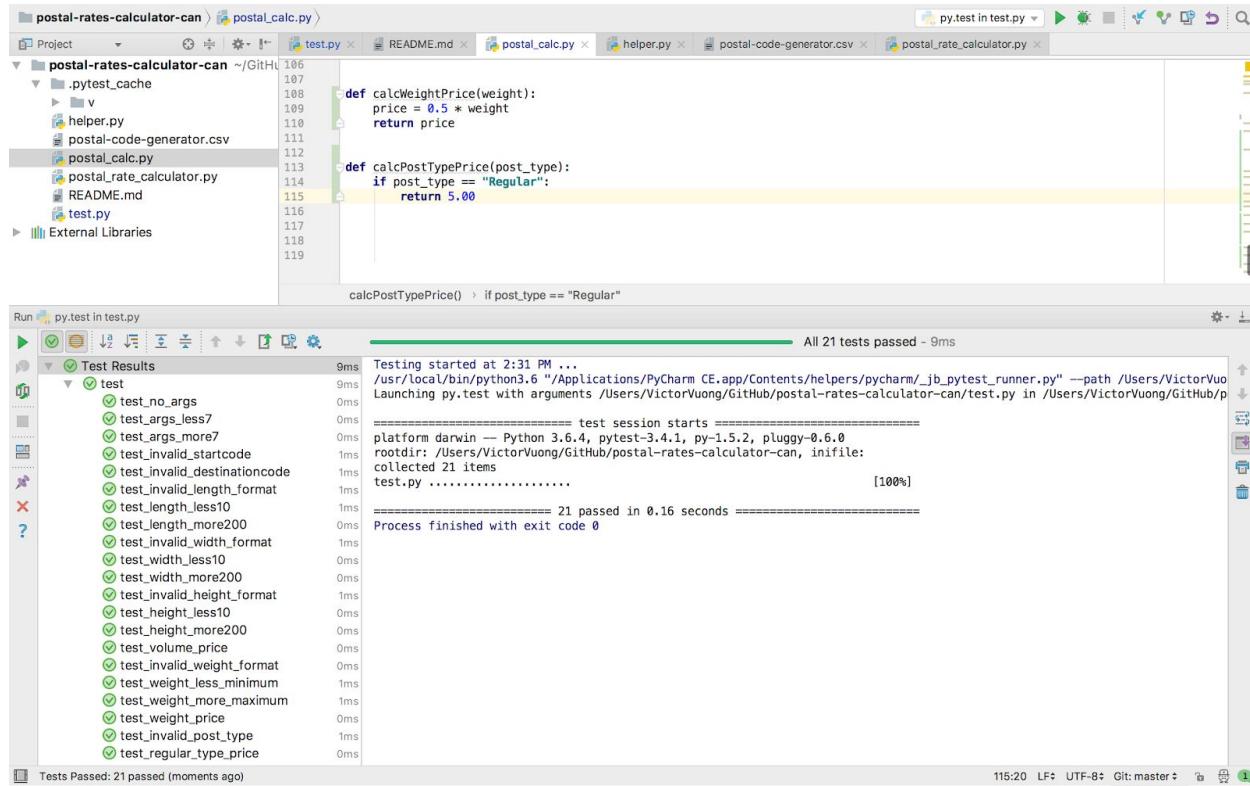


The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, which contains several test cases for postal rates. The bottom window shows the "Test Results" tool window with a green checkmark next to "All 21 tests passed - 9ms". The log output shows the test session starting, platform details, and the process finished with exit code 0.

```
def test_invalid_post_type():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "hello"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."  
  
def test_regular_type_price():
    post_type = "Regular"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 5.00
```

Tests Passed: 21 passed (moments ago)

Test 21 Code with changes:



The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, which has been modified to include a new function `calcPostTypePrice`. The bottom window shows the "Test Results" tool window with a green checkmark next to "All 21 tests passed - 9ms". The log output shows the test session starting, platform details, and the process finished with exit code 0.

```
def calcWeightPrice(weight):
    price = 0.5 * weight
    return price  
  
def calcPostTypePrice(post_type):
    if post_type == "Regular":
        return 5.00
```

Tests Passed: 21 passed (moments ago)

Test 22: Correct Xpress post-type cost

Purpose of the test: Make sure that the “Xpress” post type returns the correct additional cost

Test Name: test_Xpress_type_price

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 1 Xpress

Expected Result: 10.00

Note: this is a test for the calcPostTypePrice method, which returns a correct fixed price. It was decided that the Xpress post-type has a fixed price of \$10.00.

Test 22 Fail:

The screenshot shows the PyCharm IDE interface with the project 'postal-rates-calculator-can' open. The code editor displays `test.py` containing two failing test cases: `test_regular_type_price` and `test_Xpress_type_price`. The `test_Xpress_type_price` test fails with an assertion error at line 154, where it expects `None` but gets `10.0`.

Run Results: 22 tests done: 1 failed - 25ms

Test Results:

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price
- test_invalid_weight_format
- test_weight_less_minimum
- test_weight_more_maximum
- test_weight_price
- test_invalid_post_type
- test_regular_type_price
- test_Xpress_type_price (failed)

===== FAILURES =====

def test_Xpress_type_price():
 post_type = "Xpress"
 price = postal_calc.calcPostTypePrice(post_type)
> assert price == 10.00
E assert None == 10.0

===== 1 failed, 21 passed in 0.27 seconds =====

Test 22 Code:

The screenshot shows the PyCharm IDE interface with the project 'postal-rates-calculator-can' open. The code editor displays `postal_calc.py` containing the implementation of `calcPostTypePrice`. The function returns `5.00` for the `"Regular"` post type.

Run Results: 22 tests done: 1 failed - 25ms

Test Results:

- test_no_args
- test_args_less7
- test_args_more7
- test_invalid_startcode
- test_invalid_destinationcode
- test_invalid_length_format
- test_length_less10
- test_length_more200
- test_invalid_width_format
- test_width_less10
- test_width_more200
- test_invalid_height_format
- test_height_less10
- test_height_more200
- test_volume_price
- test_invalid_weight_format
- test_weight_less_minimum
- test_weight_more_maximum
- test_weight_price
- test_invalid_post_type
- test_regular_type_price
- test_Xpress_type_price (failed)

===== FAILURES =====

def test_Xpress_type_price():
 post_type = "Xpress"
 price = postal_calc.calcPostTypePrice(post_type)
> assert price == 10.00
E assert None == 10.0

===== 1 failed, 21 passed in 0.27 seconds =====

Test 22 Succeed:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, specifically the `test_Xpress_type_price` test which asserts that the price is `10.00`. The bottom window shows the `Test Results` tool window with a green checkmark next to each test name, indicating they all passed. The output pane shows the command-line logs for the pytest run, confirming 22 tests passed in 0.20 seconds.

```
def test_Xpress_type_price():
    post_type = "Xpress"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 10.00

All 22 tests passed - 25ms
```

Testing started at 2:40 PM ...
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
test.py [100%]
===== 22 passed in 0.20 seconds =====
Process finished with exit code 0

Test 22 Code with changes:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, specifically the `calcPostTypePrice` function which now includes an `elif` clause for the "Xpress" case. The bottom window shows the `Test Results` tool window with a green checkmark next to each test name, indicating they all passed. The output pane shows the command-line logs for the pytest run, confirming 22 tests passed in 0.20 seconds.

```
def calcPostTypePrice(post_type):
    if post_type == "Regular":
        return 5.00
    elif post_type == "Xpress":
        return 10.00

calcPostTypePrice() : elif post_type == "Xpress"
```

Testing started at 2:40 PM ...
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
test.py [100%]
===== 22 passed in 0.20 seconds =====
Process finished with exit code 0

Tests Passed: 22 passed (moments ago)

Test 23: Correct Priority post-type cost

Purpose of the test: Make sure that the “Priority” post type returns the correct additional cost

Test Name: test_priority_type_price

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 1 Priority

Expected Result: 15.00

Note: this is a test for the calcPostTypePrice method, which returns a correct fixed price. It was decided that the Priority post-type has a fixed price of \$15.00.

Test 23 Fail:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `test.py`, specifically focusing on two failing test cases: `test_Xpress_type_price()` and `test_priority_type_price()`. The `test_Xpress_type_price()` test fails because the expected price of 10.00 does not match the actual price of 15.00. The `test_priority_type_price()` test also fails with an assertion error. Below the code editor is the `Run` tool window, which shows the test results: 23 tests done, 1 failed. The bottom status bar indicates the current time as 159:26 and the file encoding as UTF-8.

```
def test_Xpress_type_price():
    post_type = "Xpress"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 10.00

def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 15.00
```

```
23 tests done: 1 failed - 25ms
test.py .....F
test.py:155 (test_priority_type_price)
15.0 != None
Expected :None
Actual   :15.0
<click to see difference>
```

```
def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
>     assert price == 15.00
E     assert None == 15.0

test.py:159: AssertionError
=====
test_priority_type_price [100%]
```

```
===== FAILURES =====
test.py:159: test_priority_type_price
=====
def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
>     assert price == 15.00
E     assert None == 15.0

test.py:159: AssertionError
=====
1 failed, 22 passed in 0.28 seconds =====
Process finished with exit code 0
```

PEP 8: no newline at end of file

Test 23 Code:

The screenshot shows the PyCharm IDE interface. The top window displays the code for `postal_calc.py`, specifically the implementation of the `calcPostTypePrice` function. The function uses a switch statement to calculate the price based on the `post_type`. The `test_priority_type_price()` test in the `test.py` file fails because the function returns 10.00 for the "Priority" type instead of the expected 15.00. Below the code editor is the `Run` tool window, which shows the test results: 23 tests done, 1 failed. The bottom status bar indicates the current time as 117:21 and the file encoding as UTF-8.

```
def calcPostTypePrice(post_type):
    if post_type == "Regular":
        return 5.00
    elif post_type == "Xpress":
        return 10.00
```

```
23 tests done: 1 failed - 25ms
test.py .....F
test.py:155 (test_priority_type_price)
15.0 != None
Expected :None
Actual   :15.0
<click to see difference>
```

```
def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
>     assert price == 15.00
E     assert None == 15.0

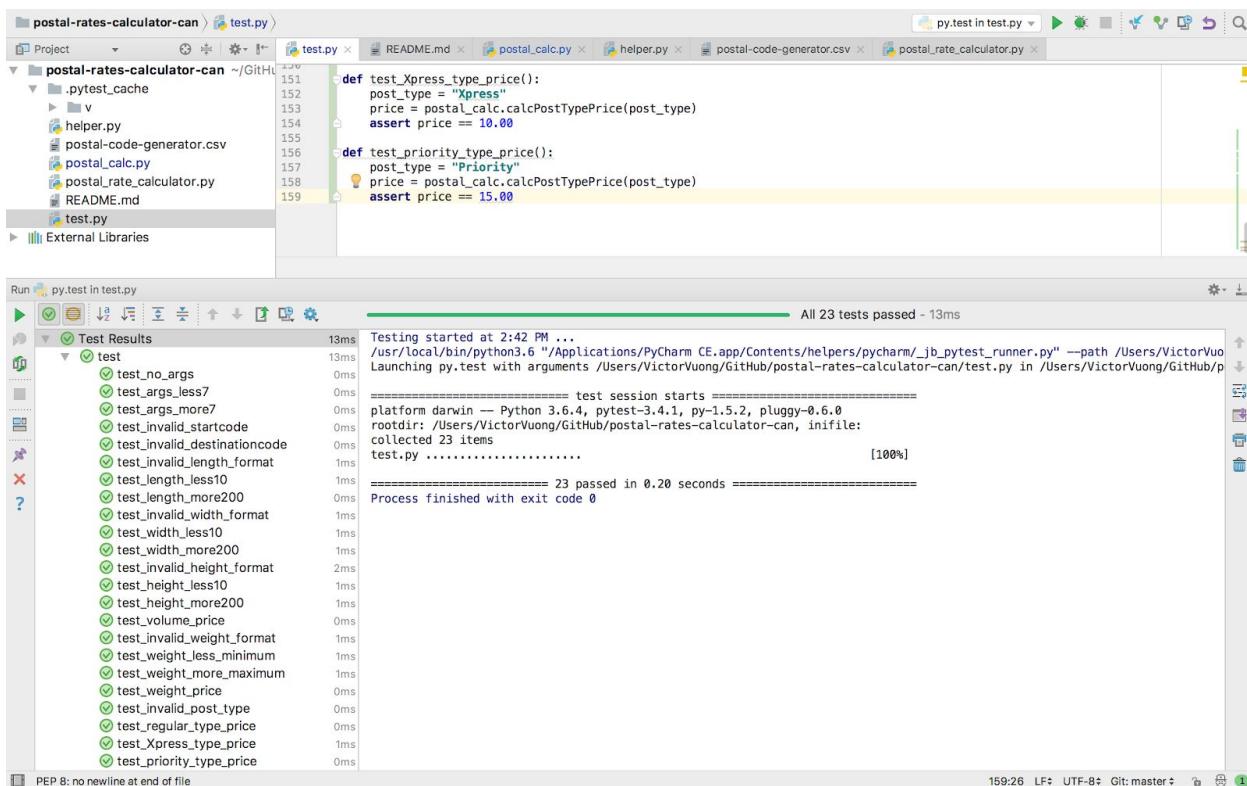
test.py:159: AssertionError
=====
test_priority_type_price [100%]
```

```
===== FAILURES =====
test.py:159: test_priority_type_price
=====
def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
>     assert price == 15.00
E     assert None == 15.0

test.py:159: AssertionError
=====
1 failed, 22 passed in 0.28 seconds =====
Process finished with exit code 0
```

Tests Failed: 22 passed, 1 failed (moments ago)

Test 23 Succeed:



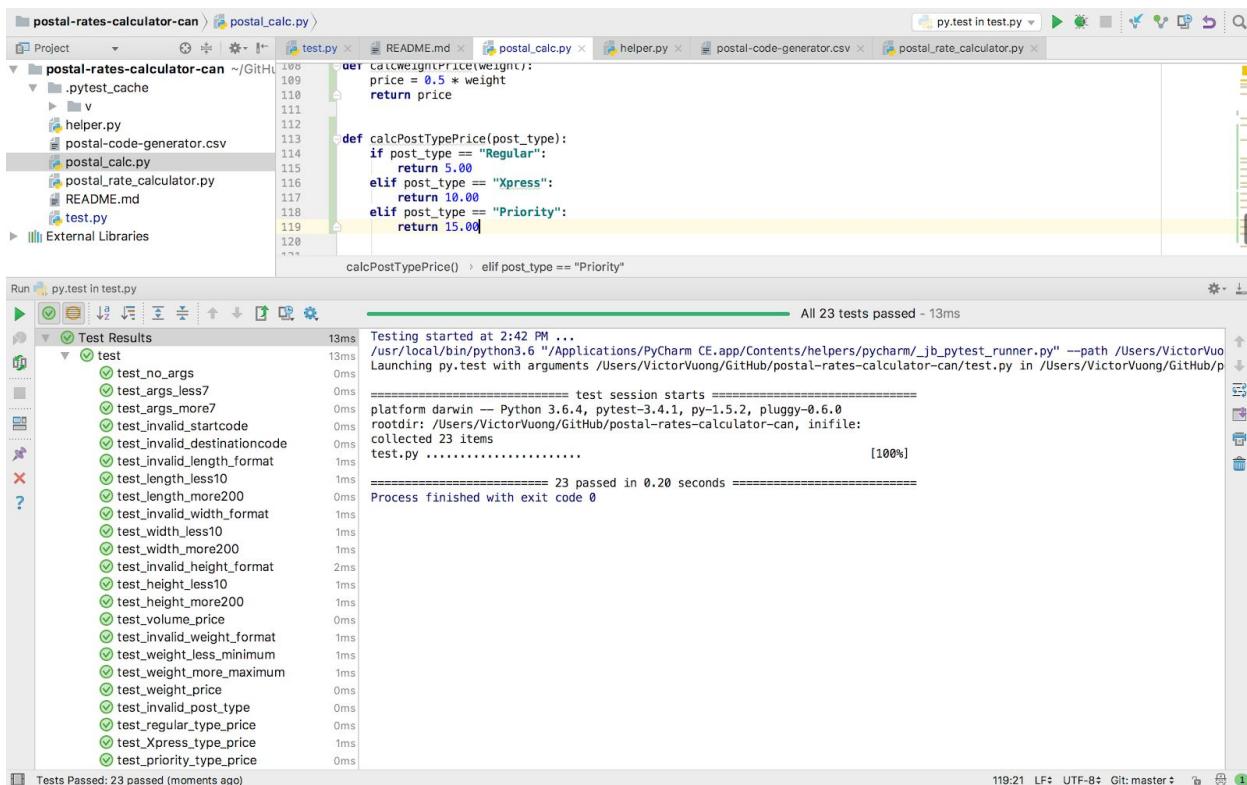
The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'test.py' and other files like 'README.md', 'postal_calc.py', 'helper.py', 'postal_code-generator.csv', and 'postal_rate_calculator.py'. The left sidebar shows a project tree with files like '.pytest_cache', 'v', 'helper.py', 'postal-code-generator.csv', 'postal_calc.py', 'postal_rate_calculator.py', and 'README.md'. The main code editor window contains Python test code. Below it, the 'Run' tool window shows 'Test Results' for 'test.py', indicating 'All 23 tests passed - 13ms'. The terminal window at the bottom shows the command line output of the pytest run, confirming all tests passed.

```
def test_Xpress_type_price():
    post_type = "Xpress"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 10.00

def test_priority_type_price():
    post_type = "Priority"
    price = postal_calc.calcPostTypePrice(post_type)
    assert price == 15.00
```

```
Testing started at 2:42 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 23 items
test.py ..... [100%]
=====
23 passed in 0.20 seconds =====
Process finished with exit code 0
```

Test 23 Code with changes:



The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'postal_calc.py' and other files like 'test.py' and 'README.md'. The left sidebar shows a project tree with files like '.pytest_cache', 'v', 'helper.py', 'postal-code-generator.csv', 'postal_rate_calculator.py', and 'test.py'. The main code editor window contains Python code for calculating postage prices based on weight and type. Below it, the 'Run' tool window shows 'Test Results' for 'test.py', indicating 'All 23 tests passed - 13ms'. The terminal window at the bottom shows the command line output of the pytest run, confirming all tests passed.

```
def calcPostTypePrice(weight):
    price = 0.5 * weight
    return price

def calcPostTypePrice(post_type):
    if post_type == "Regular":
        return 5.00
    elif post_type == "Xpress":
        return 10.00
    elif post_type == "Priority":
        return 15.00
```

```
Testing started at 2:42 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 23 items
test.py ..... [100%]
=====
23 passed in 0.20 seconds =====
Process finished with exit code 0
```

Test 24: Correct post rate cost from calculator (program)

Test Name: test_total_post_rate_cost

Call setup: postal_calc.py h4k2g2 v9g8r7 100 100 100 1 Xpress

Expected Result: 15.50

Purpose of the test: Make sure that the post rate calculator program computes the right cost with all the correct arguments (safely assumed with all previous tests) entered by the user

Note: This test checks if the main method correctly uses the previously tested functions

Test 24 Fail:

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** postal-rates-calculator-can
- File:** test.py
- Code (test.py):**

```
def test_total_post_rate_cost():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "15.50"
```
- Run Results:** 24 tests done: 1 failed - 17ms
- Test Results:** A detailed log of the failed test:

```
test.py:161 (test_total_post_rate_cost)
15.50 != None
Expected :None
Actual   :15.50
<Click to see difference>
```

```
def test_total_post_rate_cost():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>   assert return_rate == "15.50"
E   AssertionError: assert None == '15.50'
```

```
test.py:166: AssertionError
[100%]
```

```
===== FAILURES =====
test_total_post_rate_cost
```

```
def test_total_post_rate_cost():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
>   assert return_rate == "15.50"
E   AssertionError: assert None == '15.50'
```

```
test.py:166: AssertionError
===== 1 failed, 23 passed in 0.22 seconds =====
```
- Bottom Status:** Process finished with exit code 0

Test 24 Code:

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** postal-rates-calculator-can
- File:** postal_calc.py
- Code (postal_calc.py):**

```
post_type = argv[6]

if post_type != "Regular" and post_type != "Xpress" and post_type != "Priority":
    display_text = "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
    return display_text
```
- Run Results:** 24 tests done: 1 failed - 17ms
- Test Results:** A detailed log of the failed test, identical to the one in the previous screenshot.
- Bottom Status:** Tests Failed: 23 passed, 1 failed (moments ago)

Test 24 Succeed:

PyCharm IDE screenshot showing the project structure and a successful test run. The test results show all 24 tests passed in 0.32 seconds.

```
assert price == 15.00
def test_total_post_rate_cost():
    parameter_count = 7
    parameters = ["h4k2g2", "v9g8r7", "100", "100", "1", "Xpress"]
    return_rate = postal_calc.main(parameter_count, parameters)
    assert return_rate == "15.50"

Testing started at 3:04 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 24 items
test.py ..... [100%]
=====
24 passed in 0.32 seconds
Process finished with exit code 0
```

Test 24 Code with changes:

PyCharm IDE screenshot showing the project structure and a successful test run after making changes to the code. The test results show all 24 tests passed in 0.32 seconds.

```
if post_type != "Regular" and post_type != "Xpress" and post_type != "Priority":
    display_text = "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
    return display_text

total_cost = "{0:.2f}".format(calcVolumePrice(length,width,height) + calcWeightPrice(weight) + calcPostTypePrice(post_type))
return total_cost
```

```
Testing started at 3:04 PM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/postal-rates-calculator-can
platform darwin -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /Users/VictorVuong/GitHub/postal-rates-calculator-can, inifile:
collected 24 items
test.py ..... [100%]
=====
24 passed in 0.32 seconds
Process finished with exit code 0
```

Source code

The section below contains screenshots of the complete source code of the post rate calculator developed using TDD as described above.

```
1 import sys
2 import csv
3
4 #main method that will run the post rate calculator, argc is the parameters count and argv contains the parameters
5 def main(argc, argv):
6
7     #CHECK CORRECT NUMBER OF ARGUMENTS
8     if (argc < 7 or argc > 7):
9         display_text = "Usage: starting_code ending_code length width height weight post_type"
10        return display_text    #terminate with error display string
11
12     #CHECK VALID POSTAL CODES
13     with open('postal-code-generator.csv') as csvfile:
14         readCSV = csv.reader(csvfile, delimiter=',')
15         codes = []
16
17         for row in readCSV:
18             code = row[0]
19             codes.append(code)
20
21     if argv[0] not in codes:
22         display_text = "Error: not a valid canadian starting postal code."
23         return display_text
24
25     if argv[1] not in codes:
26         display_text = "Error: not a valid canadian destination postal code."
27         return display_text
28
29     #CHECK VALID LENGTH
30     try:
31         length = float(argv[2])
32     except:
33         display_text = "Error: not a valid numerical length (cm)."
34         return display_text
35
36     if length < 10:
37         display_text = "Error: length must be at least 10cm."
38         return display_text
39
40     if length > 200:
41         display_text = "Error: maximum length is 200cm."
42         return display_text
```

```

44     #CHECK VALID WIDTH
45     try:
46         width = float(argv[3])
47     except:
48         display_text = "Error: not a valid numerical width (cm)."
49         return display_text
50
51     if width < 10:
52         display_text = "Error: width must be at least 10cm."
53         return display_text
54
55     if width > 200:
56         display_text = "Error: maximum width is 200cm."
57         return display_text
58
59     #CHECK VALID HEIGHT
60     try:
61         height = float(argv[4])
62
63     except:
64         display_text = "Error: not a valid numerical height (cm)."
65         return display_text
66
67     if height < 10:
68         display_text = "Error: height must be at least 10cm."
69         return display_text
70
71     if height > 200:
72         display_text = "Error: maximum height is 200cm."
73         return display_text
74
75     #CHECK VALID WEIGHT
76     try:
77         weight = float(argv[5])
78     except:
79         display_text = "Error: not a valid numerical weight (kg)."
80         return display_text
81
82     if weight < 0.1:
83         display_text = "Error: minimum weight is 0.1kg."
84         return display_text
85
86     if weight > 30:
87         display_text = "Error: maximum weight is 30kg."
88         return display_text
89
90
91     #CHECK VALID POST TYPE
92     post_type = argv[6]
93
94     if post_type != "Regular" and post_type != "Xpress" and post_type != "Priority":
95         display_text = "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
96         return display_text
97
98     #CALCULATE TOTAL POST RATE
99     total_cost = "{0:.2f}".format(calcVolumePrice(length,width,height) + calcWeightPrice(weight) + calcPostTypePrice(post_type))
100
101     return total_cost
102
103
104     #function to calculate the volume price (assumed to be 5 dollars per meters cube)
105     def calcVolumePrice(length, width, height):
106         price = (0.01 * length * 0.01 * width * 0.01 * height) * 5
107         return price
108
109
110     #function to calculate the weight price (assumed to be 50 cents per kilogram)
111     def calcWeightPrice(weight):
112         price = 0.5 * weight
113         return price
114
115
116     #function to return the correct price of the post type (assumed to be 5 for regular, 10 for xpress, 15 for priority)
117     def calcPostTypePrice(post_type):
118         if post_type == "Regular":
119             return 5.00
120         elif post_type == "Xpress":
121             return 10.00
122         elif post_type == "Priority":
123             return 15.00

```

Source Code: Test

The screenshots below are snapshots of all the written tests for the functionality of the application.

```
1 import pytest
2 import postal_calc
3
4
5 def test_no_args():
6     parameter_count = 0
7     parameters = []
8     return_rate = postal_calc.main(parameter_count, parameters)
9     assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
10
11
12 def test_args_less7():
13     parameter_count = 3
14     parameters = ["h4k2g2" "h1t3r3" "3"]
15     return_rate = postal_calc.main(parameter_count, parameters)
16     assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
17
18
19 def test_args_more7():
20     parameter_count = 8
21     parameters = ["h4k2g2" "h3l1o0" "1" "2" "3" "4" "Xpress"]
22     return_rate = postal_calc.main(parameter_count, parameters)
23     assert return_rate == "Usage: starting_code ending_code length width height weight post_type"
24
25
26 def test_invalid_startcode():
27     parameter_count = 7
28     parameters = ["123456", "654321", "1", "2", "3", "4", "Xpress"]
29     return_rate = postal_calc.main(parameter_count, parameters)
30     assert return_rate == "Error: not a valid canadian starting postal code."
31
32
33 def test_invalid_destinationcode():
34     parameter_count = 7
35     parameters = ["h4k2g2", "654321", "1", "2", "3", "4", "Xpress"]
36     return_rate = postal_calc.main(parameter_count, parameters)
37     assert return_rate == "Error: not a valid canadian destination postal code."
38
39
40 def test_invalid_length_format():
41     parameter_count = 7
42     parameters = ["h4k2g2", "v9g8r7", "abc", "2", "3", "4", "Xpress"]
43     return_rate = postal_calc.main(parameter_count, parameters)
44     assert return_rate == "Error: not a valid numerical length (cm)."
```

```

89     def test_height_less10():
90         parameter_count = 7
91         parameters = ["h4k2g2", "v9g8r7", "50", "50", "0", "4", "Xpress"]
92         return_rate = postal_calc.main(parameter_count, parameters)
93         assert return_rate == "Error: height must be at least 10cm."
94
95
96     def test_height_more200():
97         parameter_count = 7
98         parameters = ["h4k2g2", "v9g8r7", "50", "50", "400", "4", "Xpress"]
99         return_rate = postal_calc.main(parameter_count, parameters)
100        assert return_rate == "Error: maximum height is 200cm."
101
102
103    def test_volume_price():
104        length = 100
105        width = 100
106        height = 100
107        price = postal_calc.calcVolumePrice(length, width, height)
108        assert price == 5.00
109
110
111    def test_invalid_weight_format():
112        parameter_count = 7
113        parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "abc", "Xpress"]
114        return_rate = postal_calc.main(parameter_count, parameters)
115        assert return_rate == "Error: not a valid numerical weight (kg)."
116
117
118    def test_weight_less_minimum():
119        parameter_count = 7
120        parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "0", "Xpress"]
121        return_rate = postal_calc.main(parameter_count, parameters)
122        assert return_rate == "Error: minimum weight is 0.1kg."
123
124
125    def test_weight_more_maximum():
126        parameter_count = 7
127        parameters = ["h4k2g2", "v9g8r7", "50", "50", "50", "31", "Xpress"]
128        return_rate = postal_calc.main(parameter_count, parameters)
129        assert return_rate == "Error: maximum weight is 30kg."
130
131
132    def test_weight_price():
133        weight = 3
134        price = postal_calc.calcWeightPrice(weight)
135        assert price == 1.5
136
137
138    def test_invalid_post_type():
139        parameter_count = 7
140        parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "hello"]
141        return_rate = postal_calc.main(parameter_count, parameters)
142        assert return_rate == "Error: not a valid post type (Regular, Xpress or Priority) case sensitive."
143
144
145    def test_regular_type_price():
146        post_type = "Regular"
147        price = postal_calc.calcPostTypePrice(post_type)
148        assert price == 5.00
149
150
151    def test_Xpress_type_price():
152        post_type = "Xpress"
153        price = postal_calc.calcPostTypePrice(post_type)
154        assert price == 10.00
155
156
157    def test_priority_type_price():
158        post_type = "Priority"
159        price = postal_calc.calcPostTypePrice(post_type)
160        assert price == 15.00
161
162
163    def test_total_post_rate_cost():
164        parameter_count = 7
165        parameters = ["h4k2g2", "v9g8r7", "100", "100", "100", "1", "Xpress"]
166        return_rate = postal_calc.main(parameter_count, parameters)
167        assert return_rate == "15.50"

```