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:

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 performed in PyCharm IDE for Python programming language.

Pytest framework will be used 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 and the development process. First, a list of postal code will be generated from a simple CSV file, where the format will only consist of existing Canadian postal code. 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:

$$postal\ parcel\ price\ =\ weight\ price\ +\ volume\ price\ +\ post\ type\ price$$

The dimensions are inspired from the actual canada post parcel rate rules:
https://www.canadapost.ca/cpotools/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):

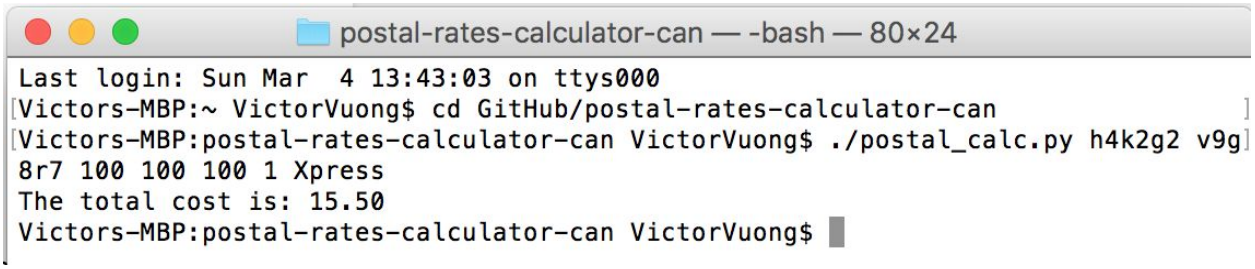
- Postal rate price based on Canada Post parcel Rate rules with slight modifications.

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.

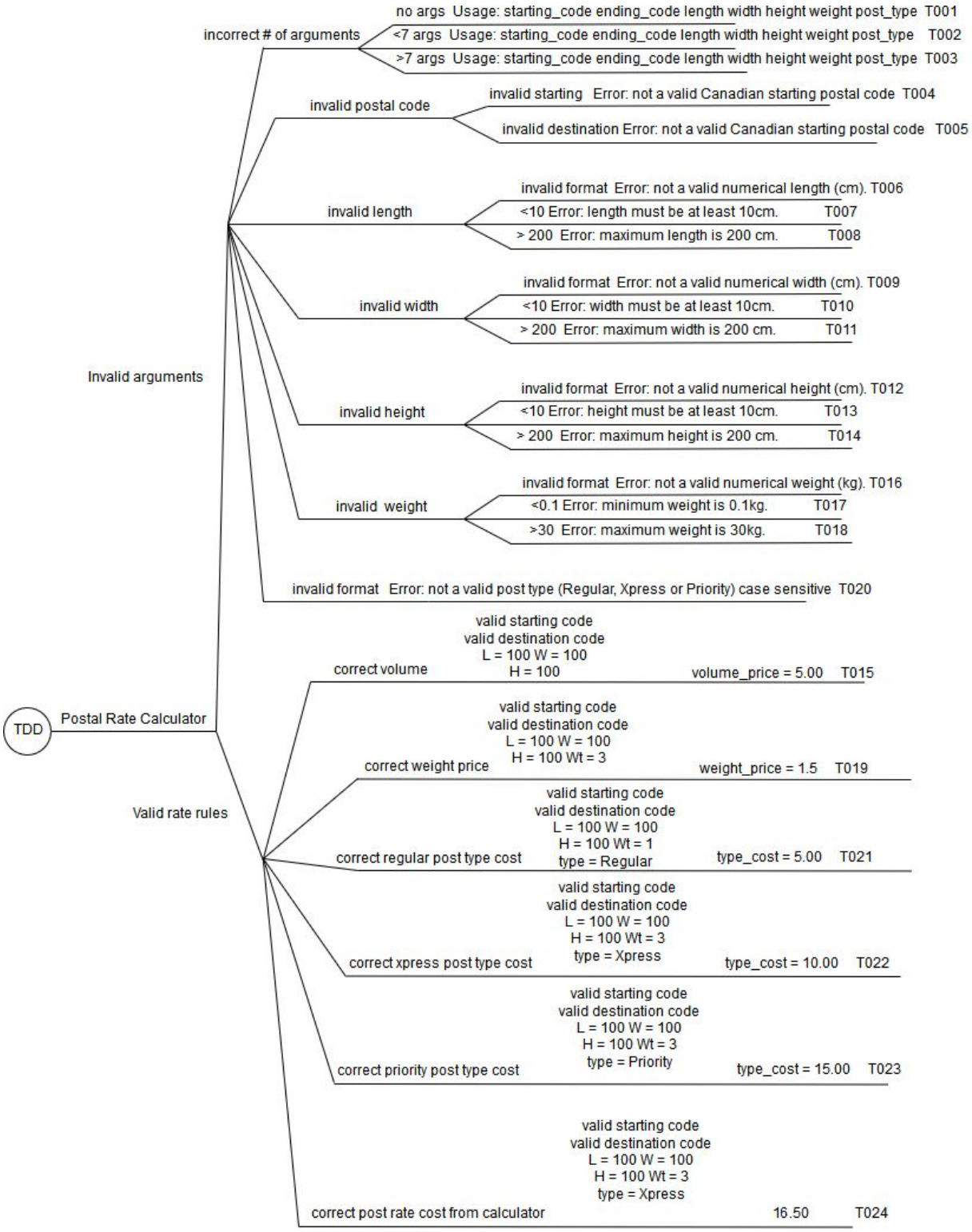
```
code
h4k2g2
v9g8r7
g3h0j9
g1k8n2
e9cm1j
t8rl6b
p7k6b2
t7sh4h
v1r7b0
j5k5n1
e5n9m1
t9v7m1
k7h3v1
g7b1y7
e6g2k2
t1w8n6
s4a9m3
g8j4o2
s9v8c3
l3m0j2
k7v0e8
t4l3j8
v9g0n2
e5s3j8
j5x0n1
v3a6k0
v3a7v3
n3y9y3
g5x0m8
```

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:



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.

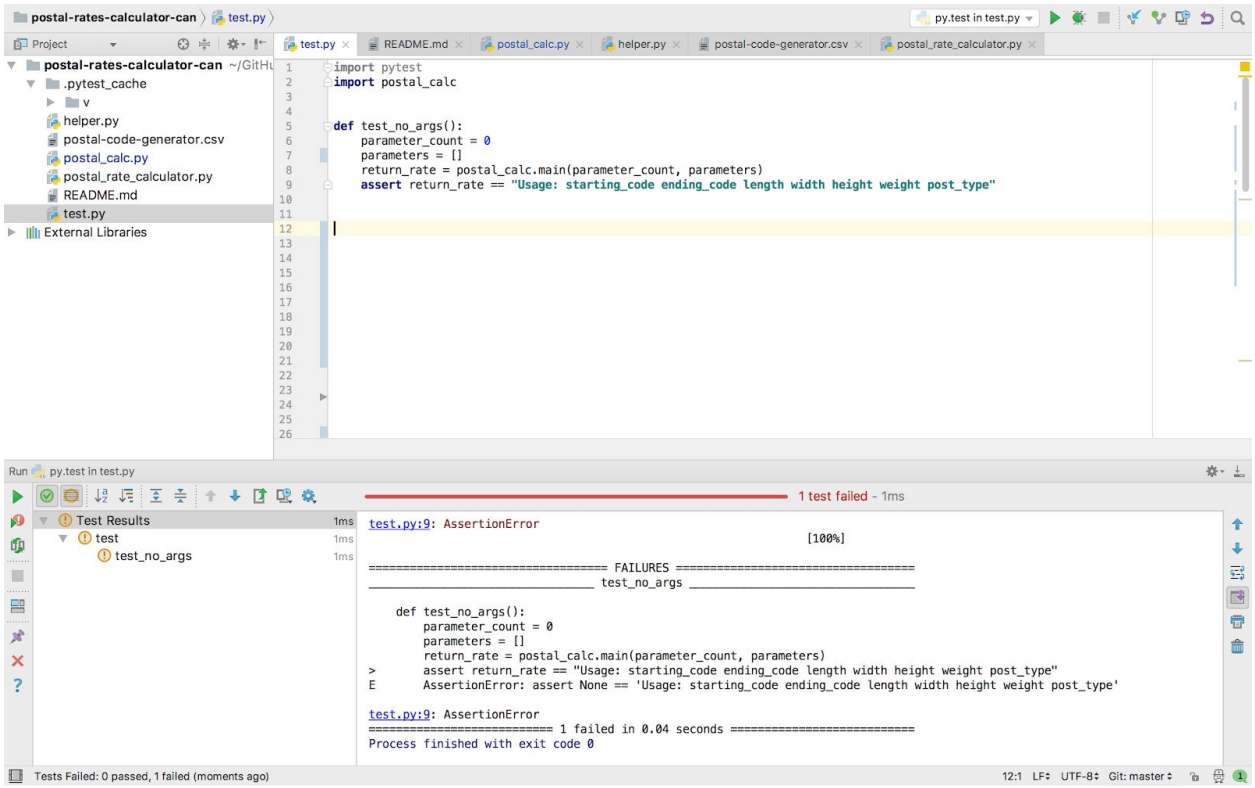
Screenshots and Implementations
Every test below is a step towards the final postal rate calculator program

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)

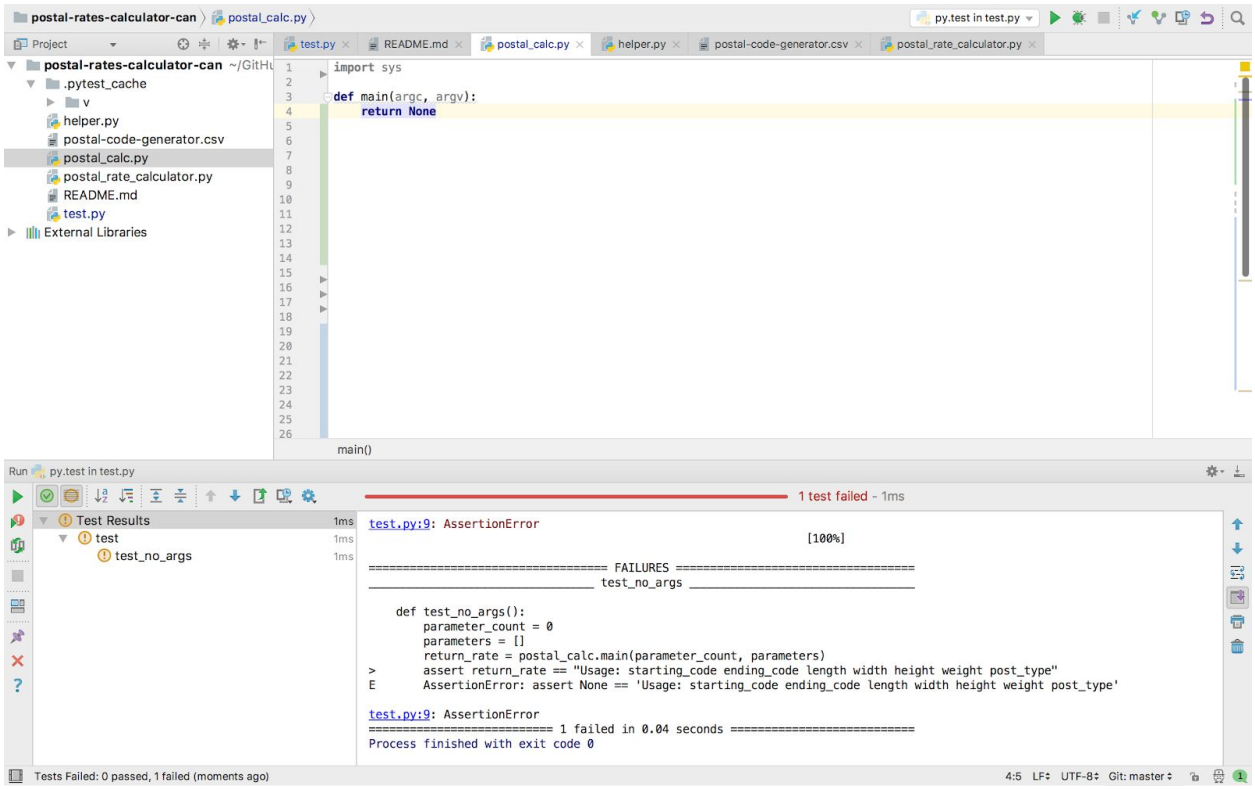
Test 1: No arguments

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.

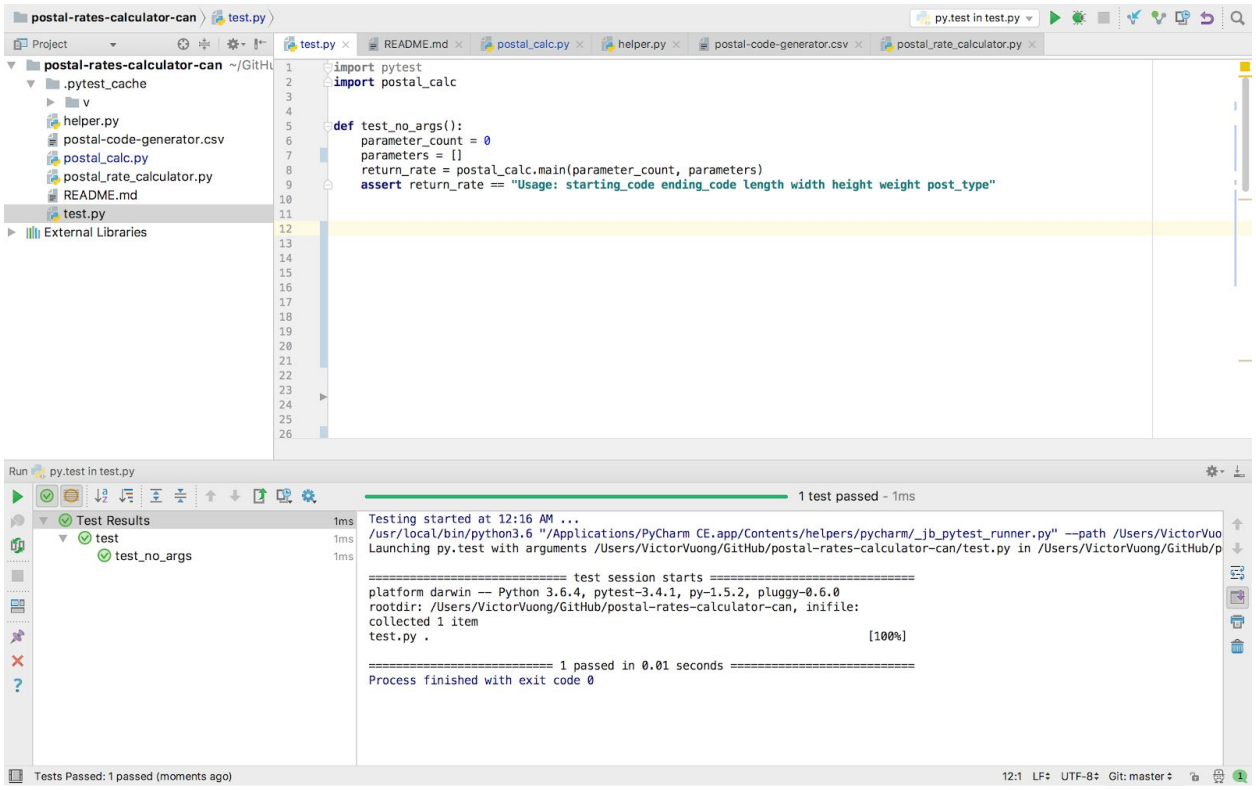
Test 1 Fail:



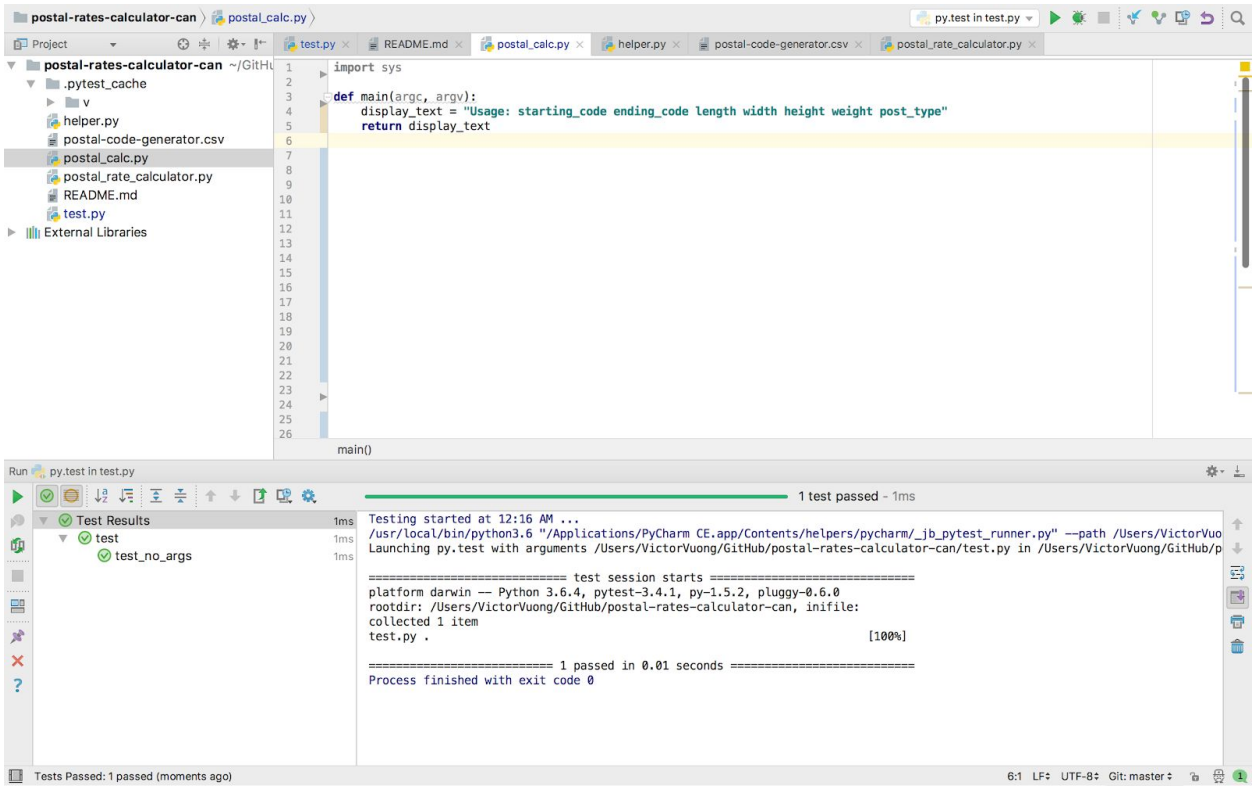
Test 1 Code:



Test 1 Success:



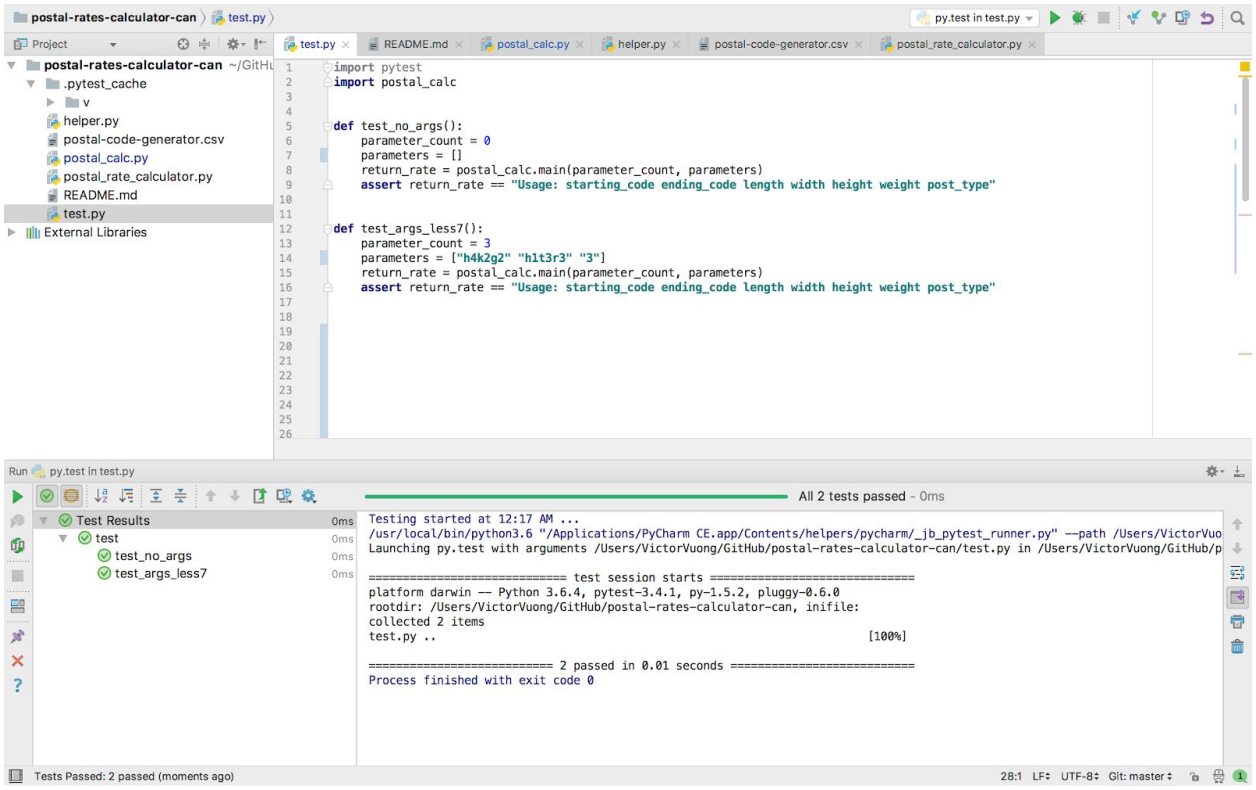
Test 1 Code with changes:



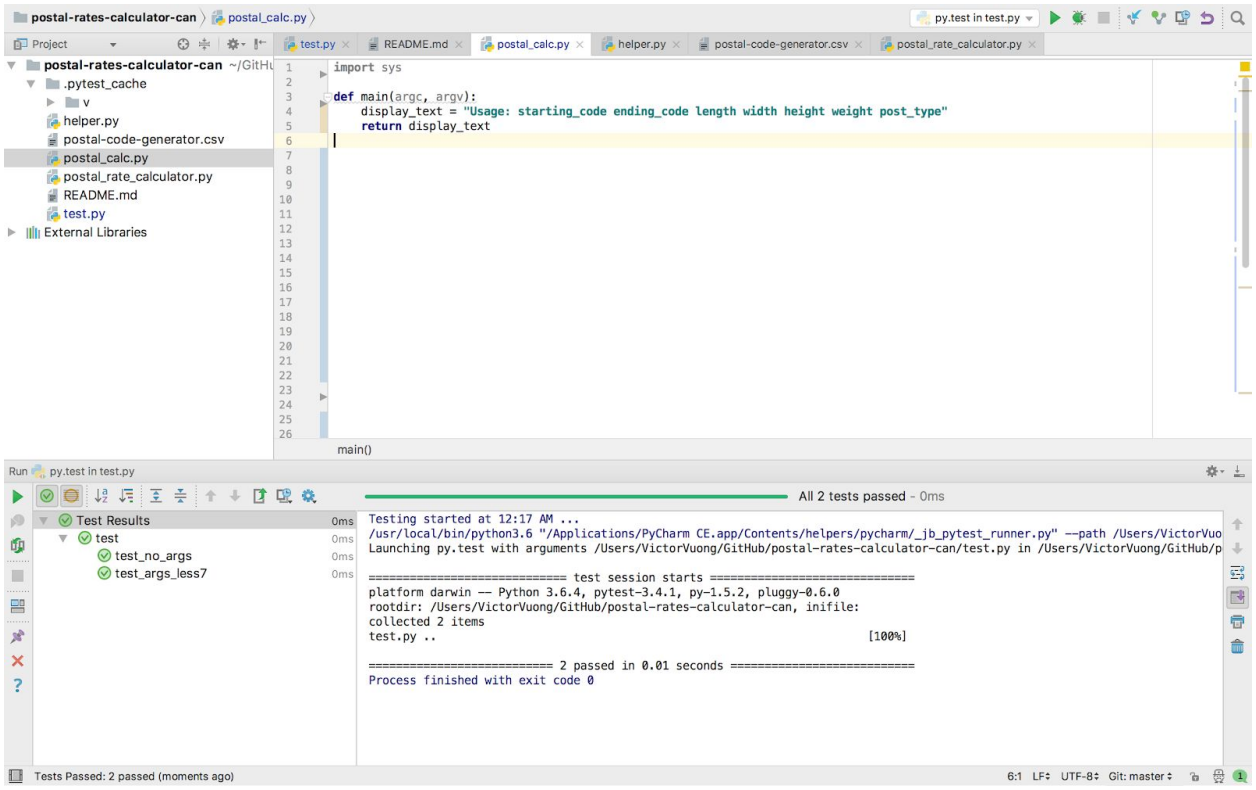
Test 2: Too few arguments

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:



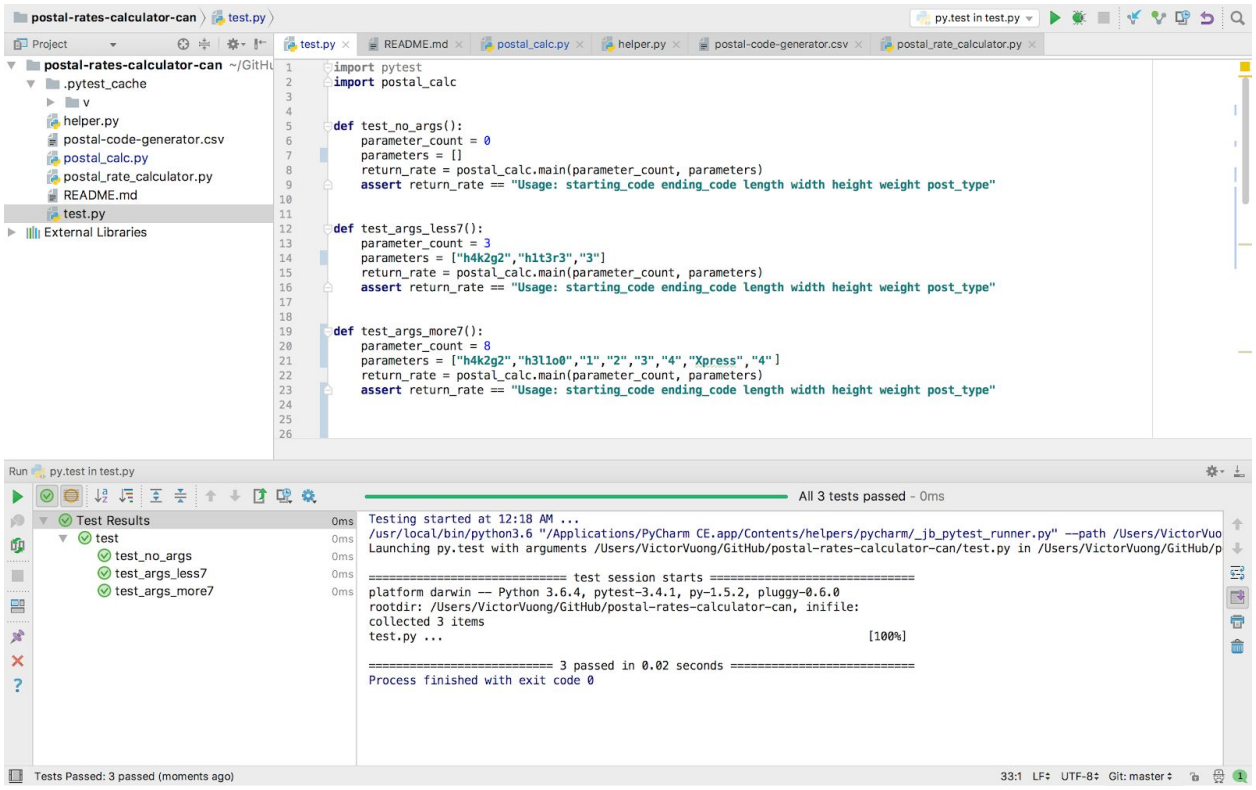
Test 2 Code with changes:



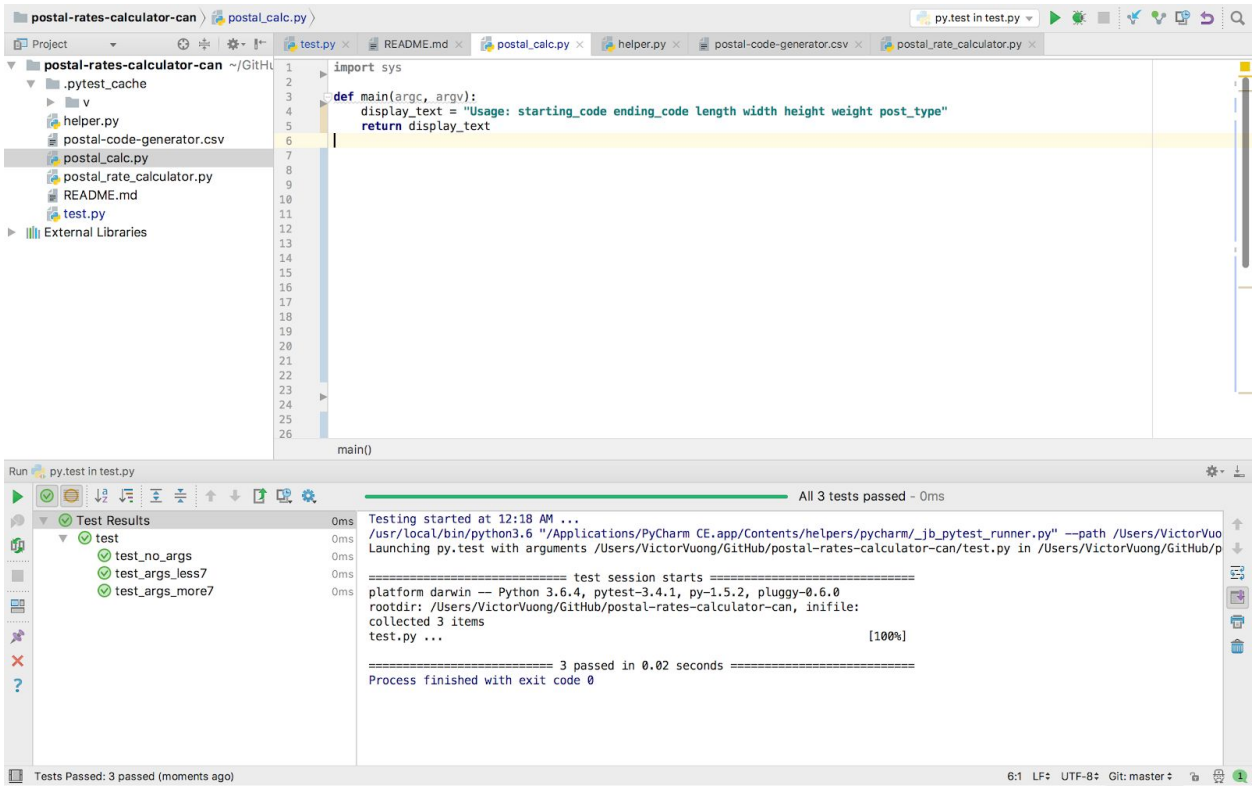
Test 3: Too many arguments

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:



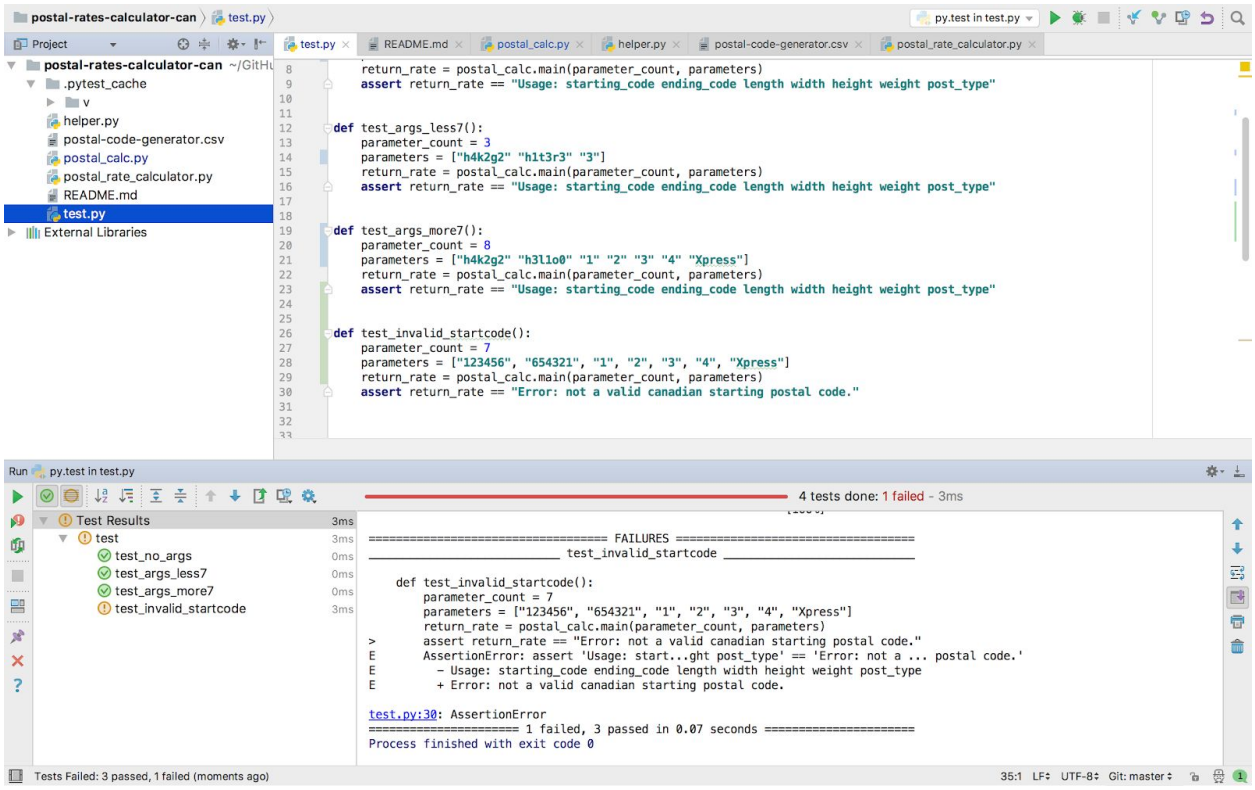
Test 3 Code with changes:



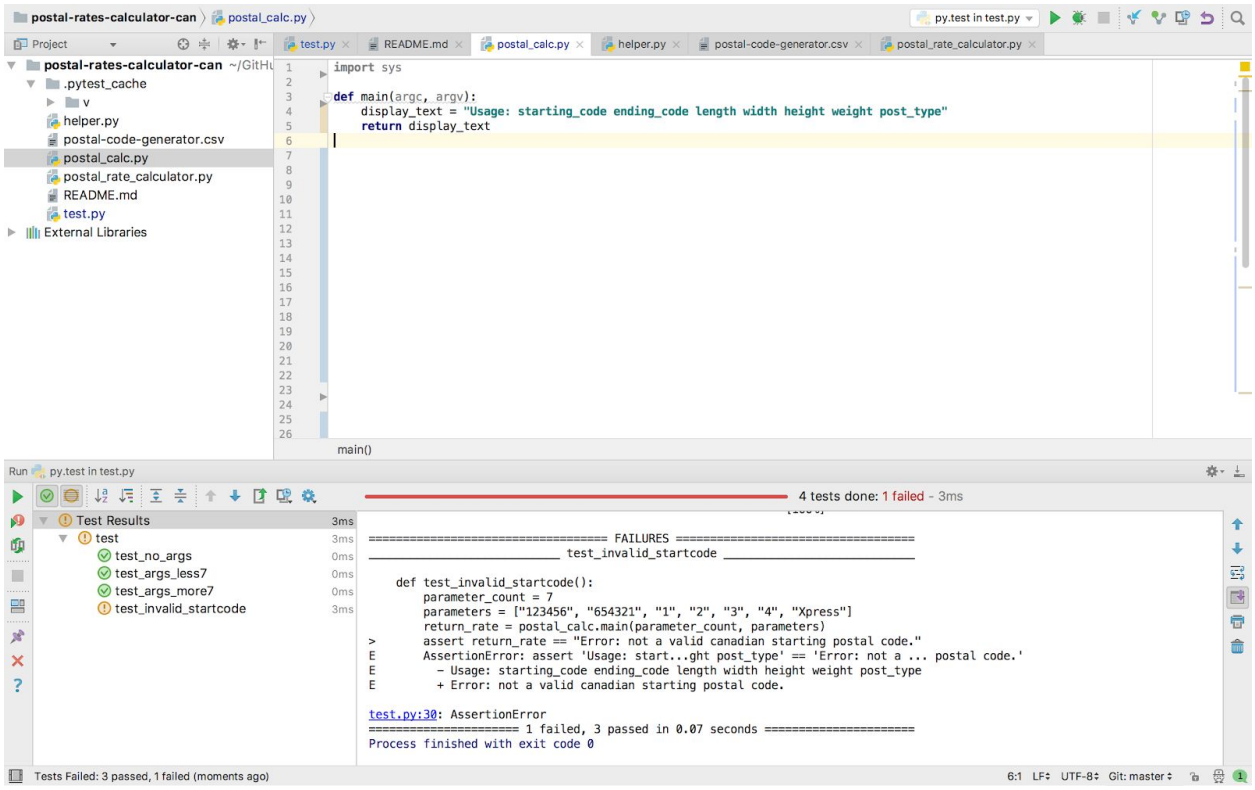
Test 4: Invalid starting postal code

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.

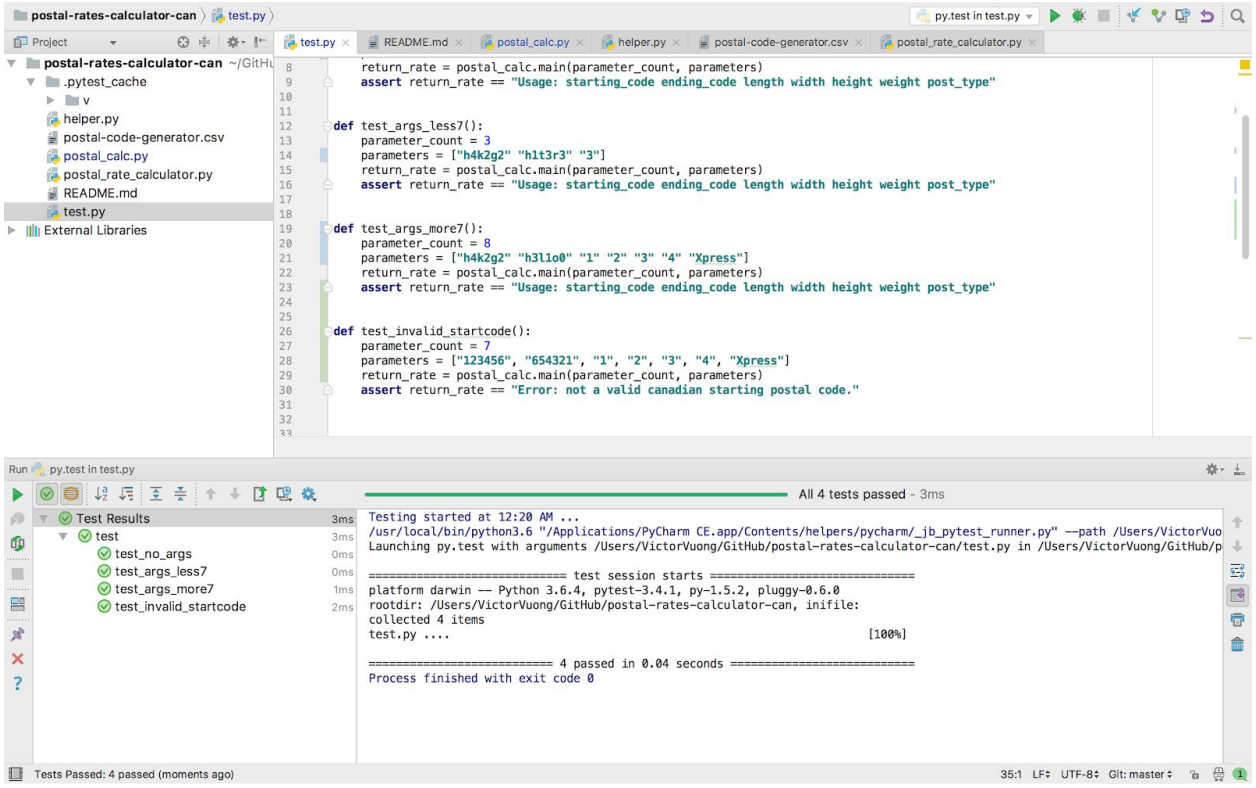
Test 4 Fail:



Test 4 Code:



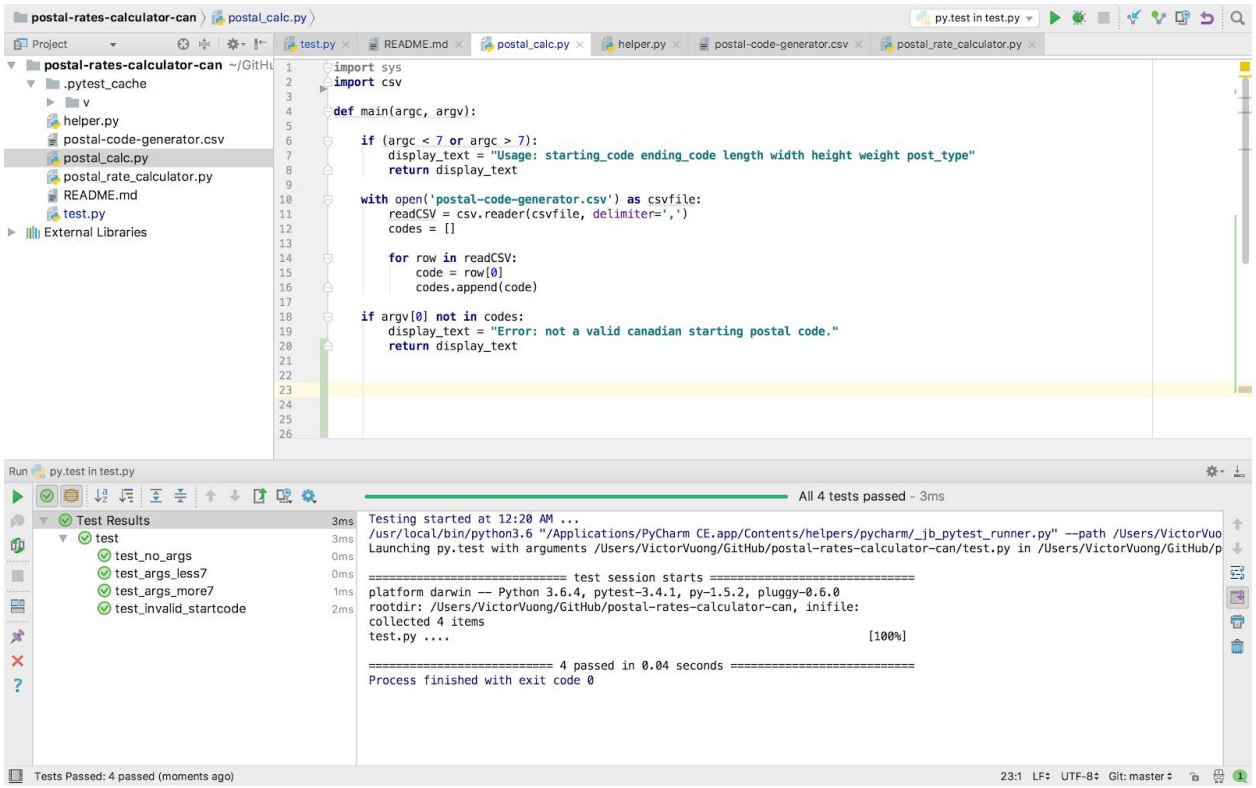
Test 4 Succeed:



```
    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."
```

```
Testing started at 12:20 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuo
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/p

============================= test session starts ==============================
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 4 items
test.py ....                                                             [100%]

========================== 4 passed in 0.04 seconds ============================
Process finished with exit code 0
```

Test 4 Code with changes:



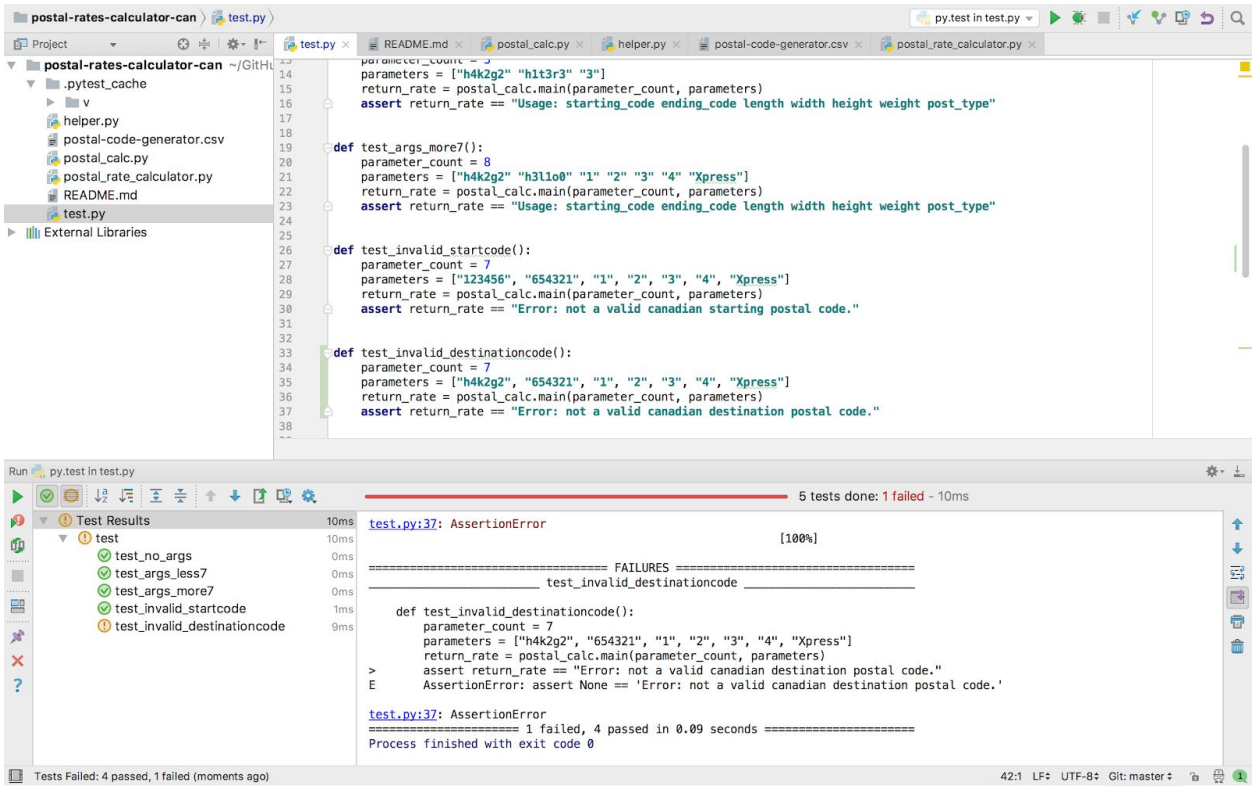
```
import sys
import csv

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
```

```
Testing started at 12:20 AM ...
/usr/local/bin/python3.6 "/Applications/PyCharm CE.app/Contents/helpers/pycharm/_jb_pytest_runner.py" --path /Users/VictorVuo
Launching py.test with arguments /Users/VictorVuong/GitHub/postal-rates-calculator-can/test.py in /Users/VictorVuong/GitHub/p

============================= test session starts ==============================
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 4 items
test.py ....                                                             [100%]

========================== 4 passed in 0.04 seconds ============================
Process finished with exit code 0
```

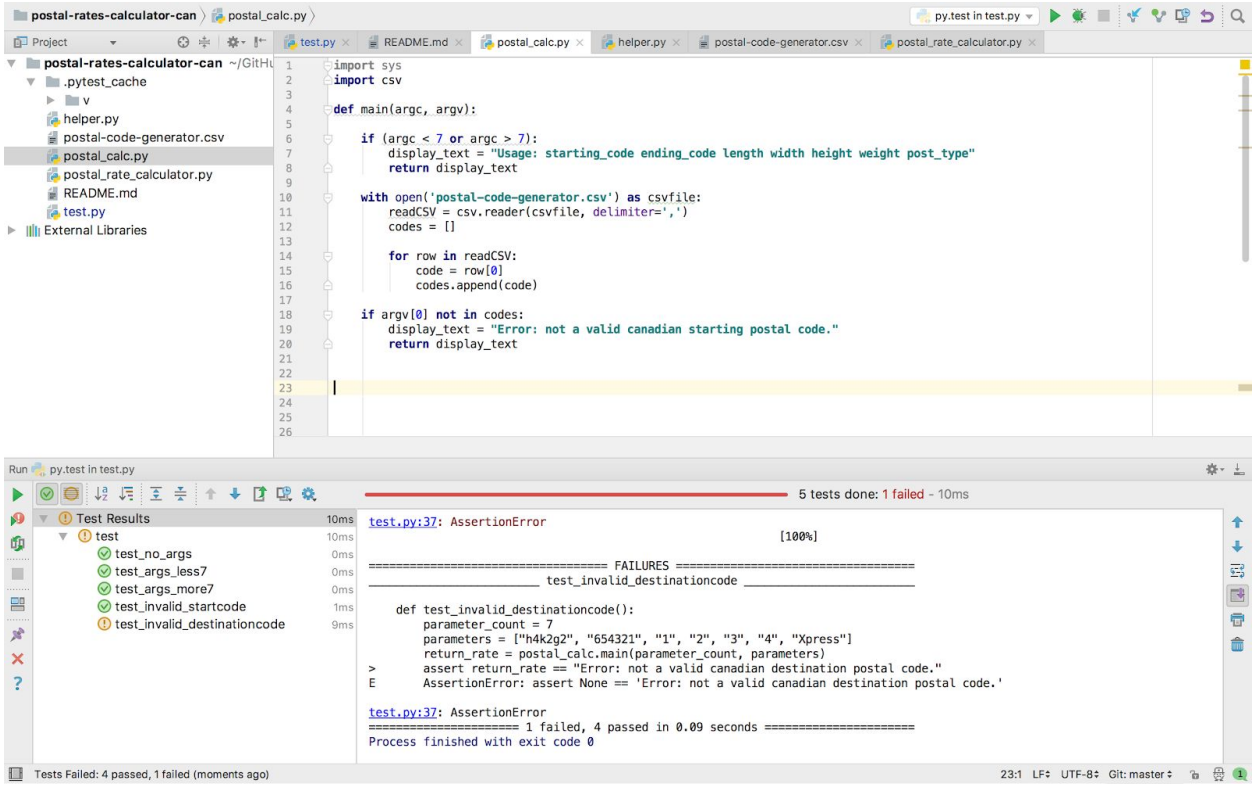
Test 5: Invalid destination postal code

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.

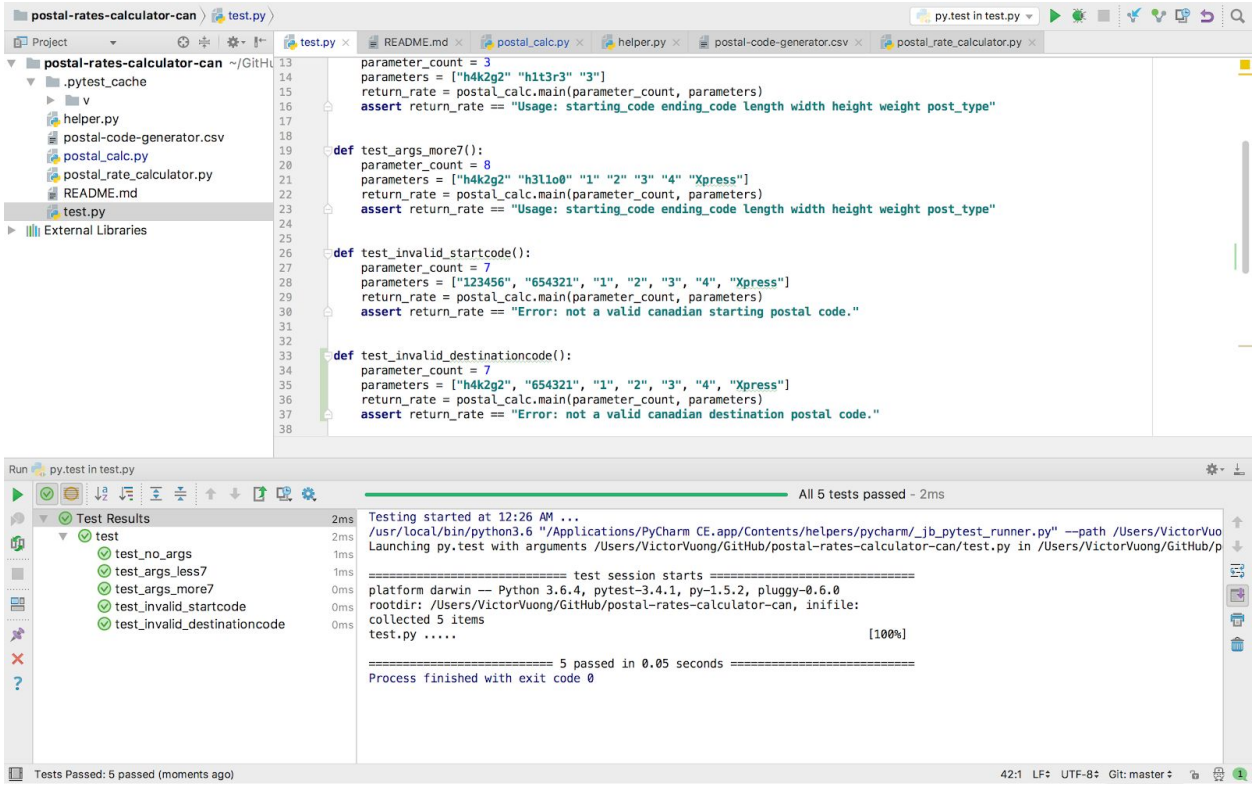
Test 5 Fail:



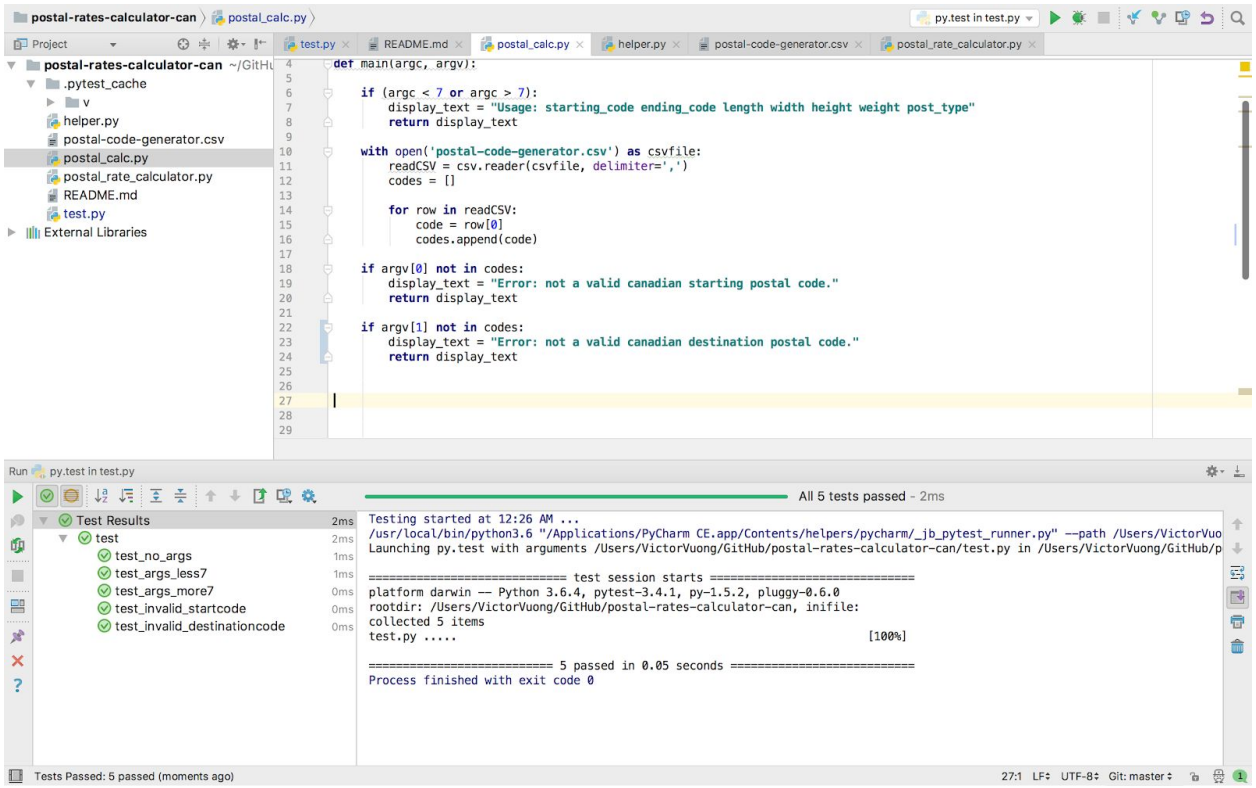
Test 5 Code:



Test 5 Succeed:



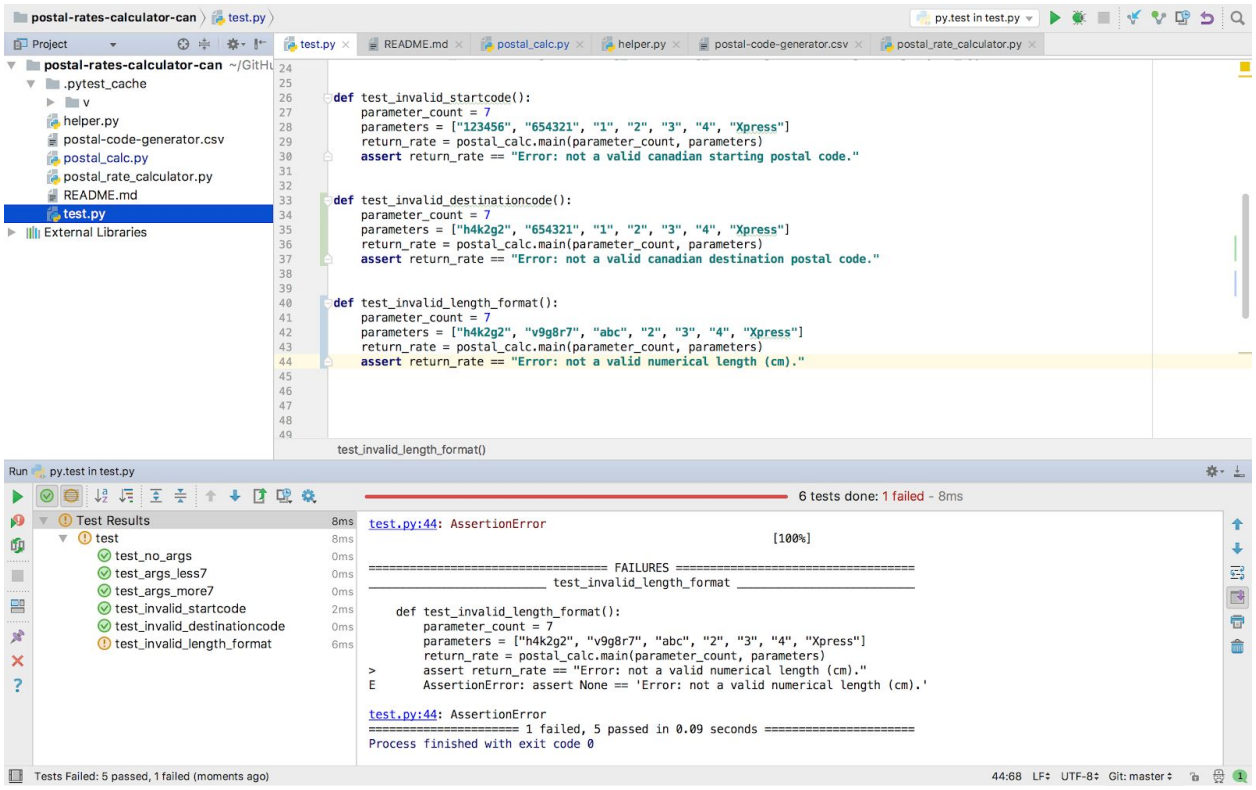
Test 5 Code with changes:



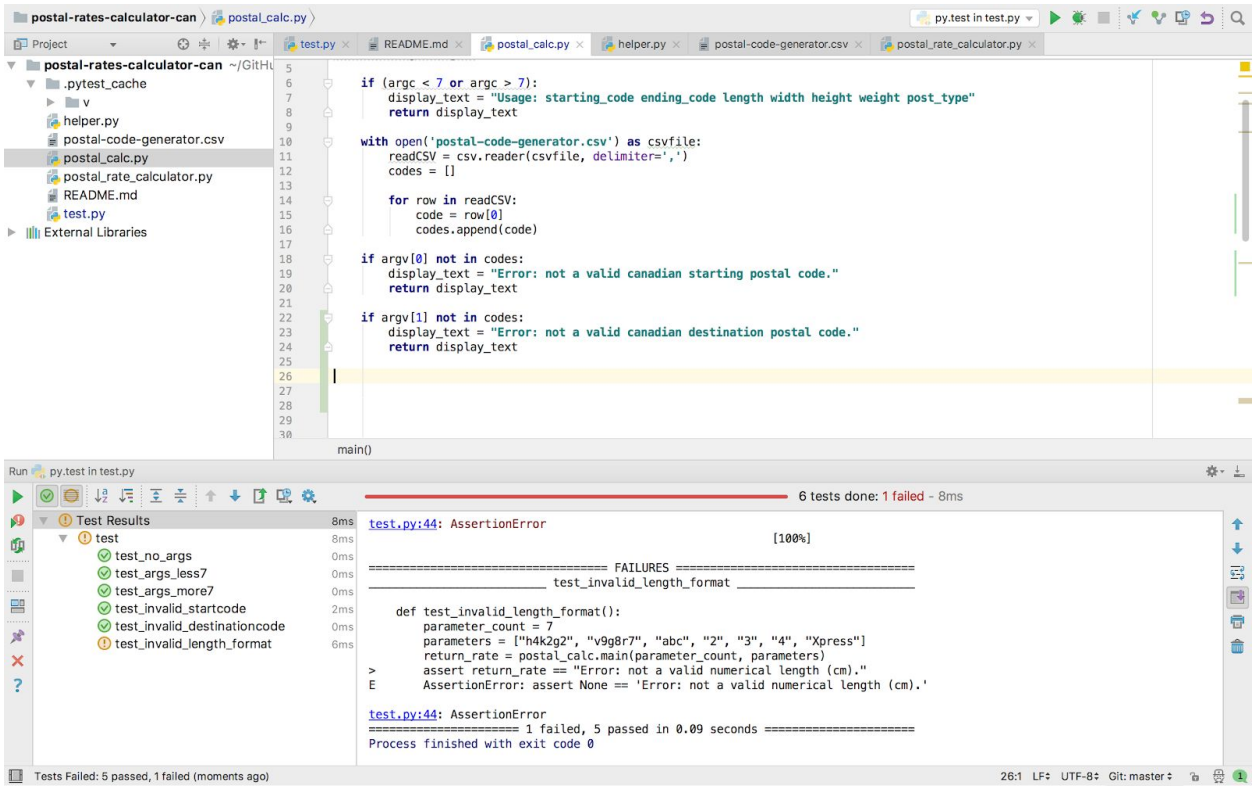
Test 6: Invalid length format

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.

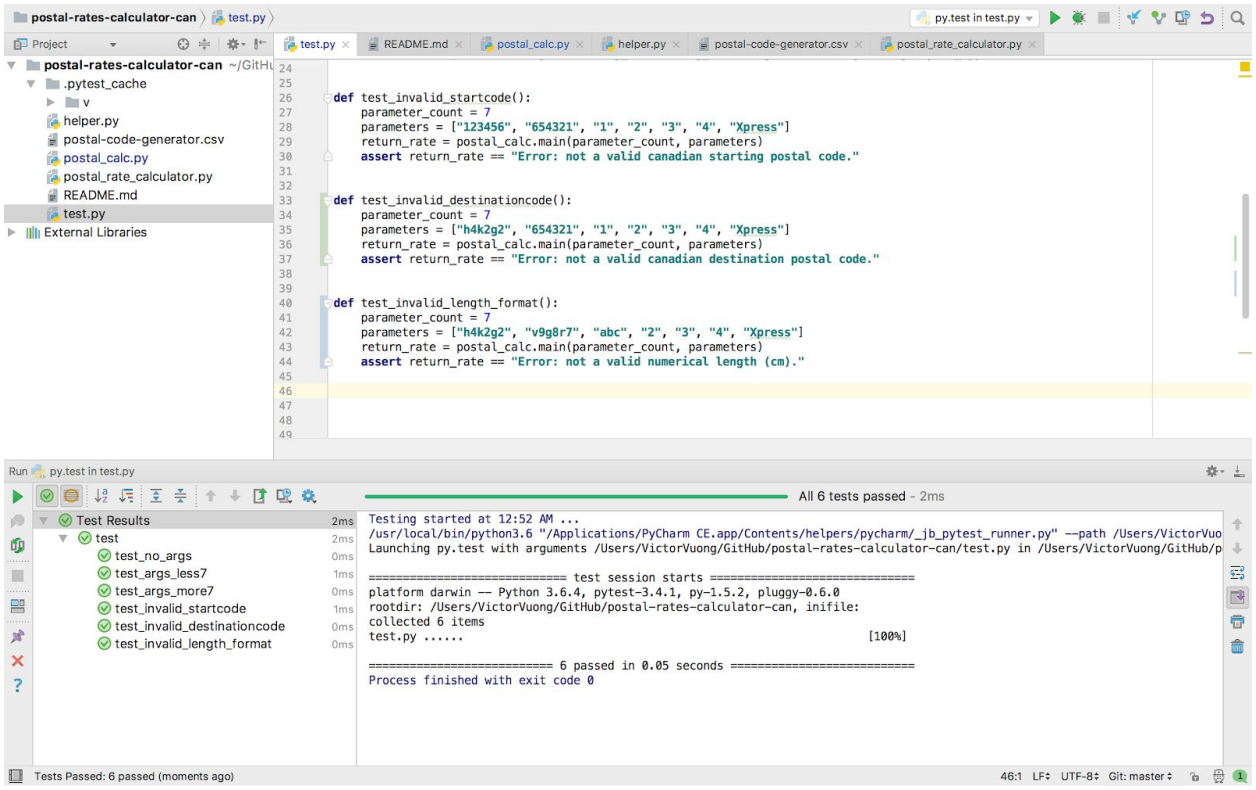
Test 6 Fail:



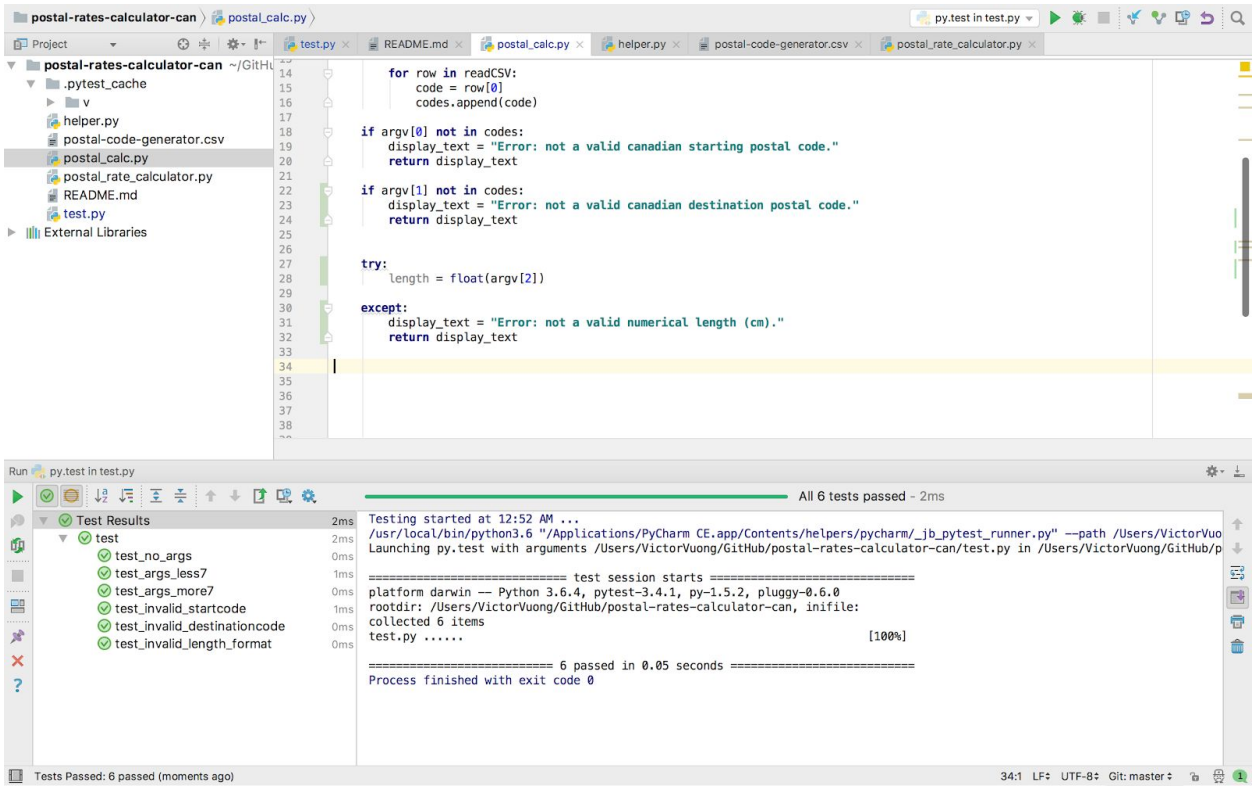
Test 6 Code:



Test 6 Succeed:



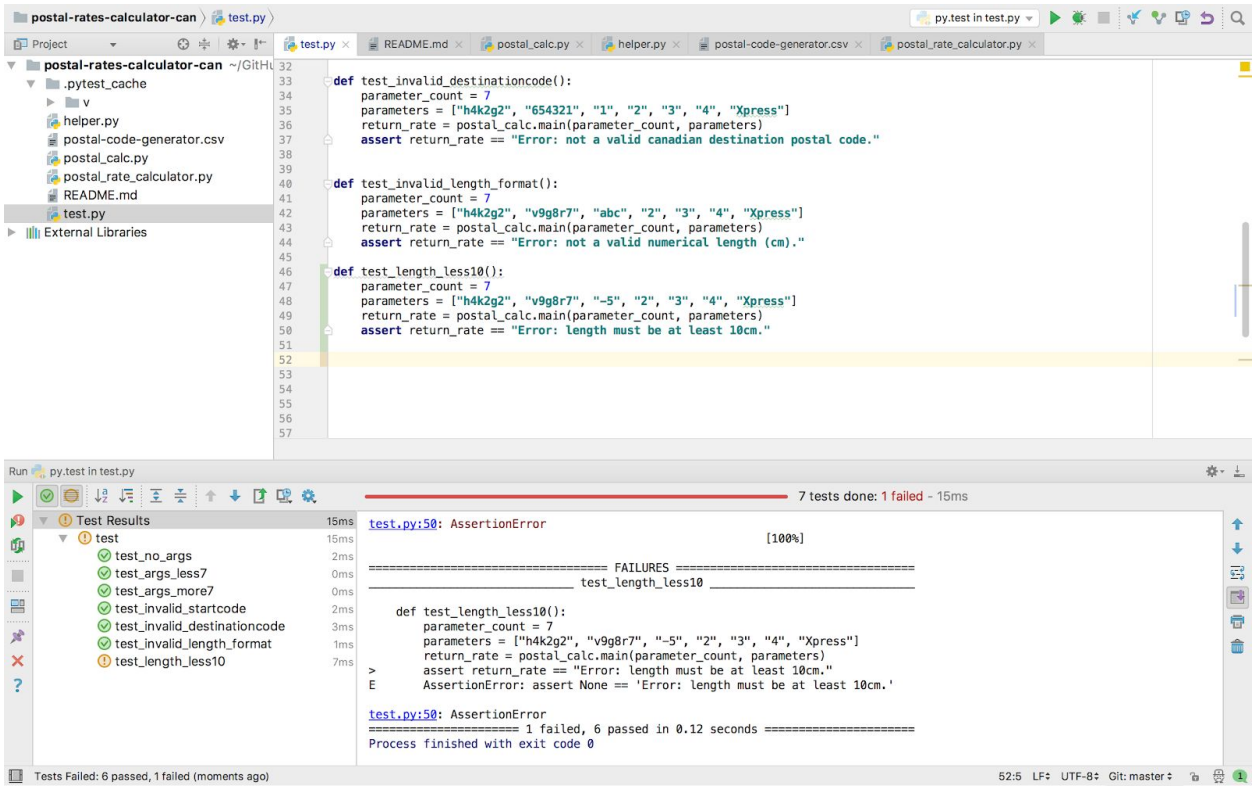
Test 6 Code with changes:



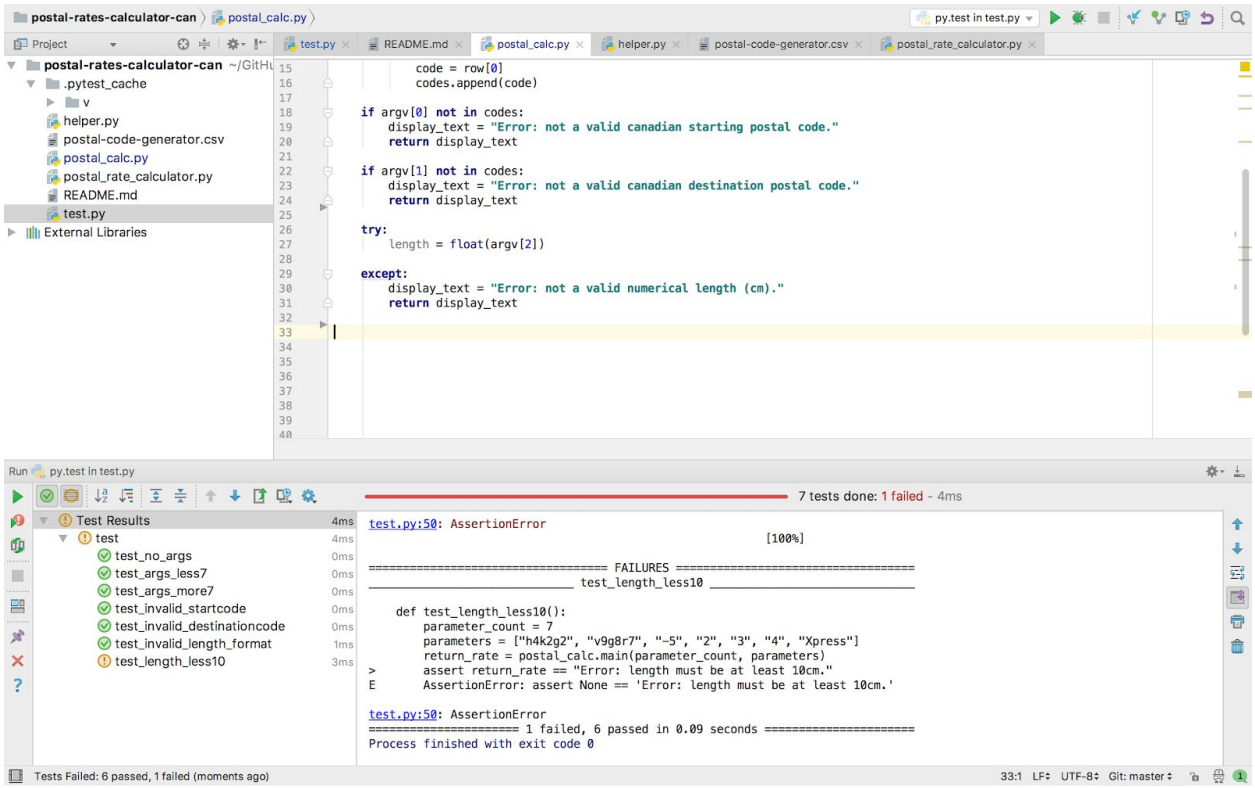
Test 7: Length too short

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.

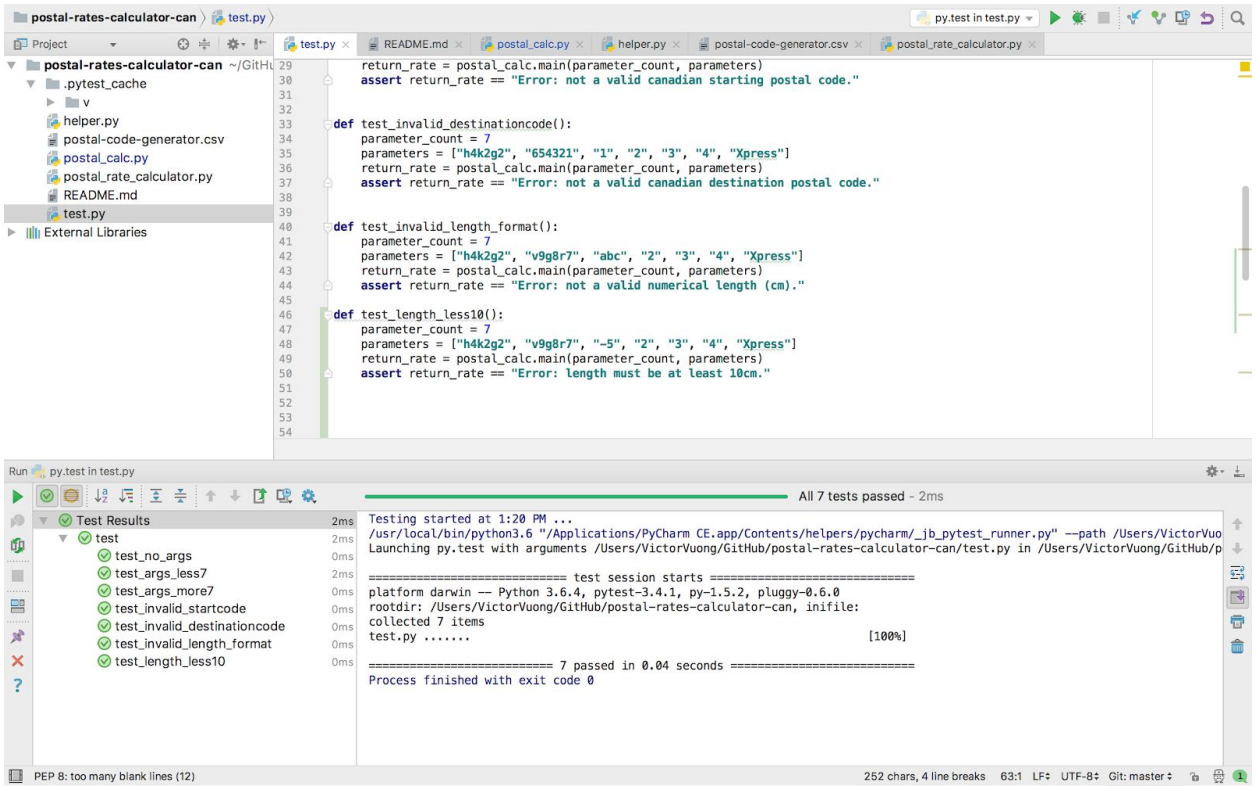
Test 7 Fail:



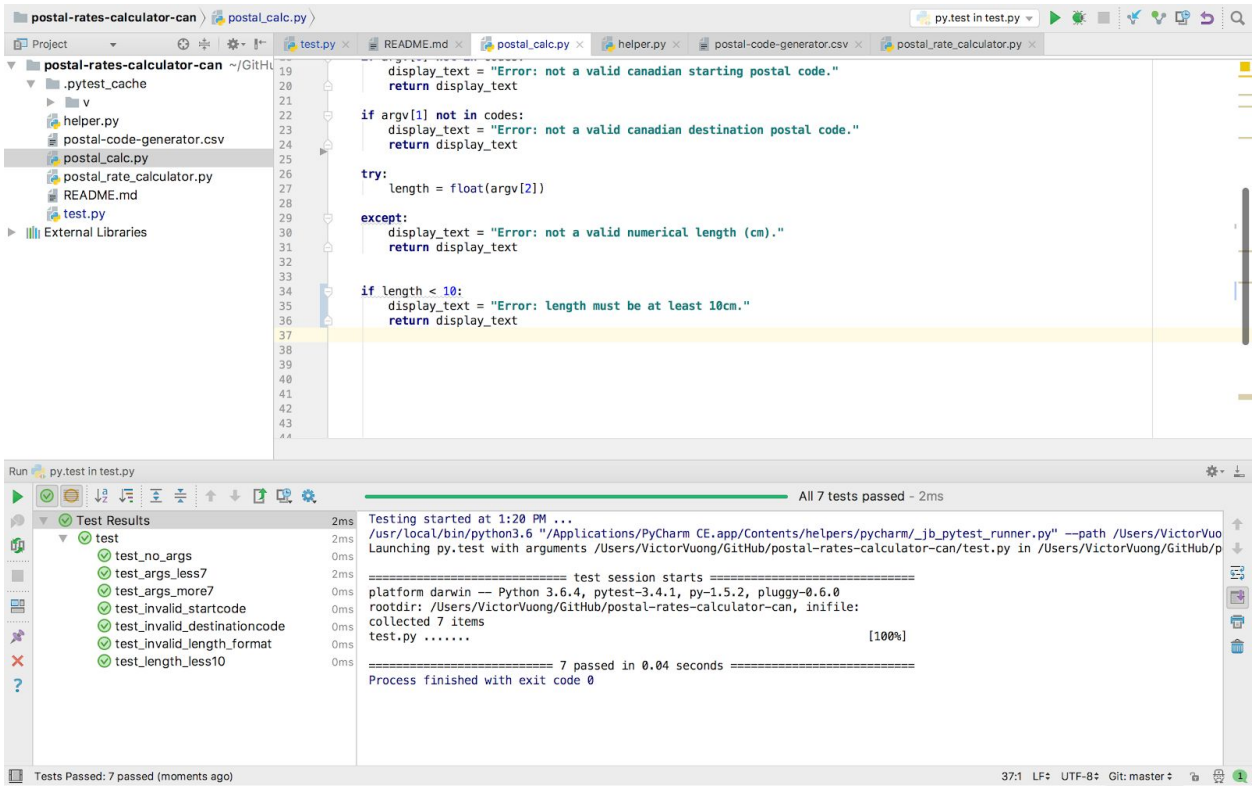
Test 7 Code:



Test 7 Succeed:



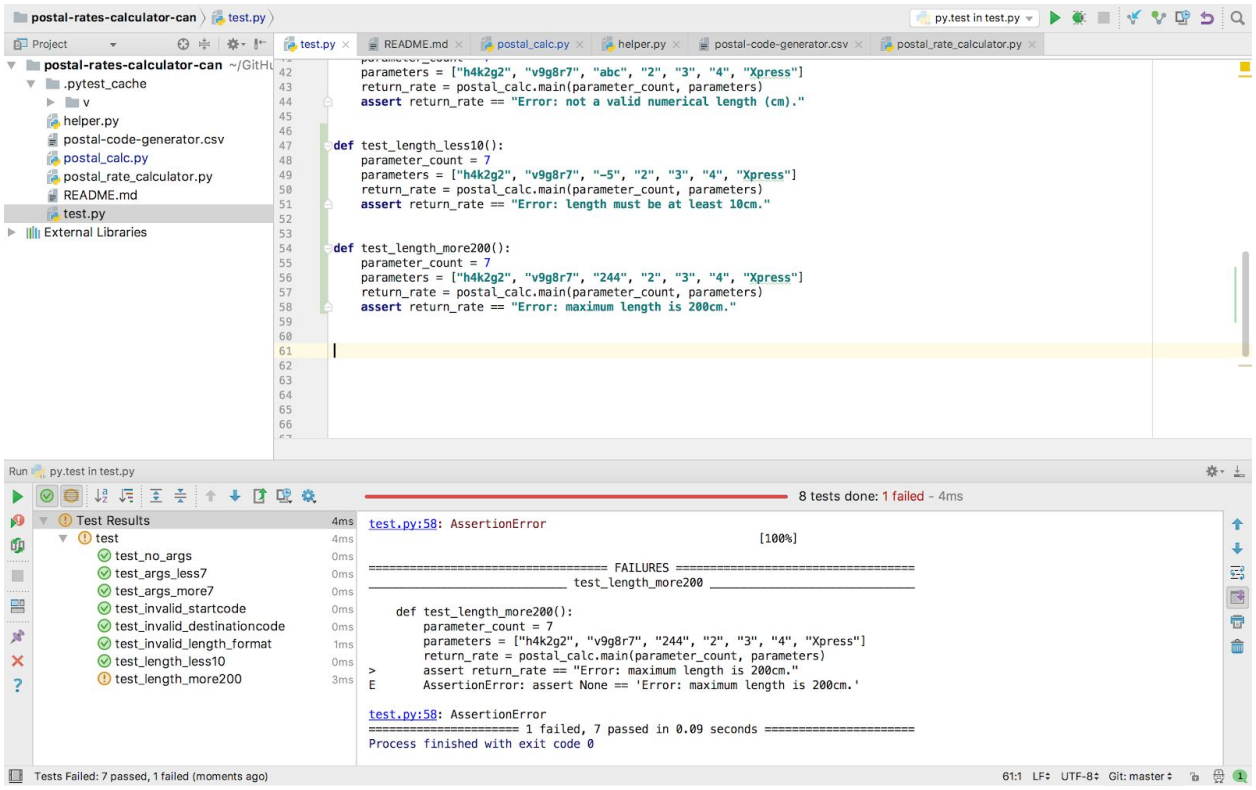
Test 7 Code with changes:



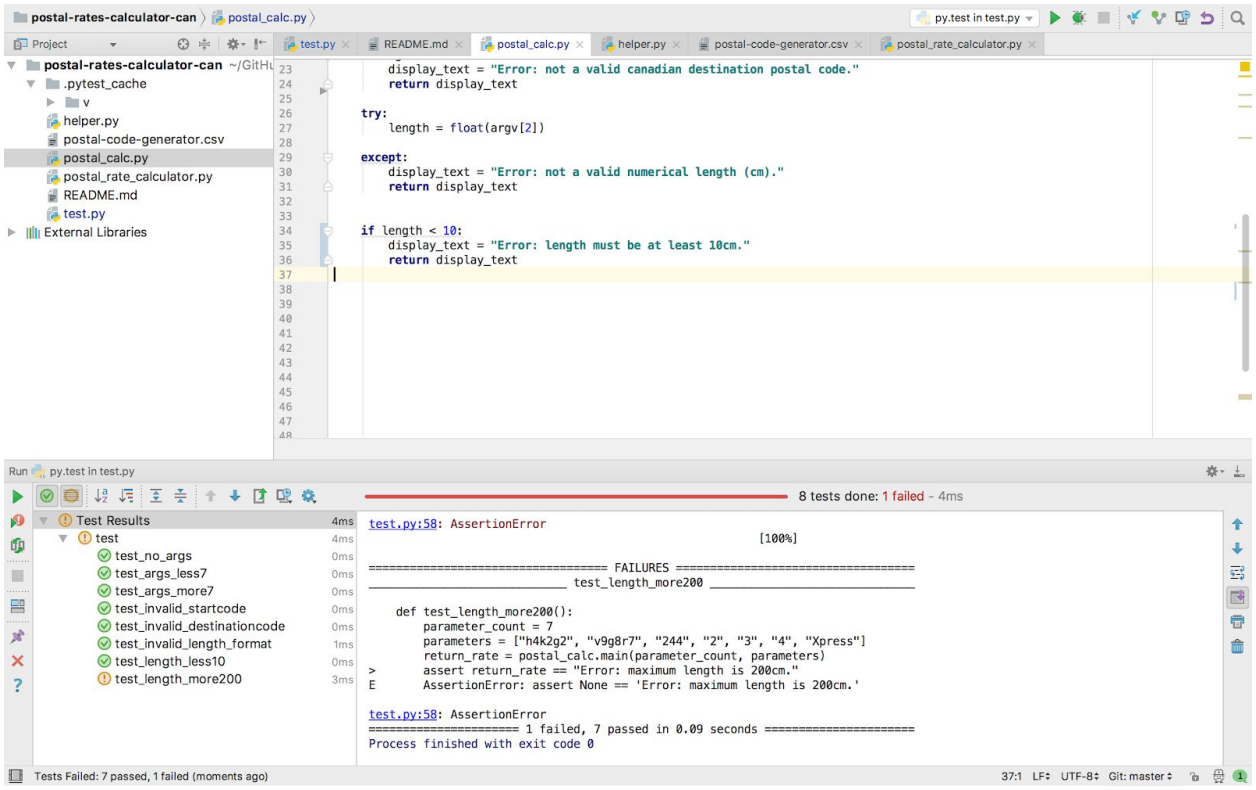
Test 8: Length too long

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.

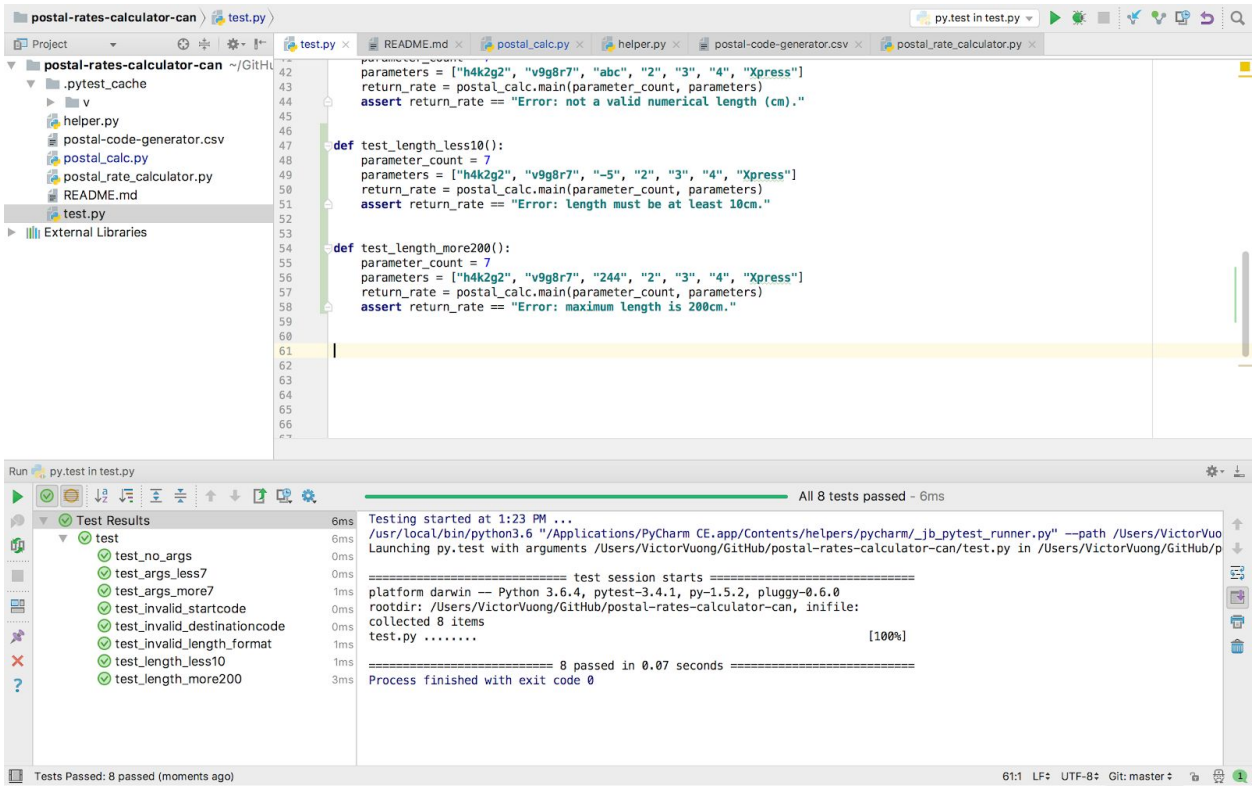
Test 8 Fail:



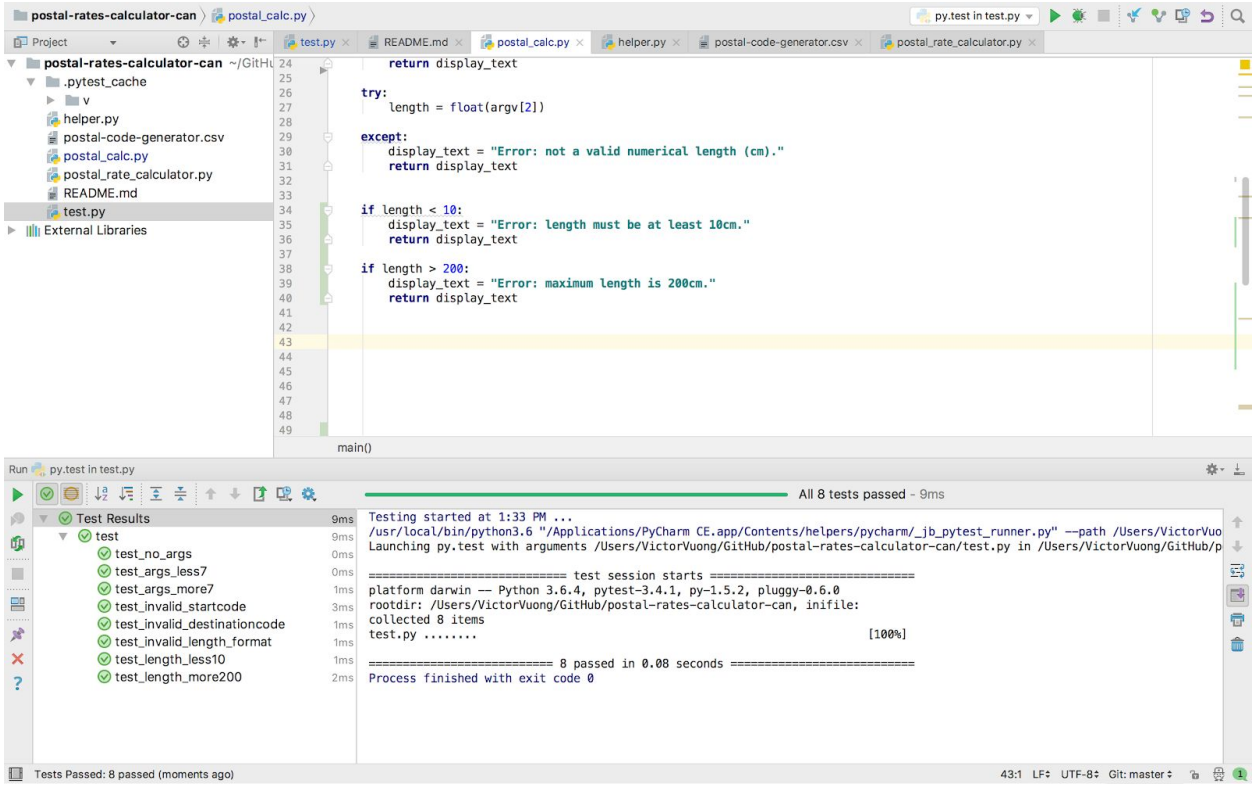
Test 8 Code:



Test 8 Succeed:



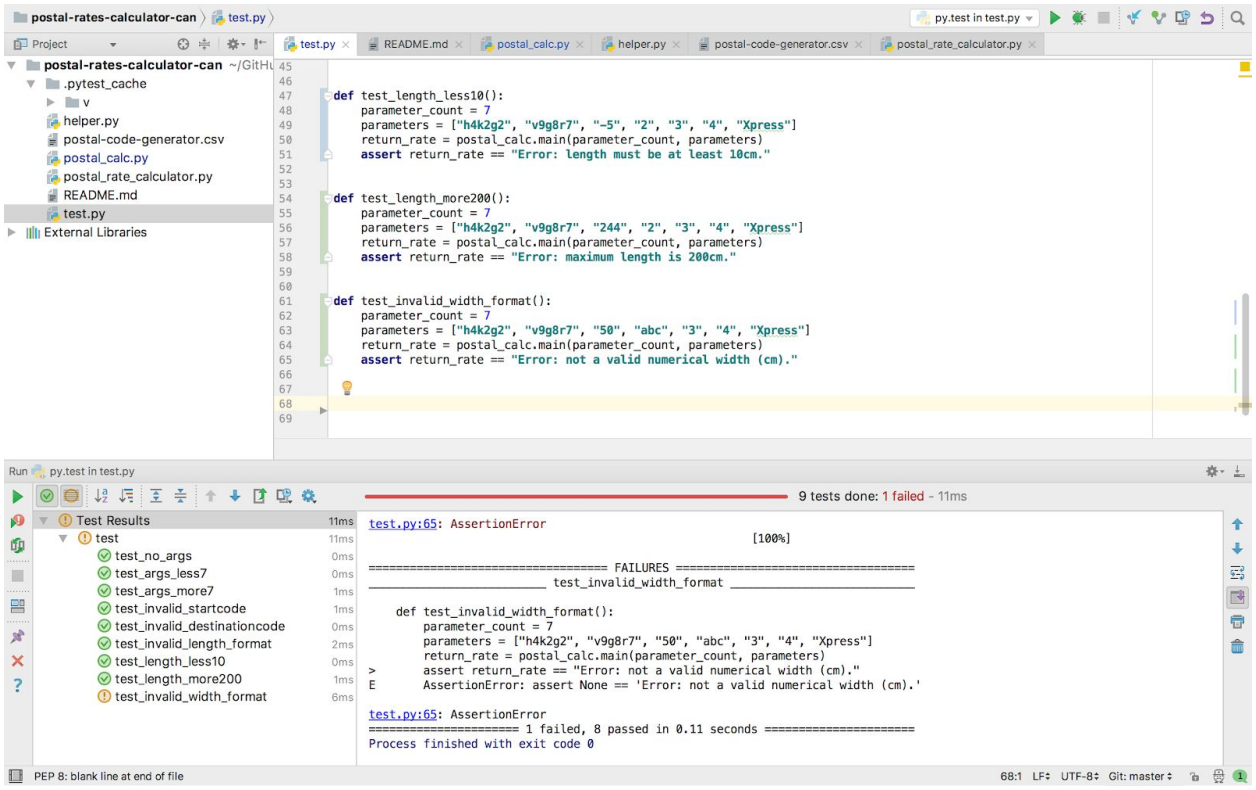
Test 8 Code with changes:



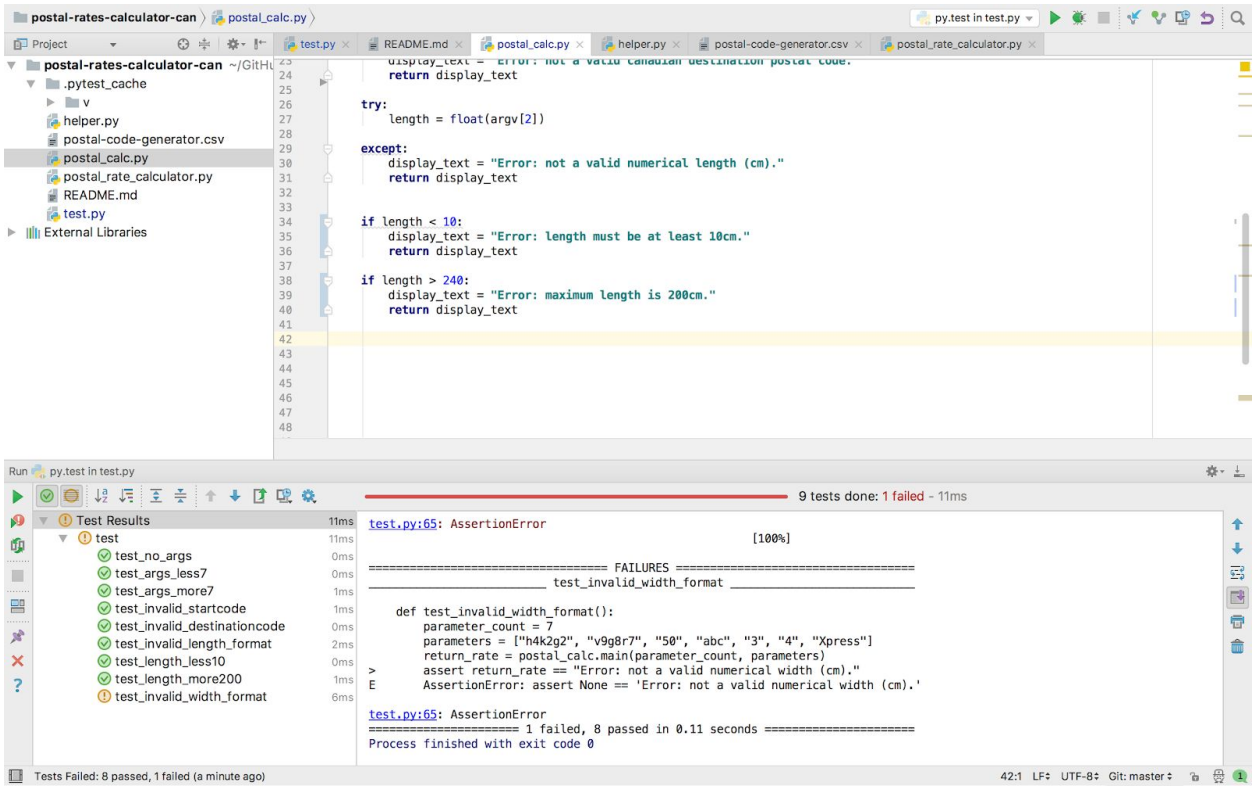
Test 9: Invalid width format

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.

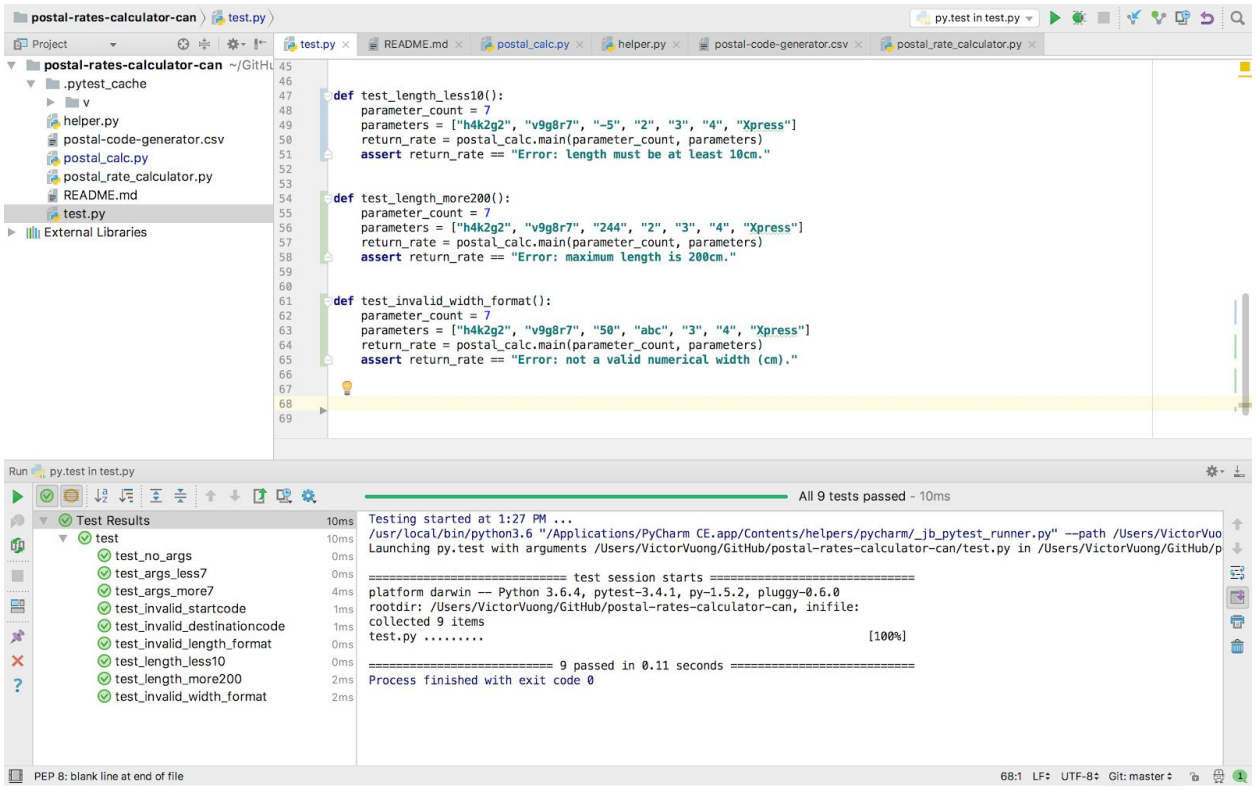
Test 9 Fail:



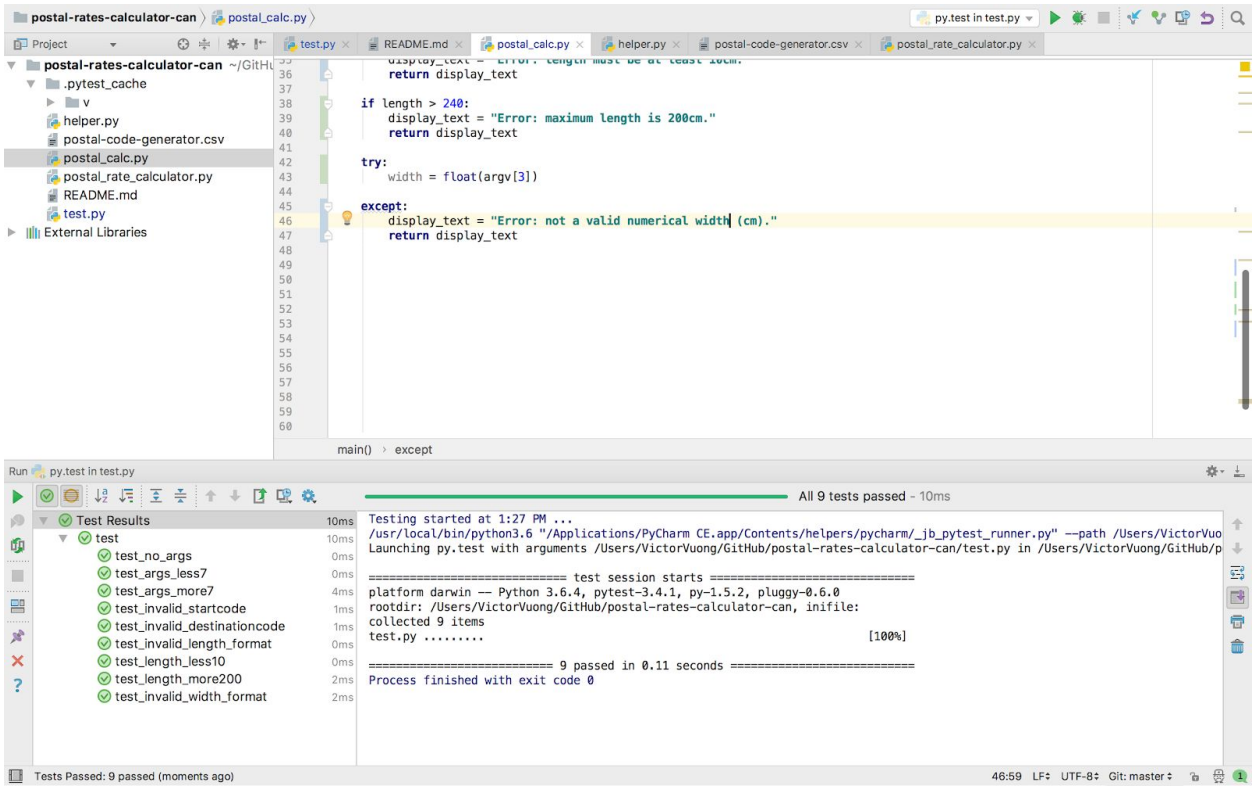
Test 9 Code:



Test 9 Succeed:



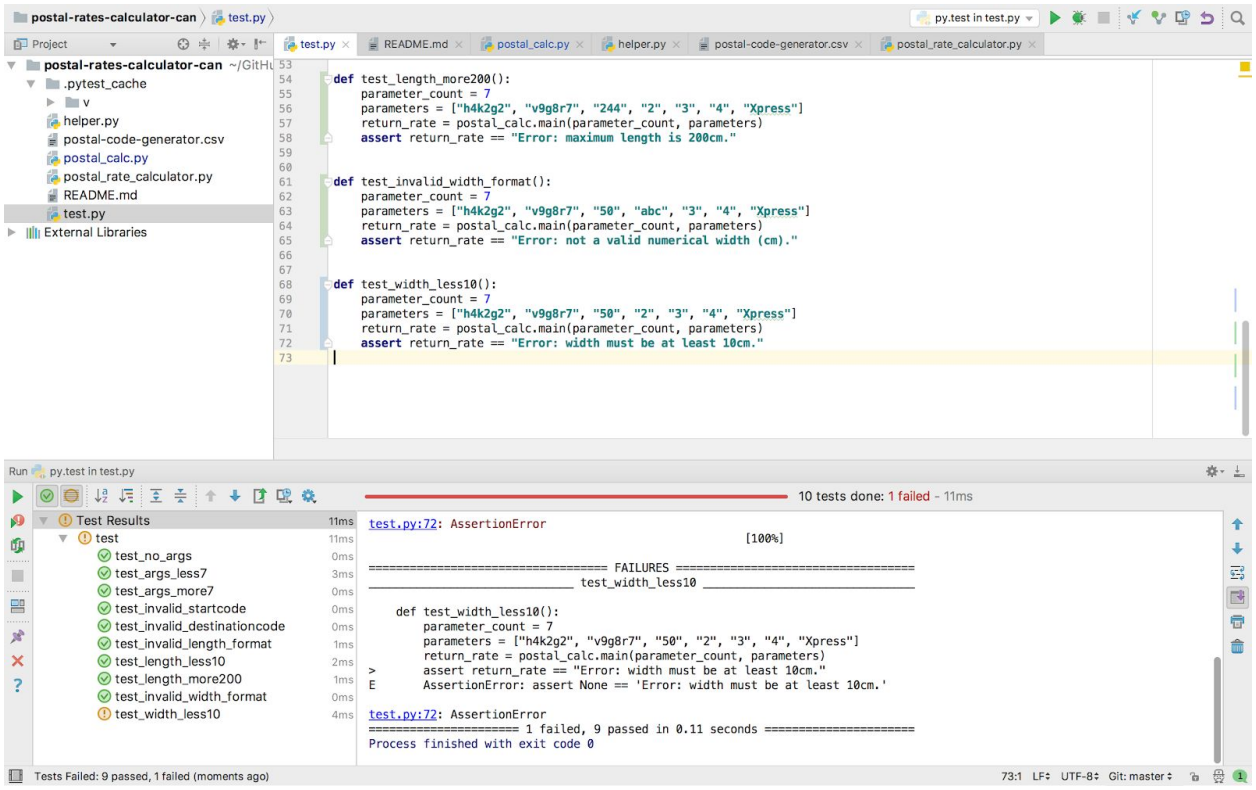
Test 9 Code with changes:



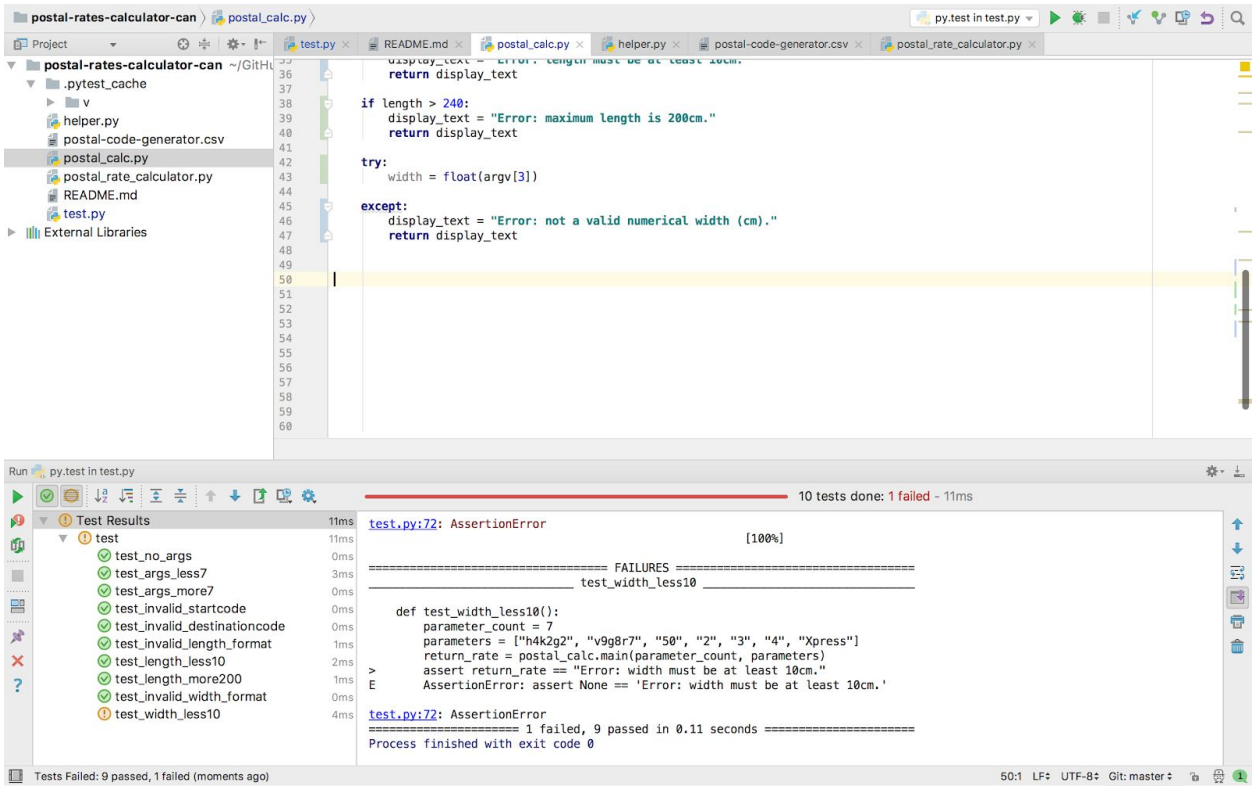
Test 10: Width too short

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.

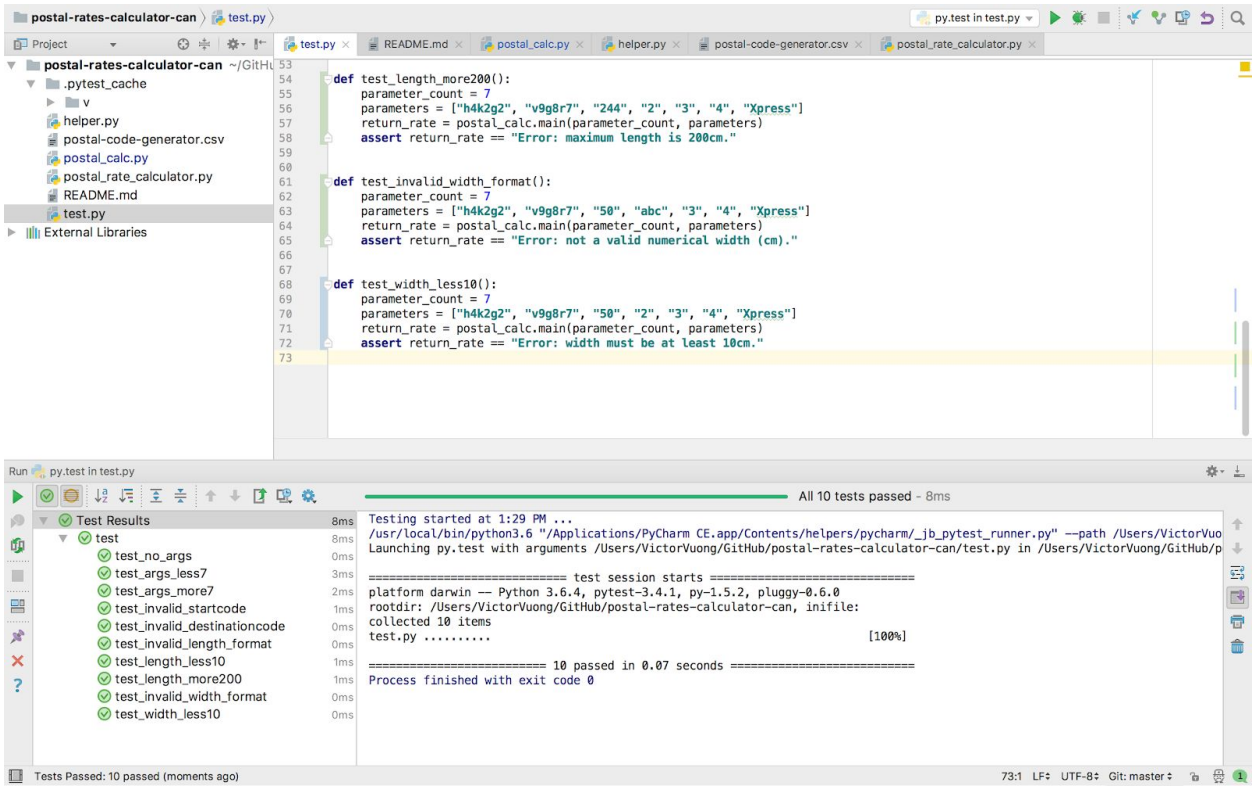
Test 10 Fail:



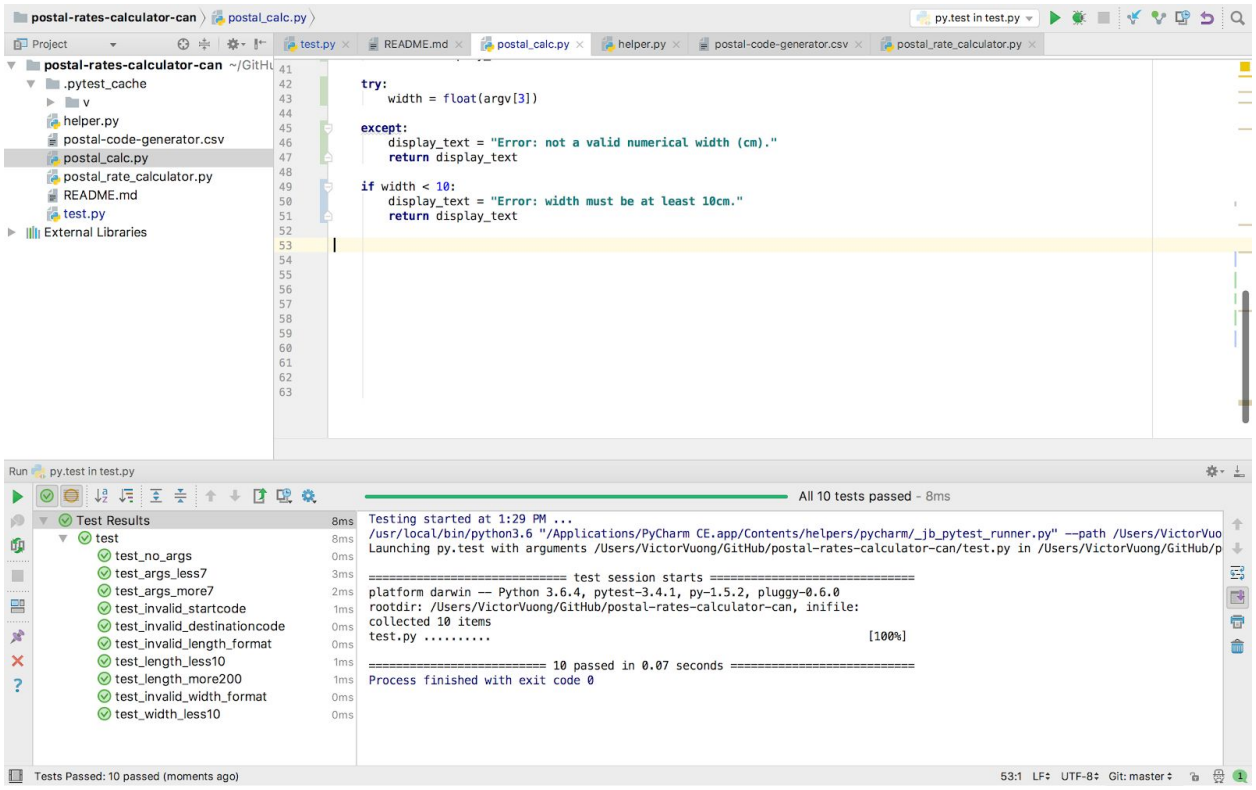
Test 10 Code:



Test 10 Succeed:



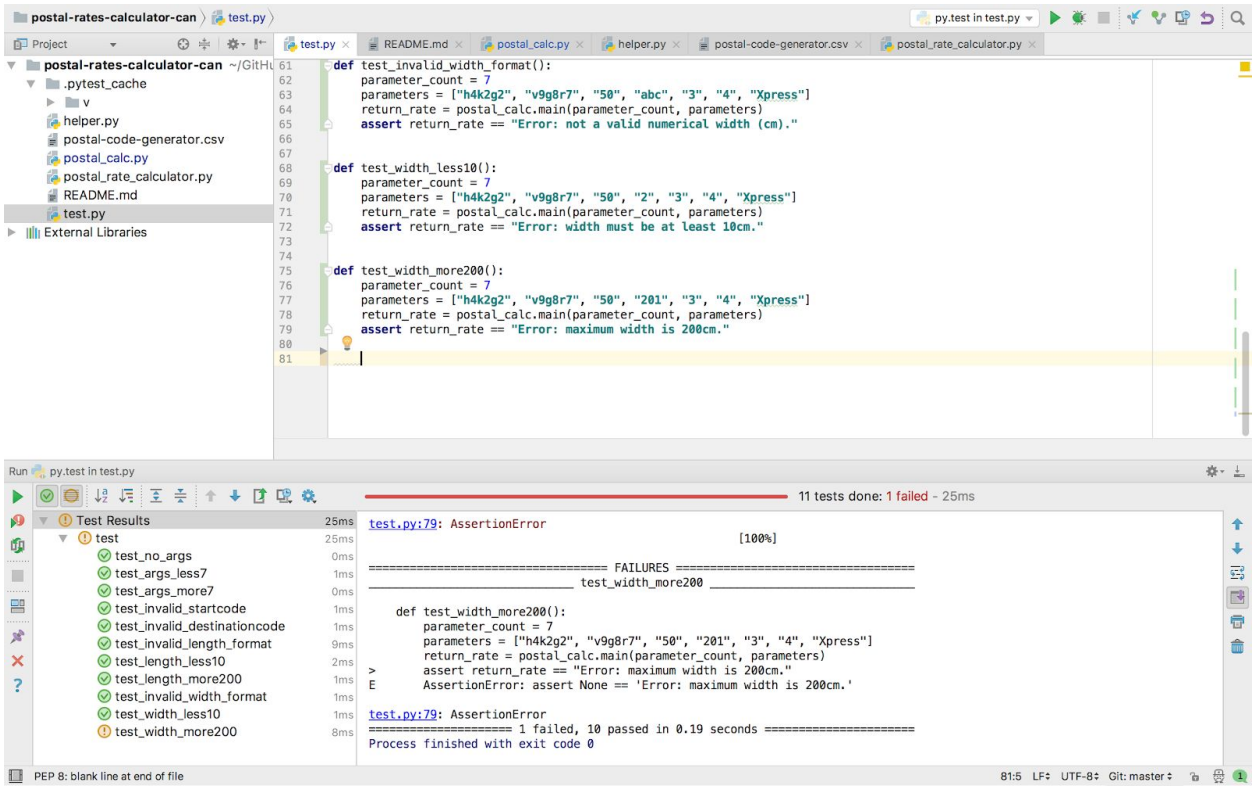
Test 10 Code with changes:



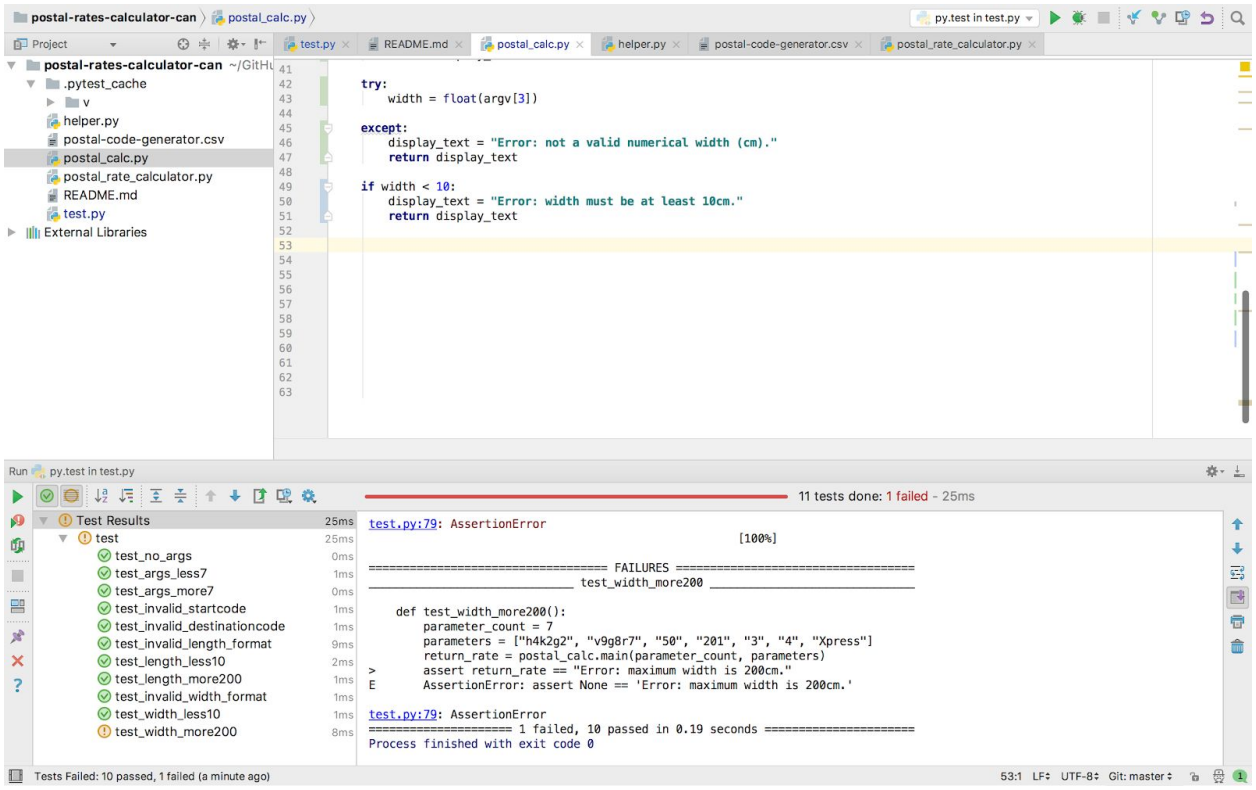
Test 11: Width too long

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.

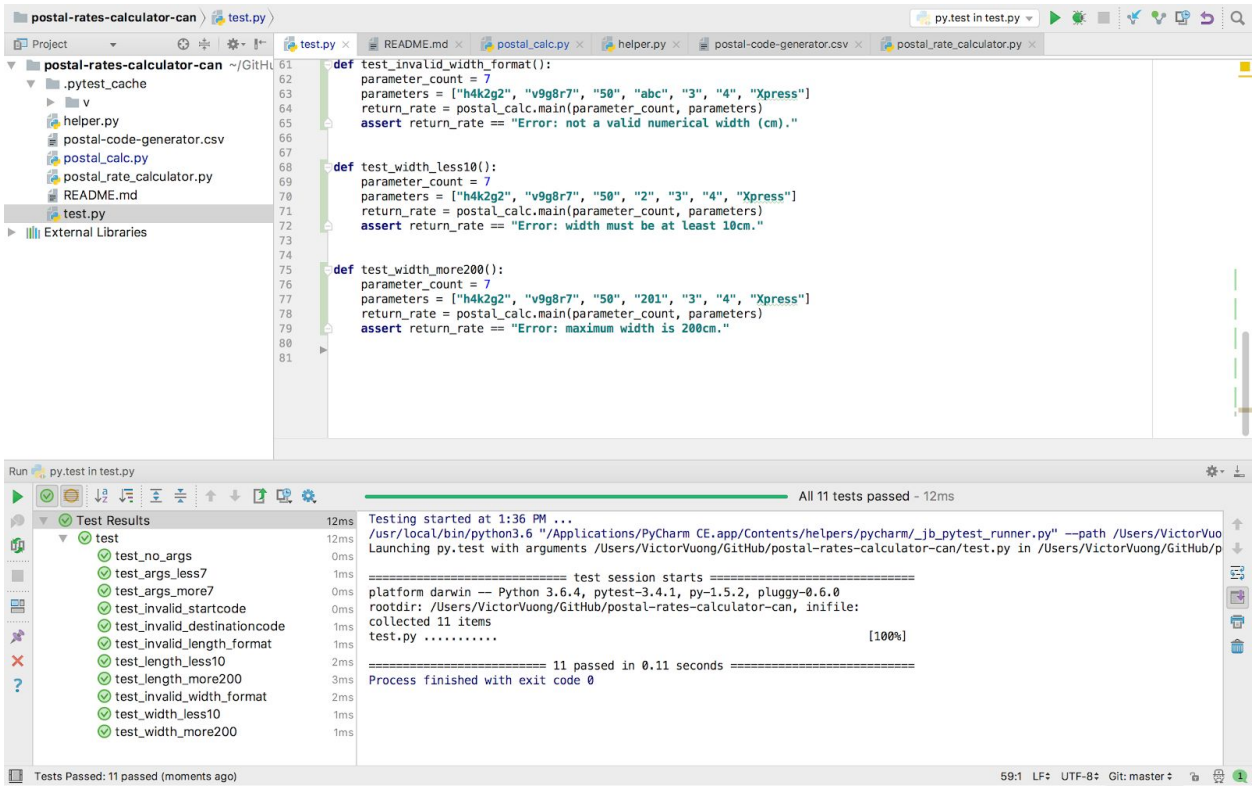
Test 11 Fail:



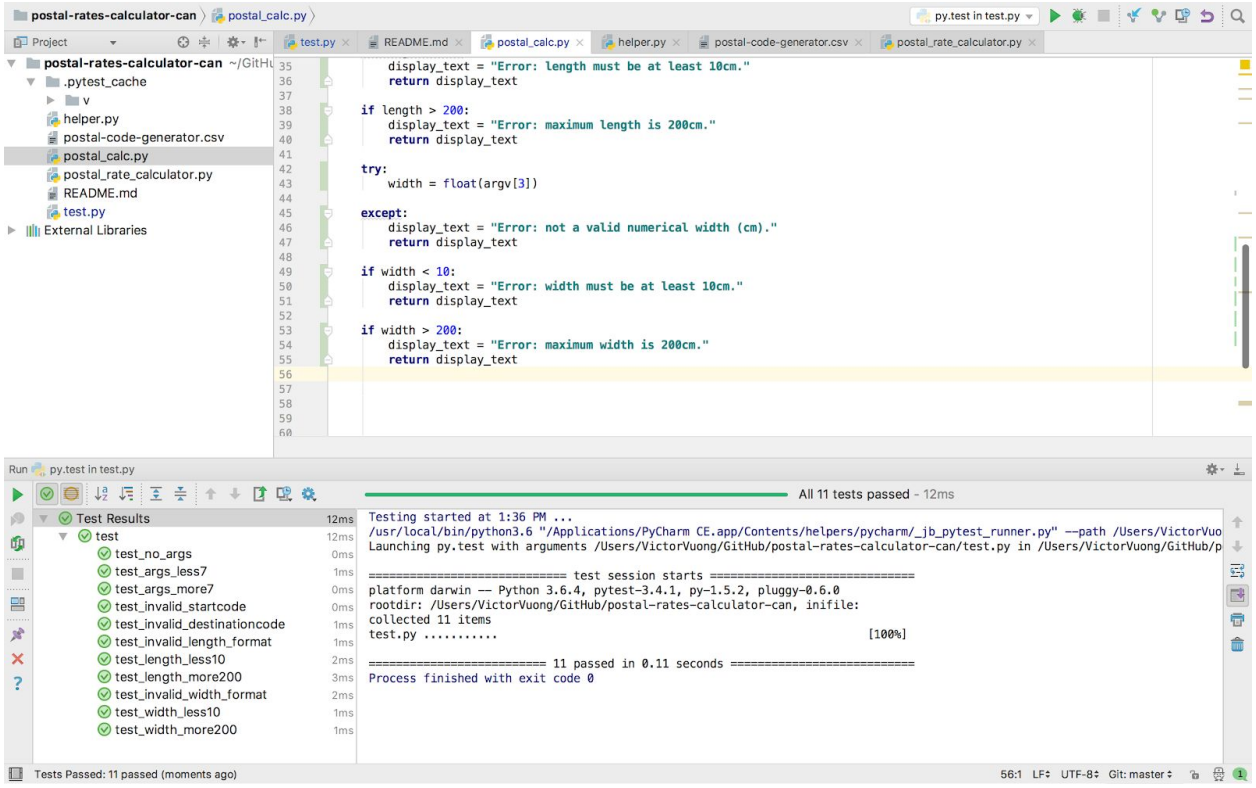
Test 11 Code:



Test 11 Succeed:



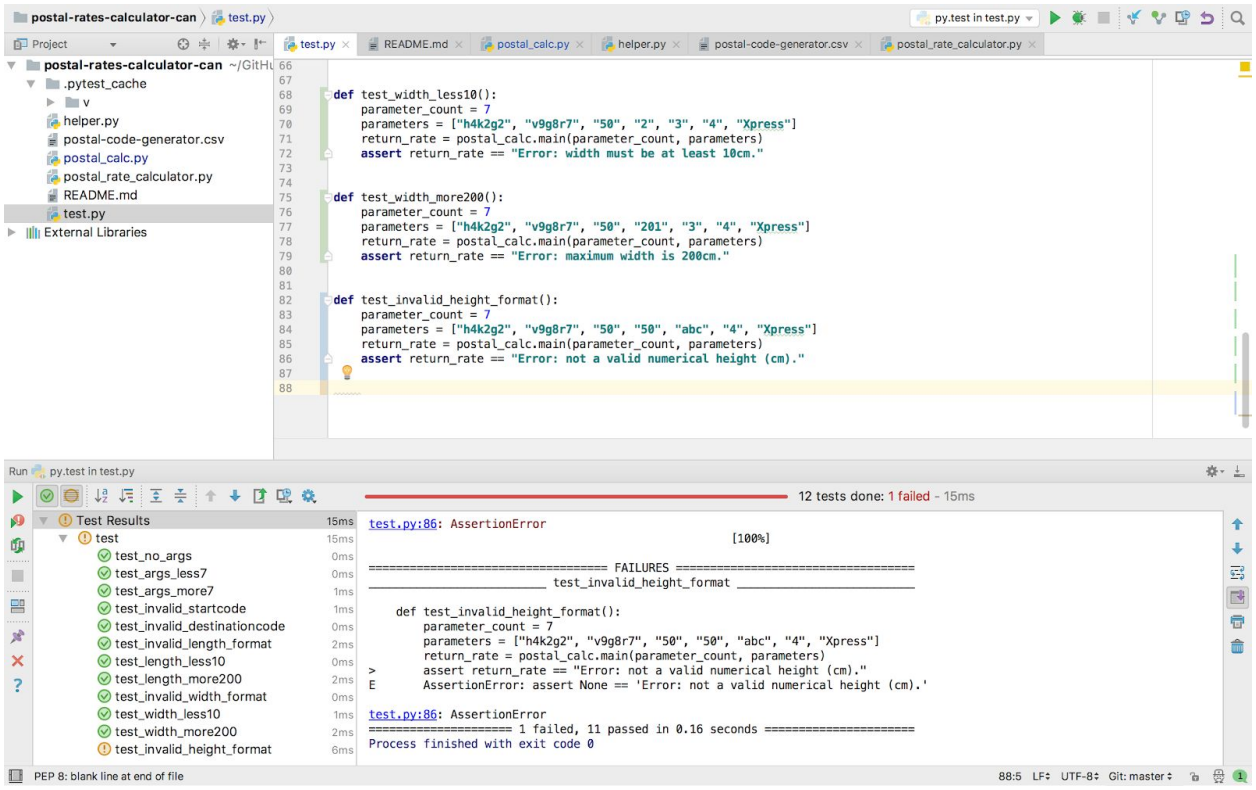
Test 11 Code with changes:



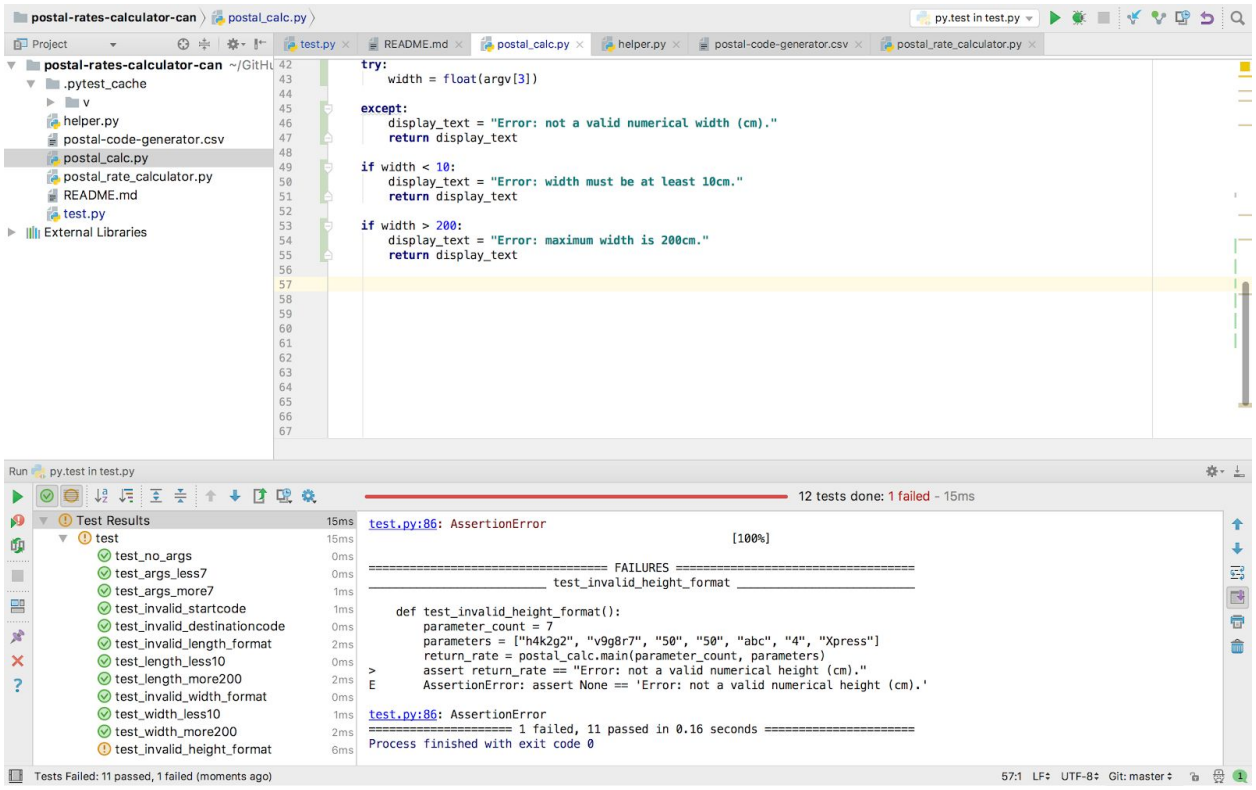
Test 12: Invalid height format

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.

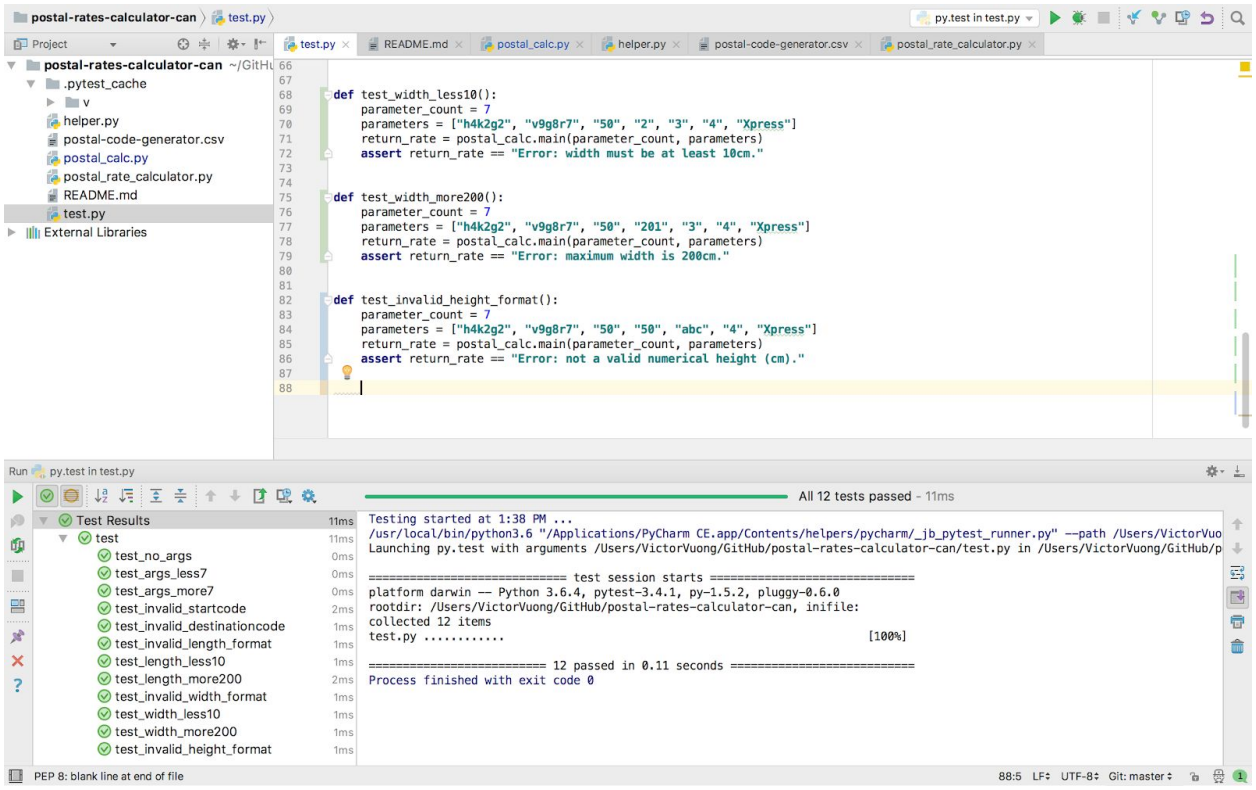
Test 12 Fail:



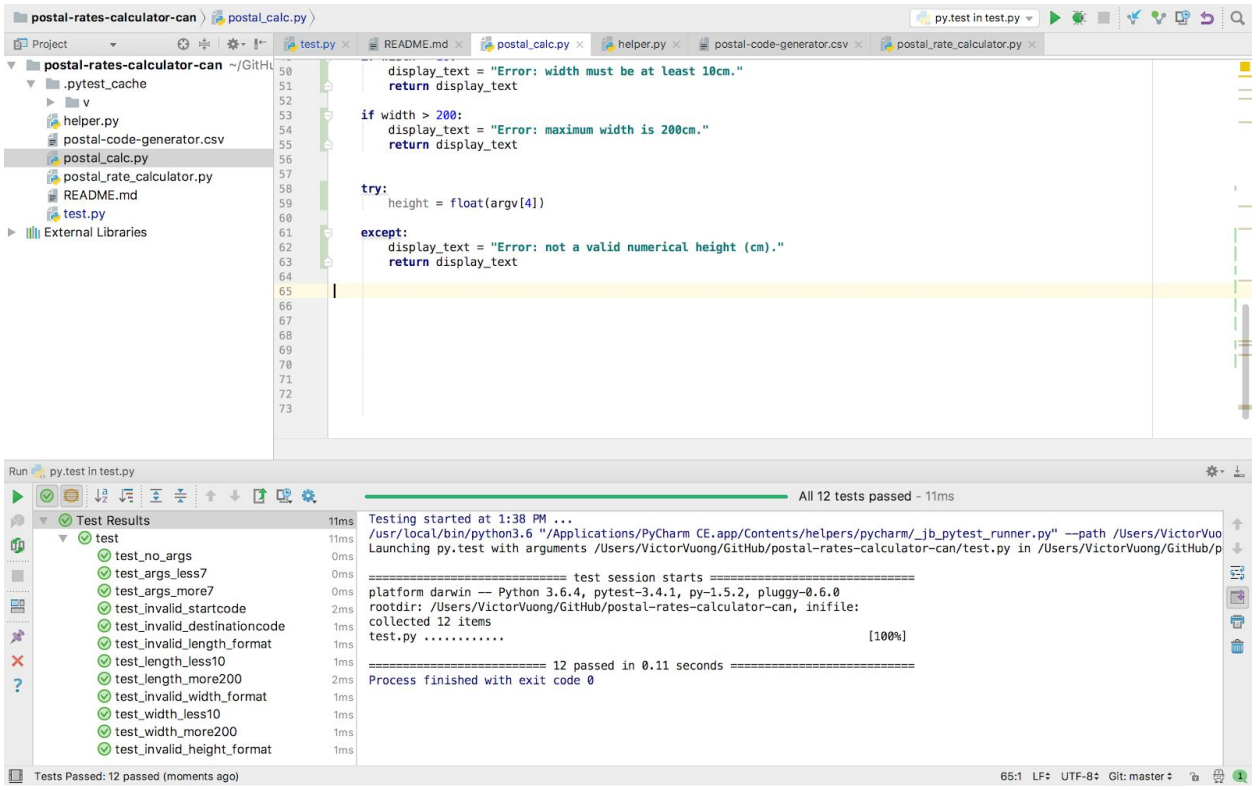
Test 12 Code:



Test 12 Succeed:



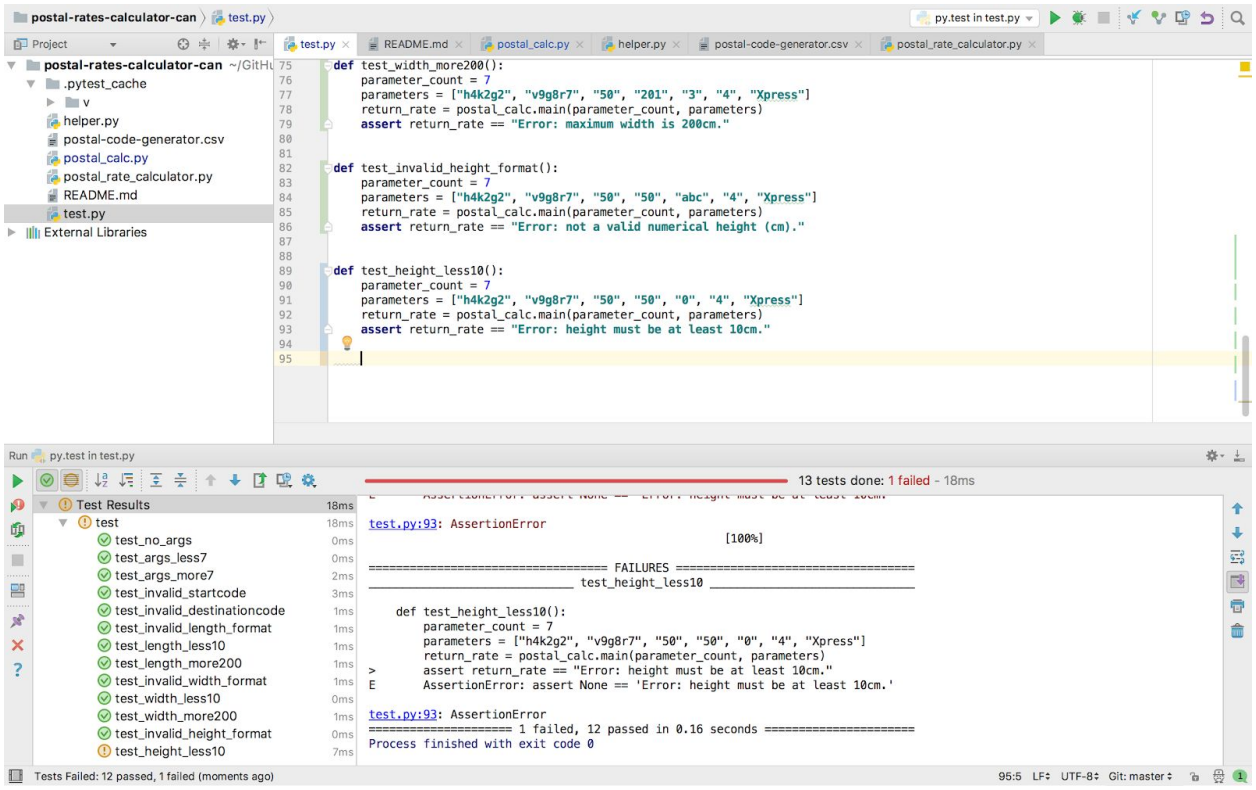
Test 12 Code with changes:



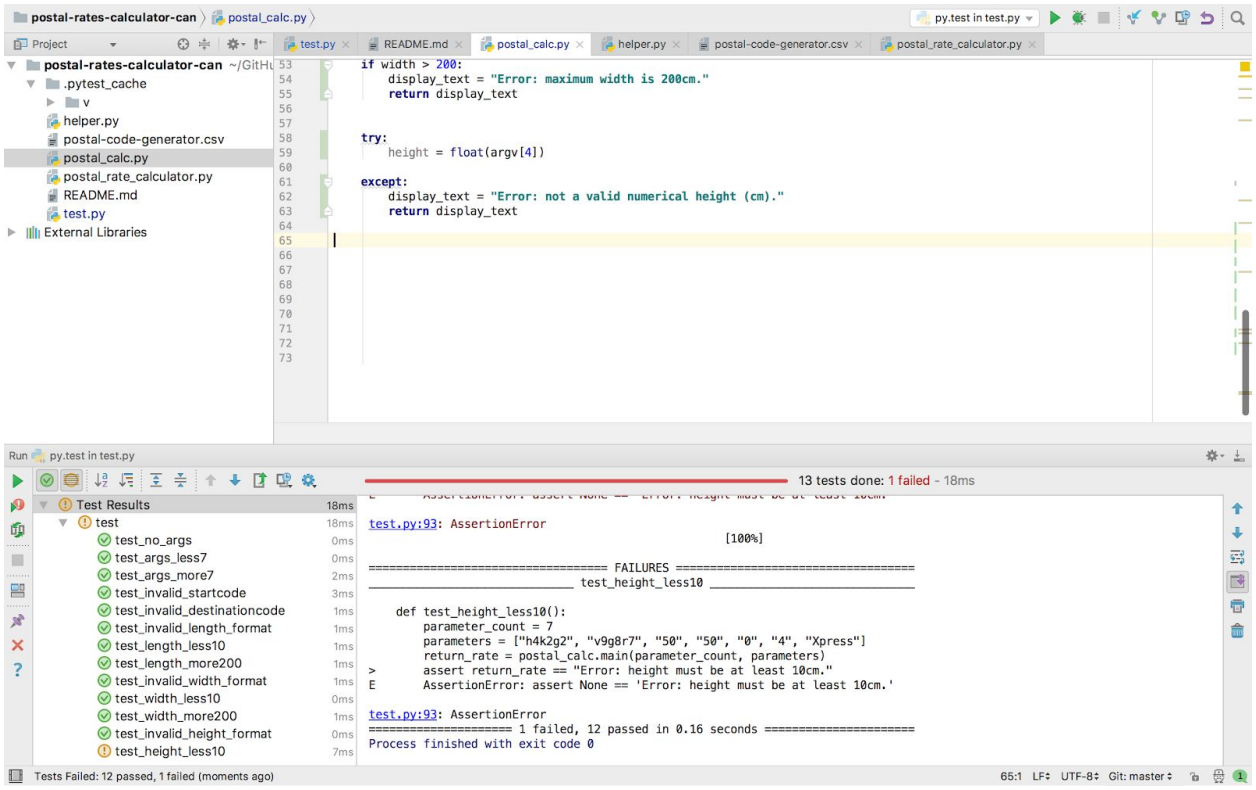
Test 13: Height too short

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.

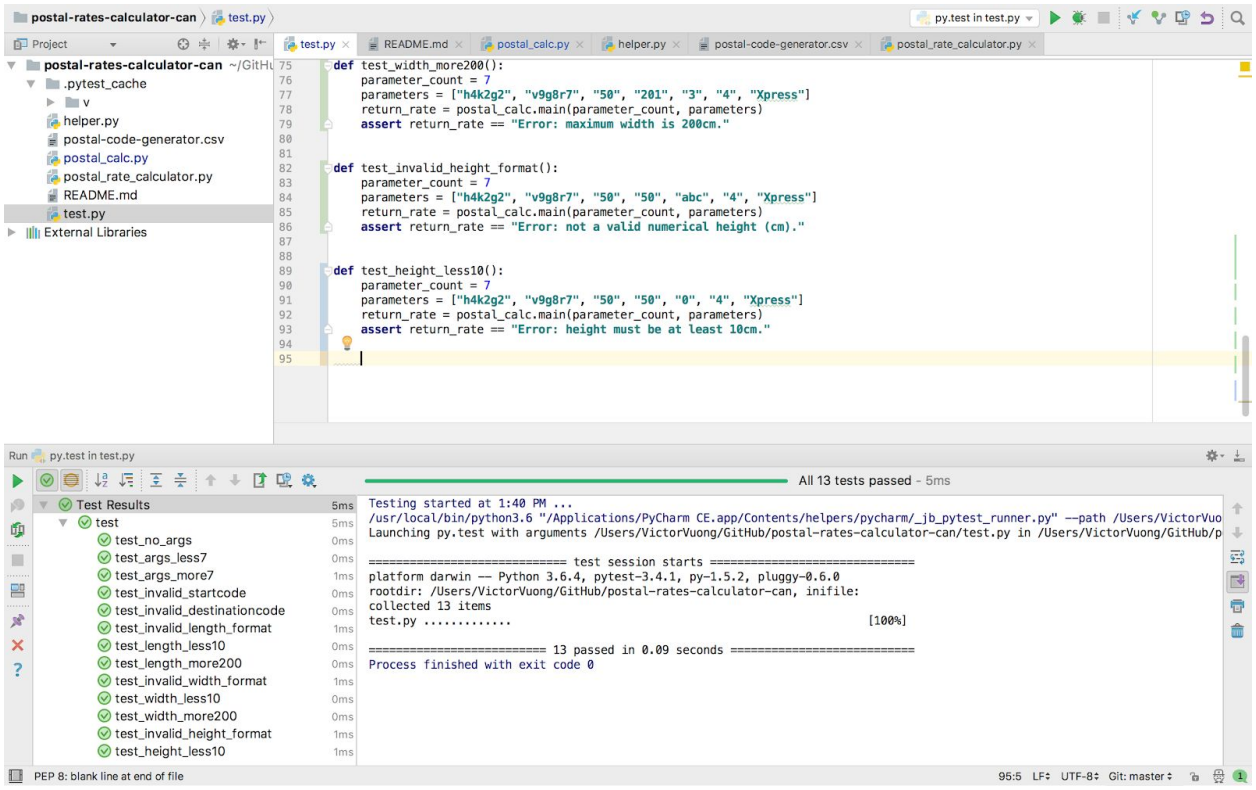
Test 13 Fail:



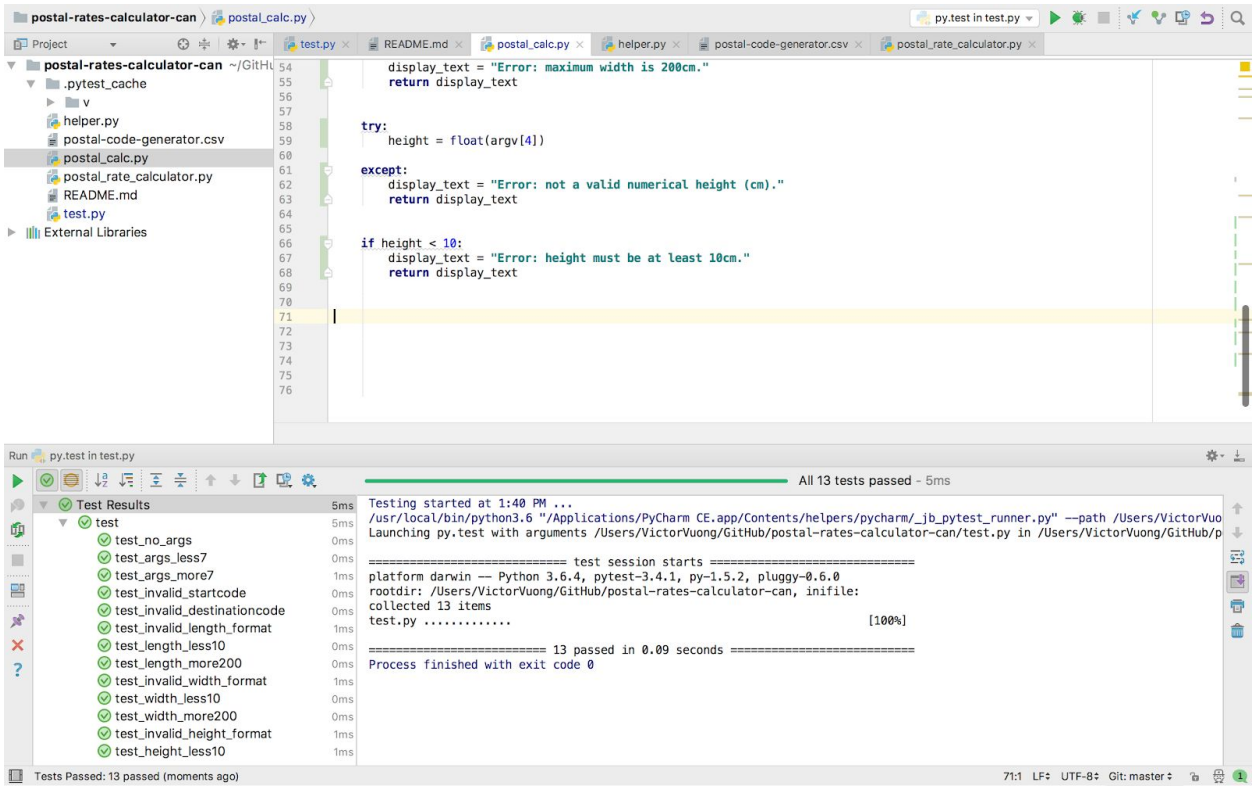
Test 13 Code:



Test 13 Succeed:



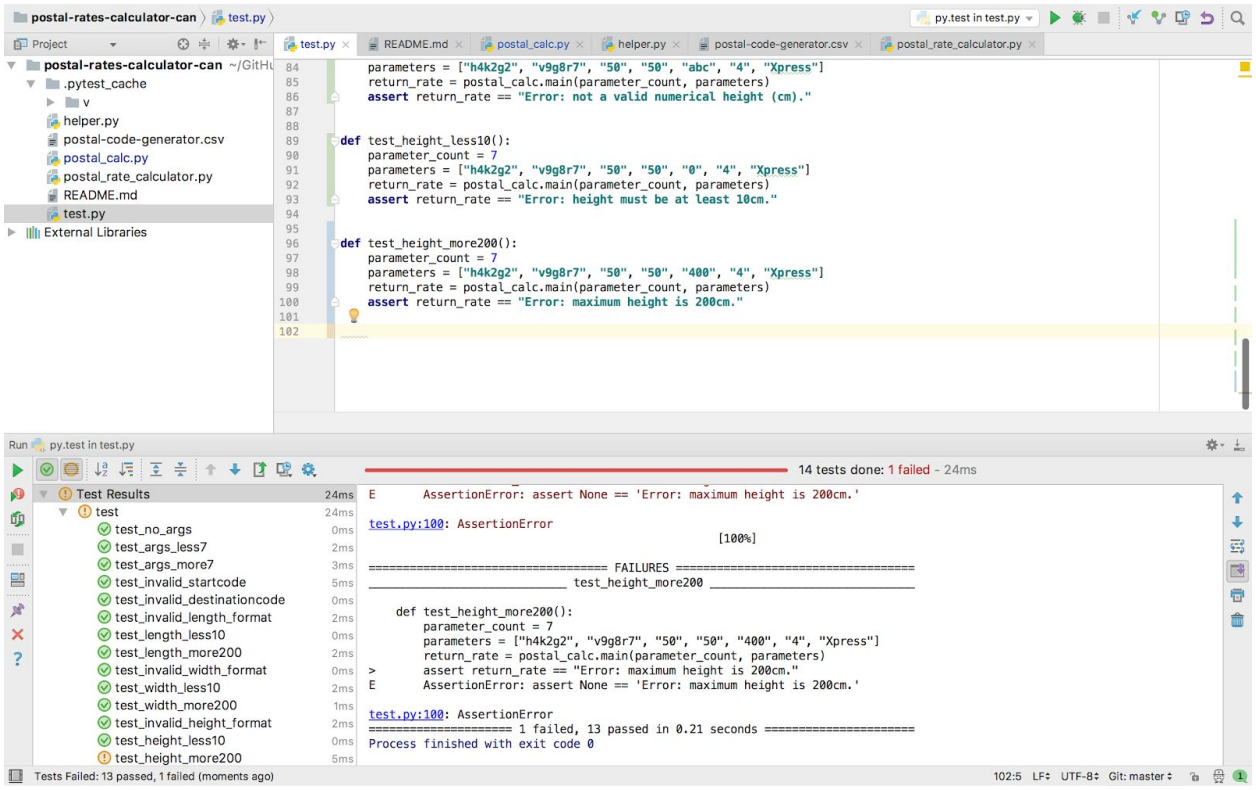
Test 13 Code with changes:



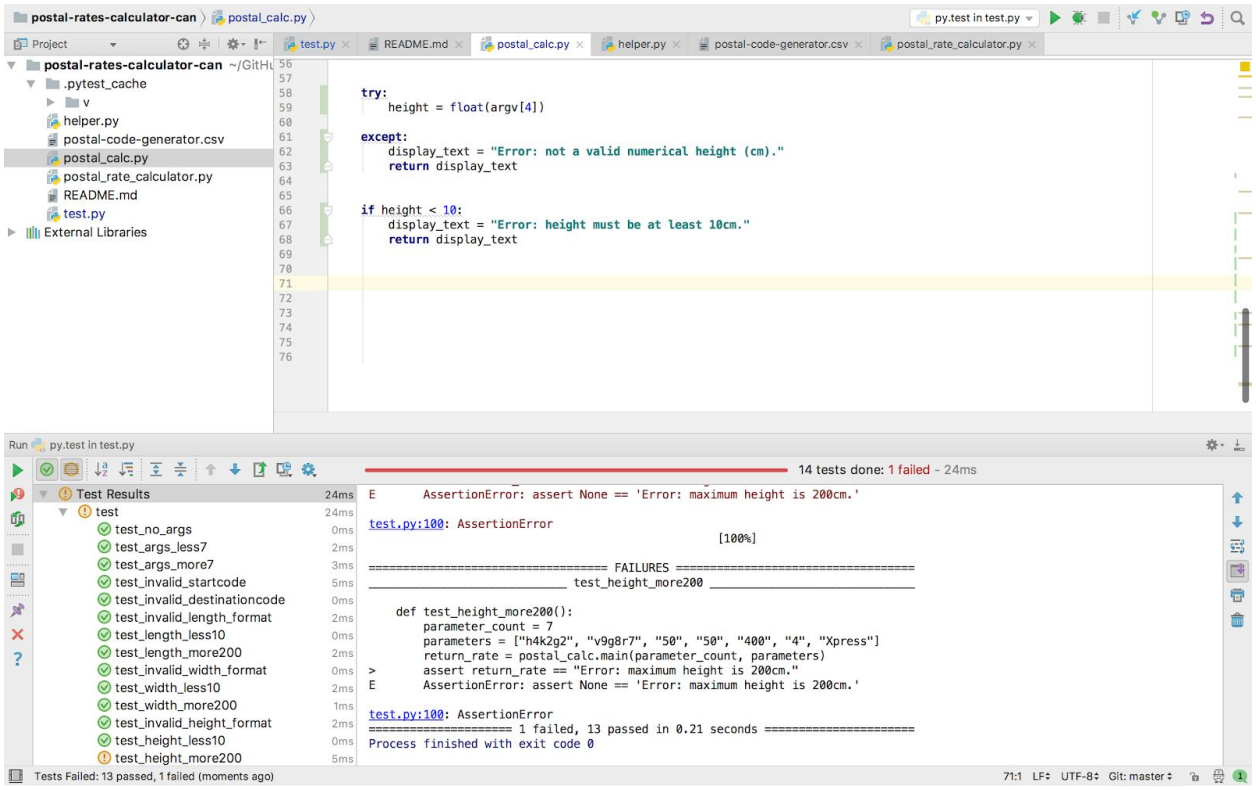
Test 14: Height too big

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.

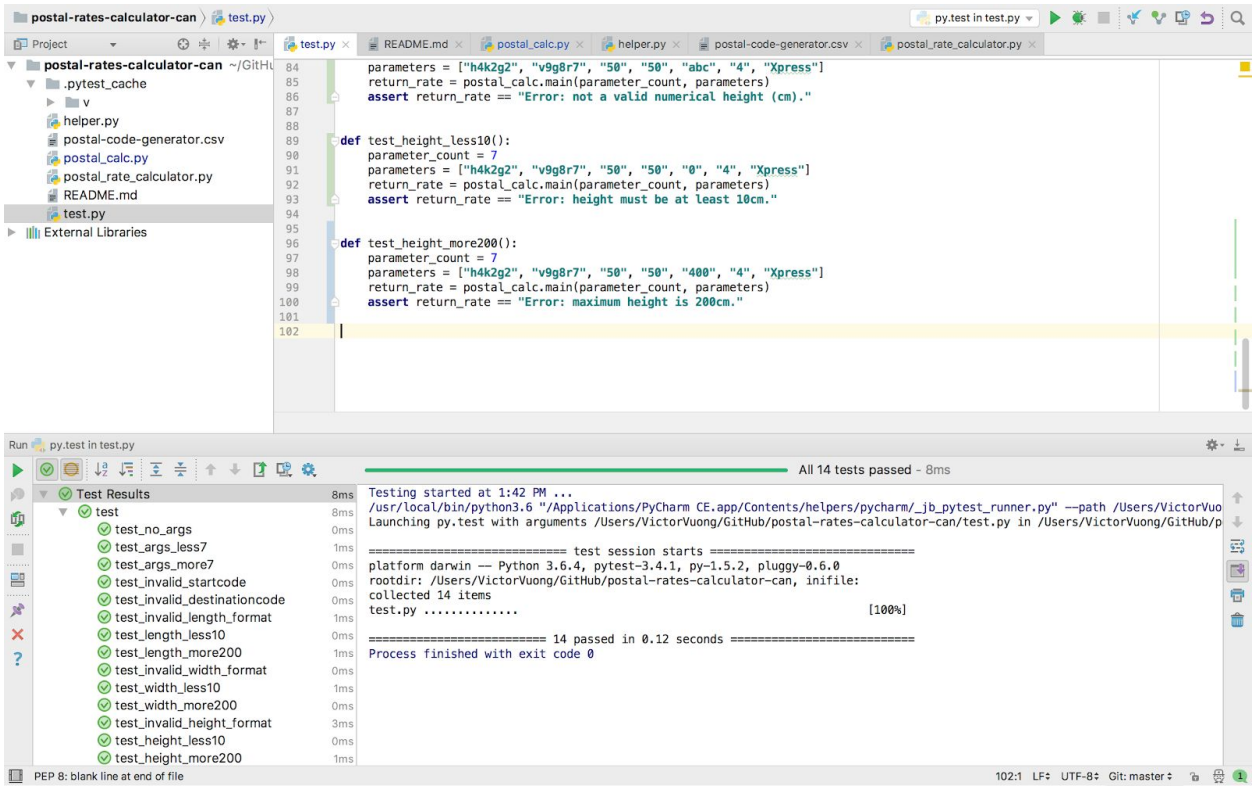
Test 14 Fail:



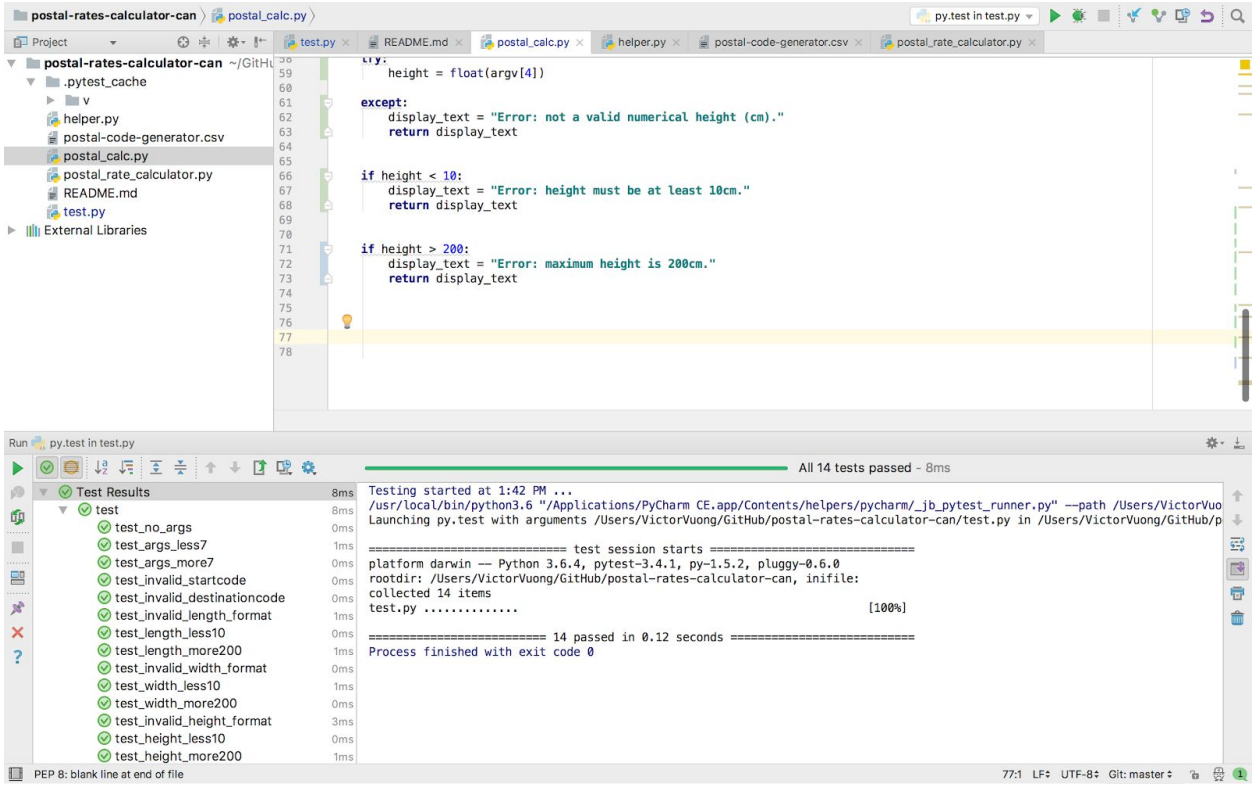
Test 14 Code:



Test 14 Succeed:



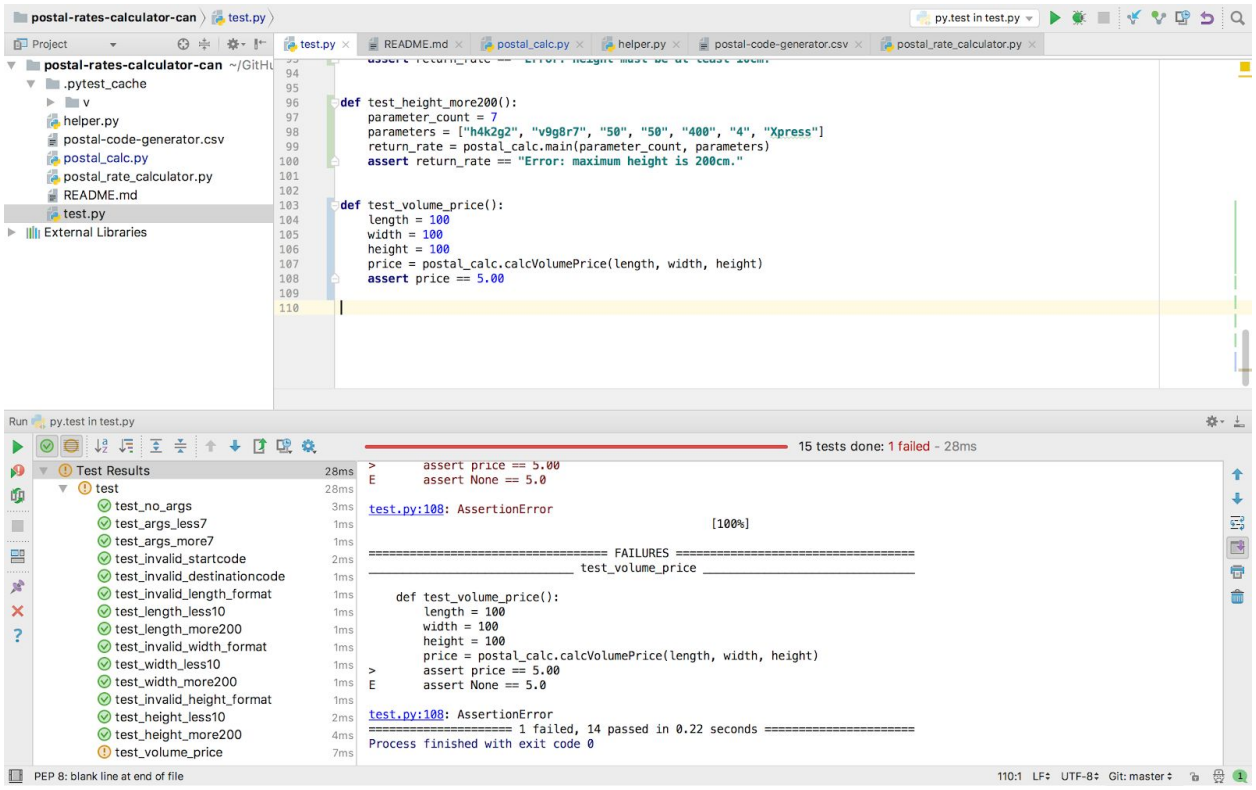
Test 14 Code with changes:



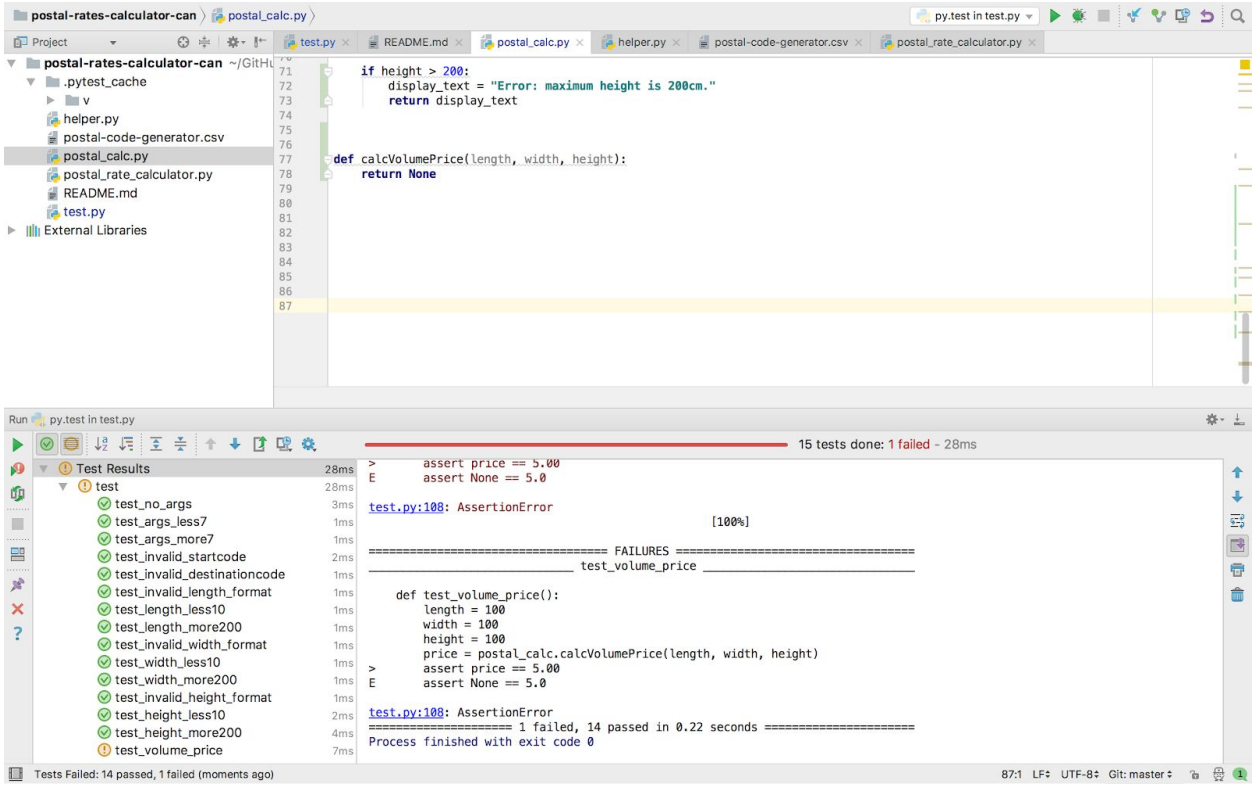
Test 15: Correct volume price

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.

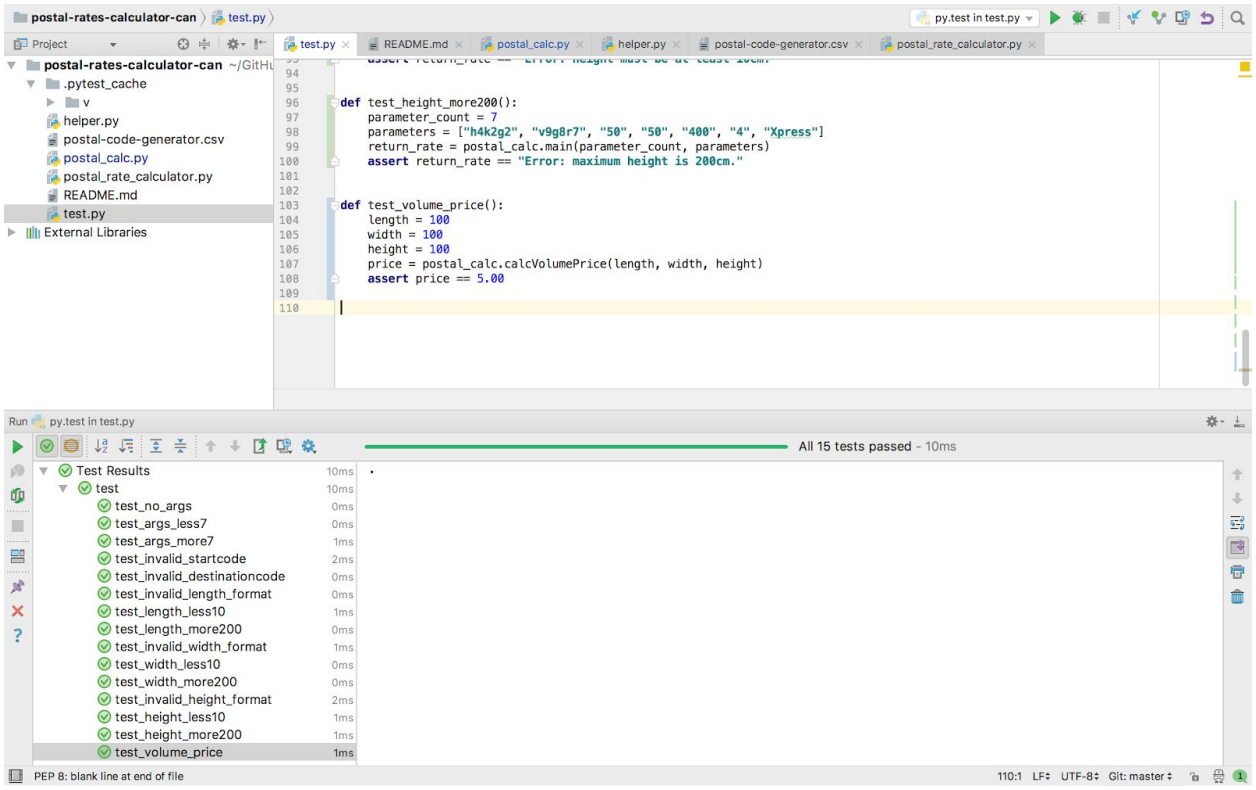
Test 15 Fail:



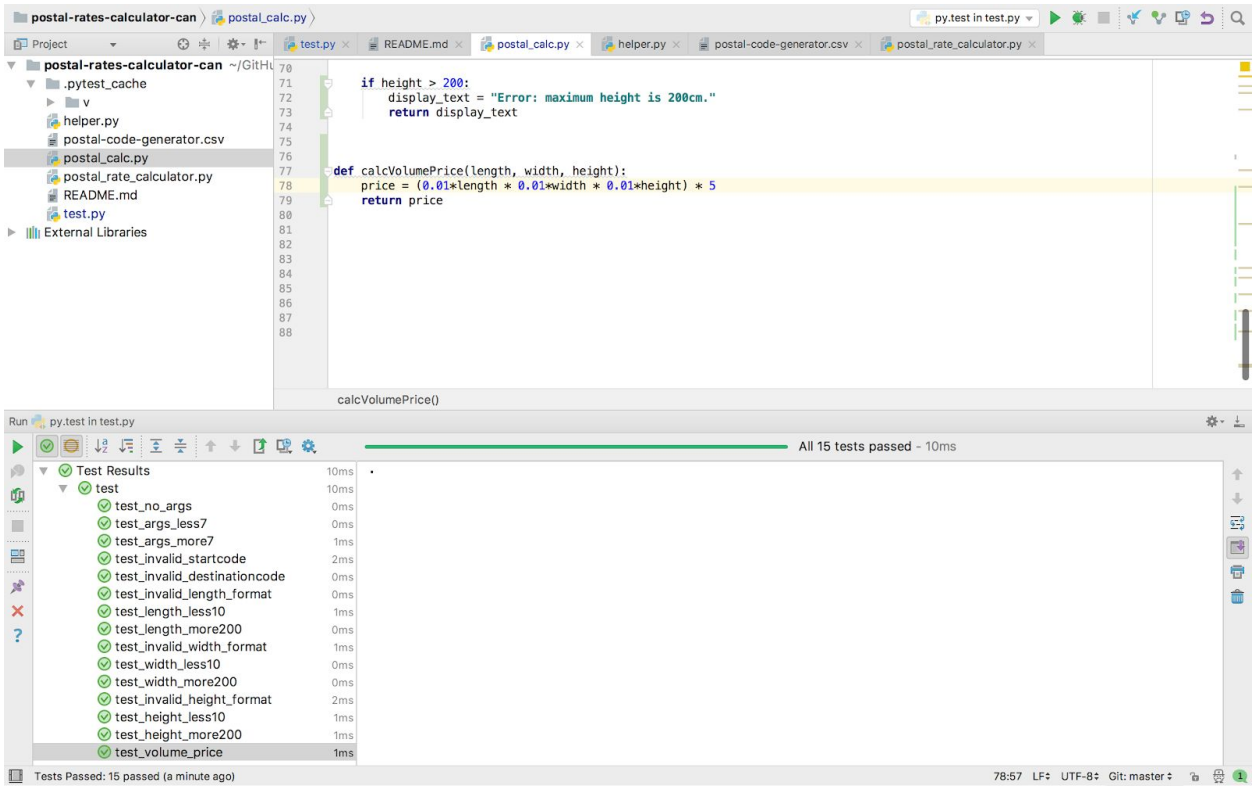
Test 15 Code:



Test 15 Succeed:



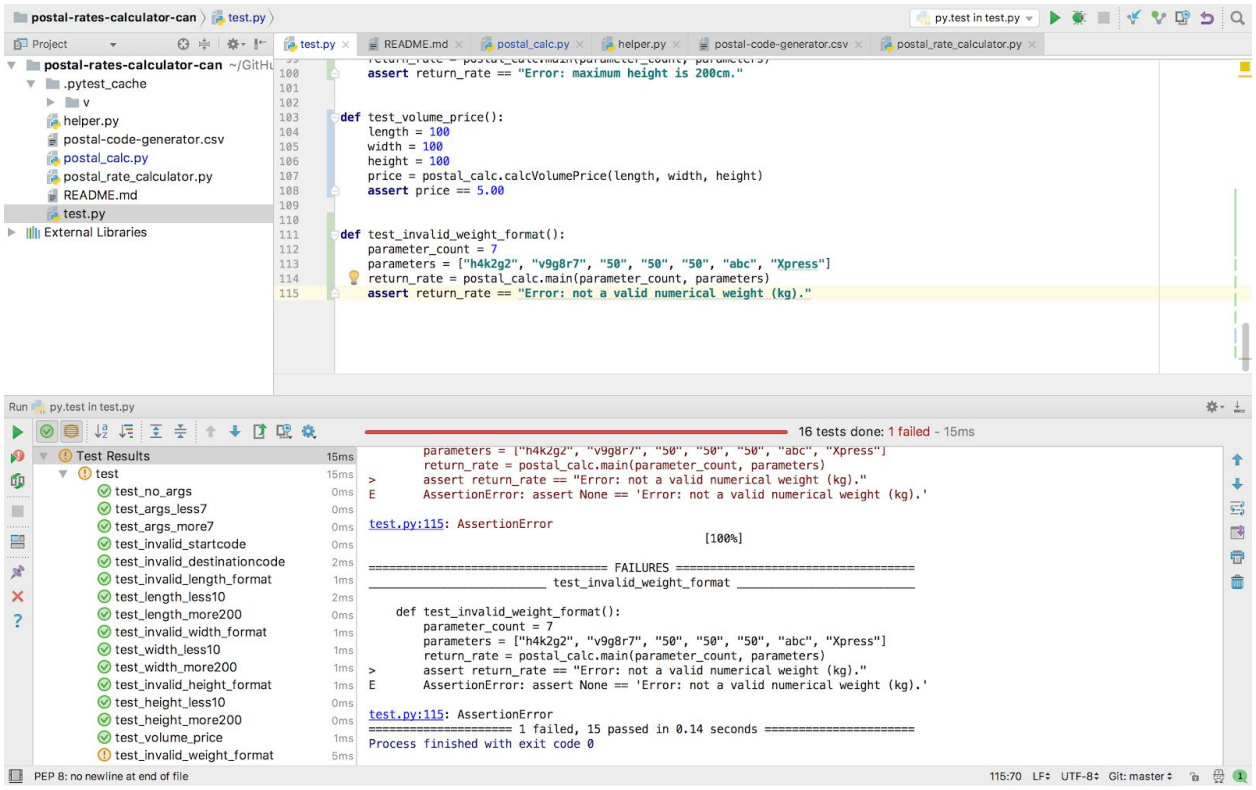
Test 15 Code with changes:



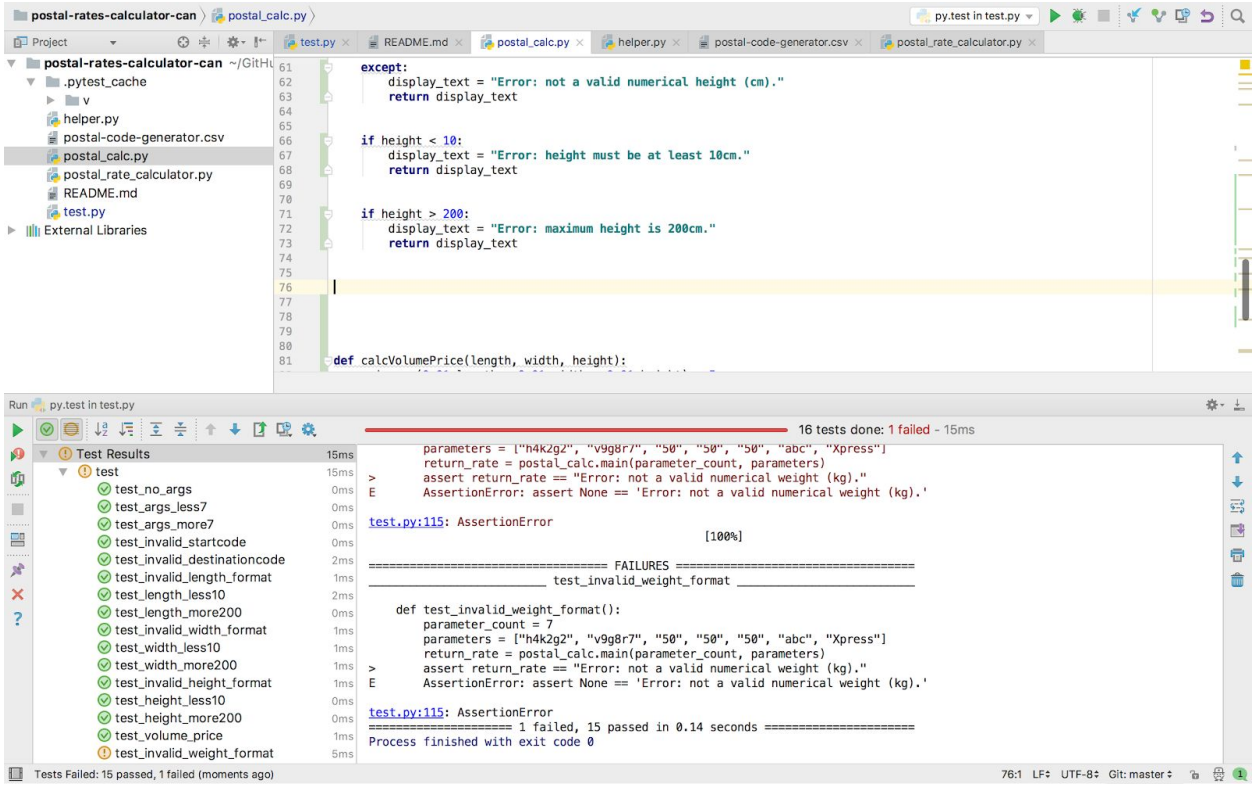
Test 16: Invalid weight format

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.

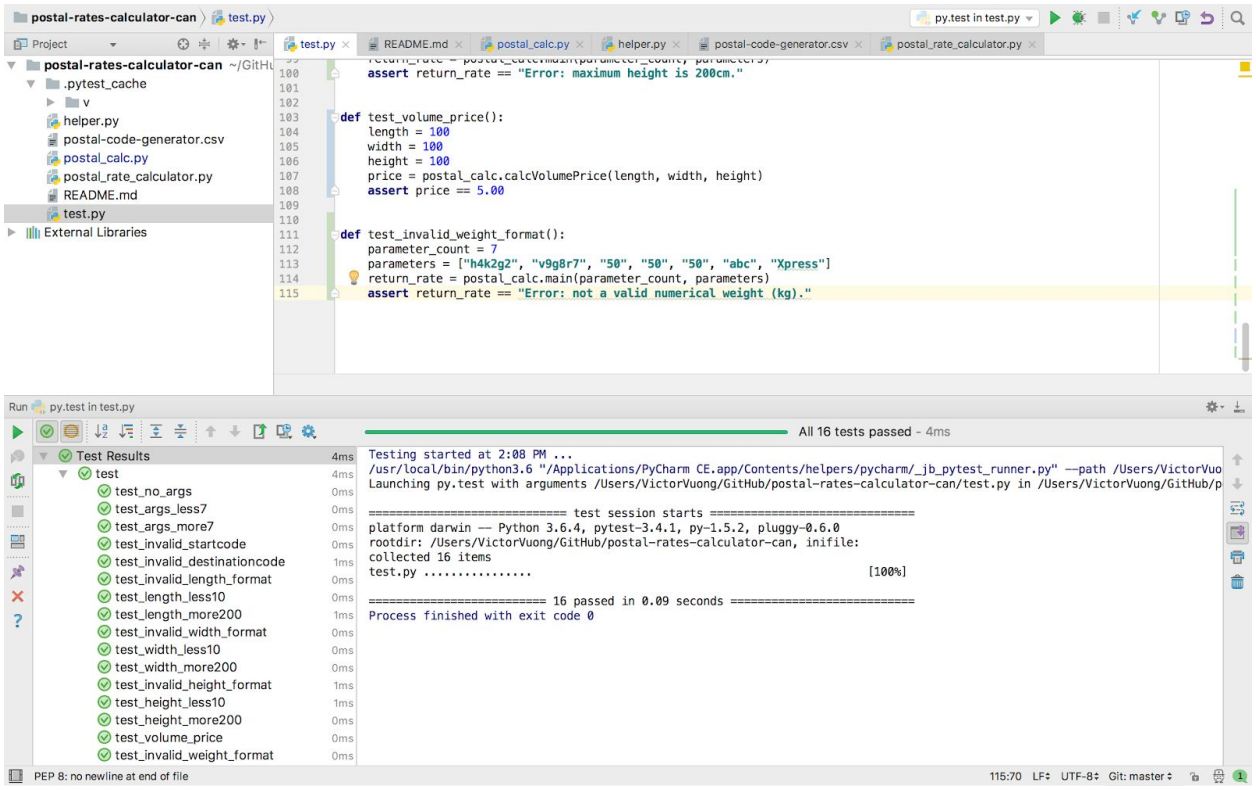
Test 16 Fail:



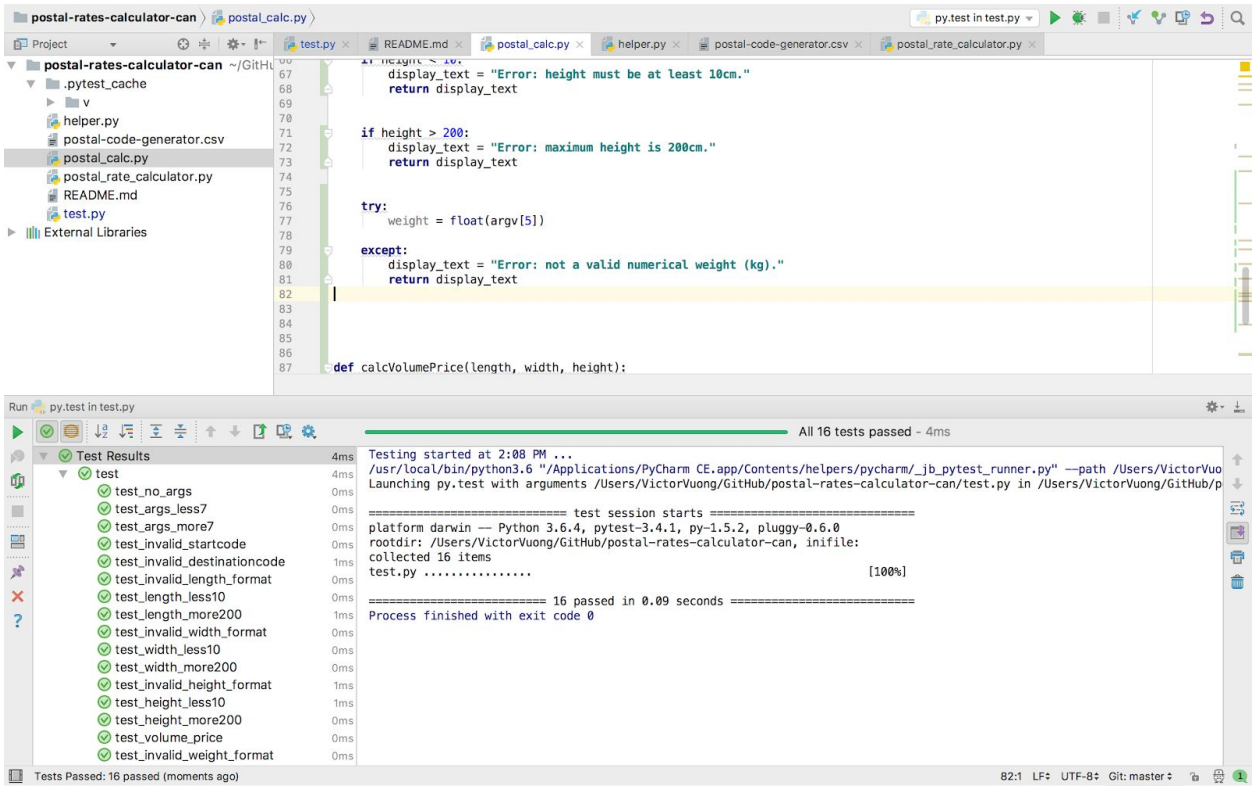
Test 16 Code:



Test 16 Succeed:



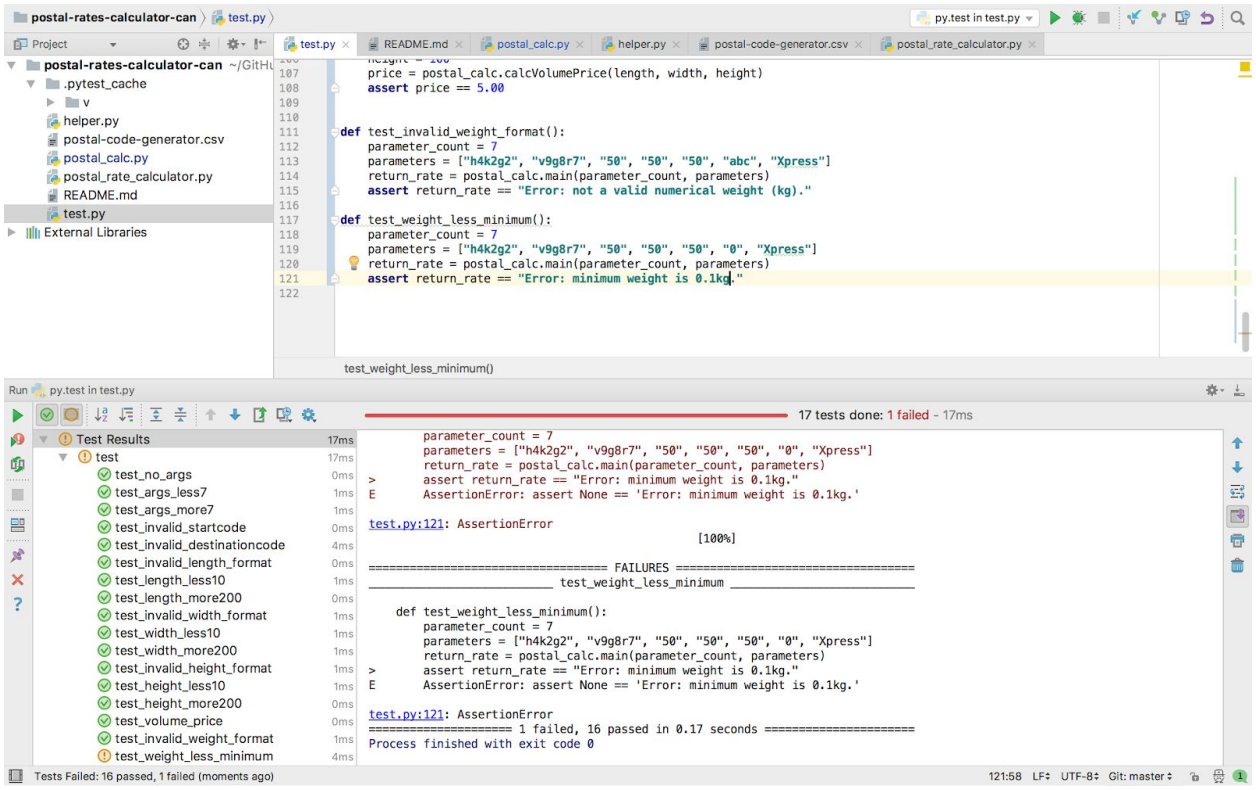
Test 16 Code with changes:



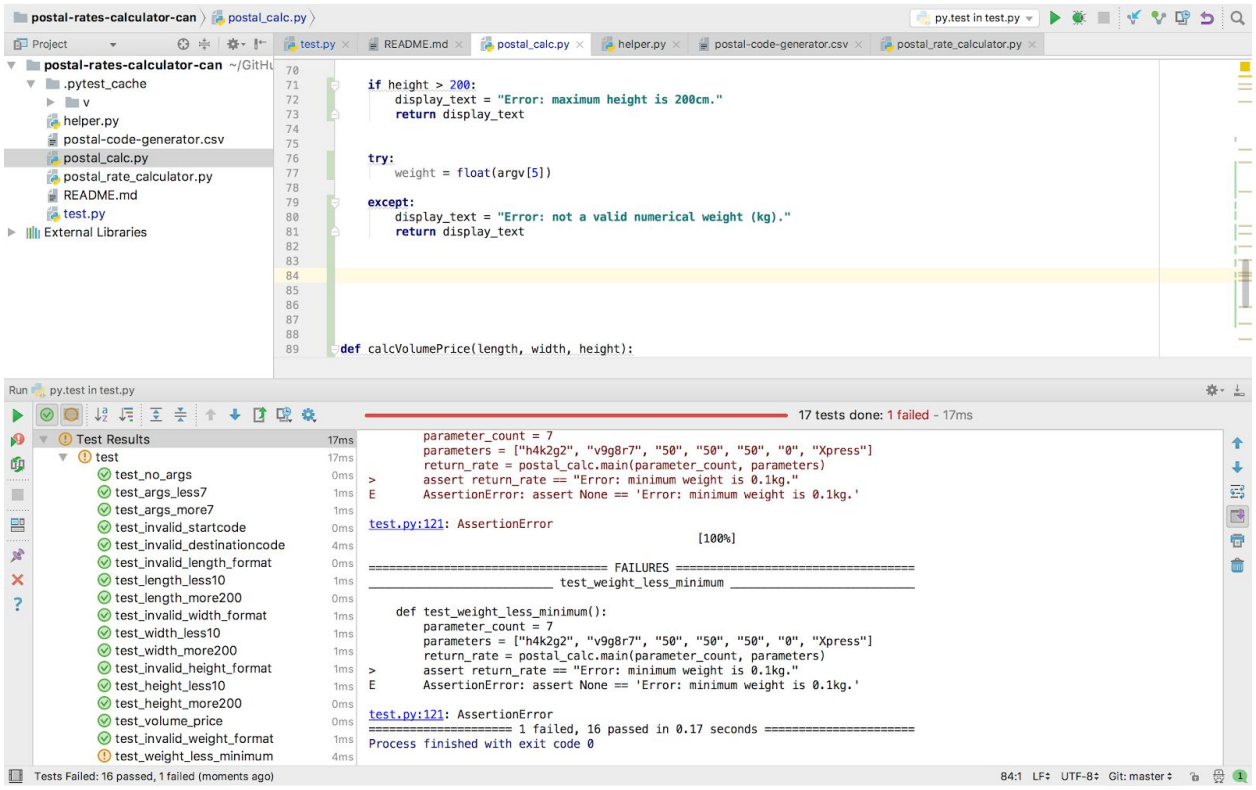
Test 17: Weight too low

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

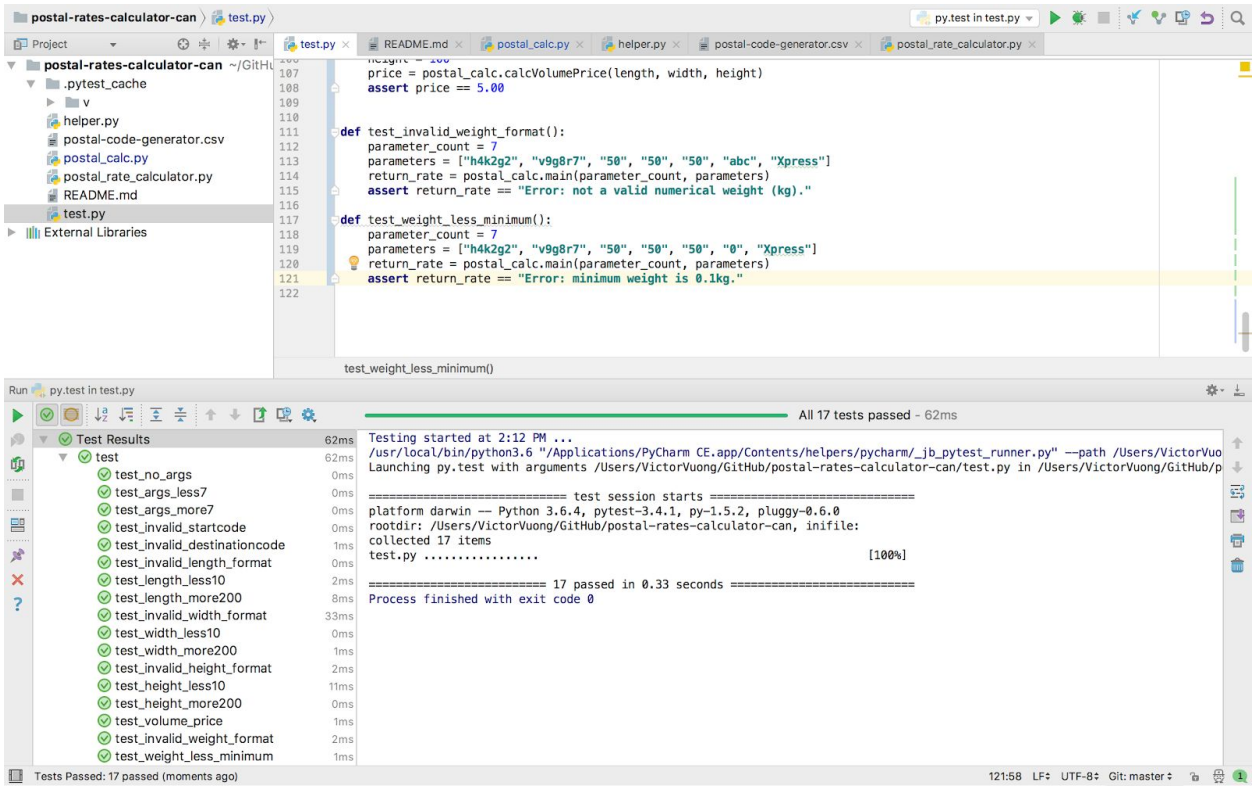
Test 17 Fail:



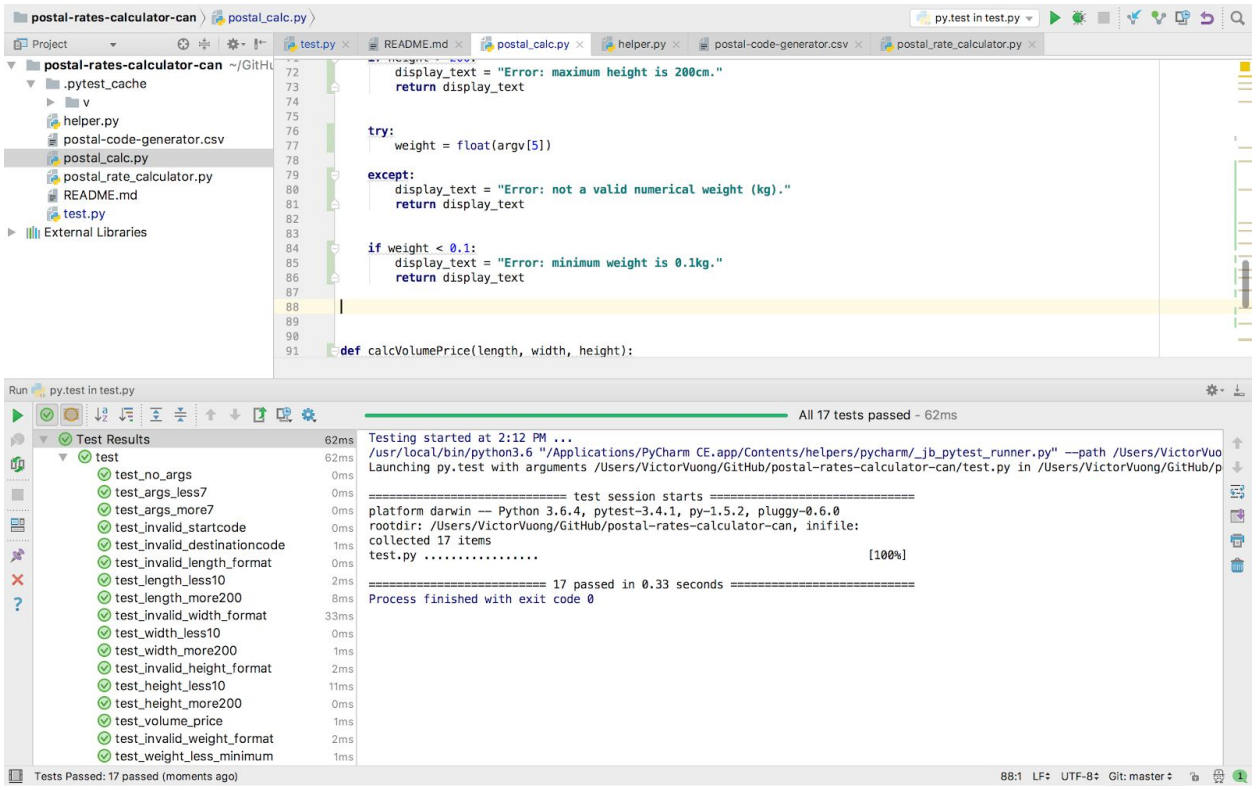
Test 17 Code:



Test 17 Succeed:



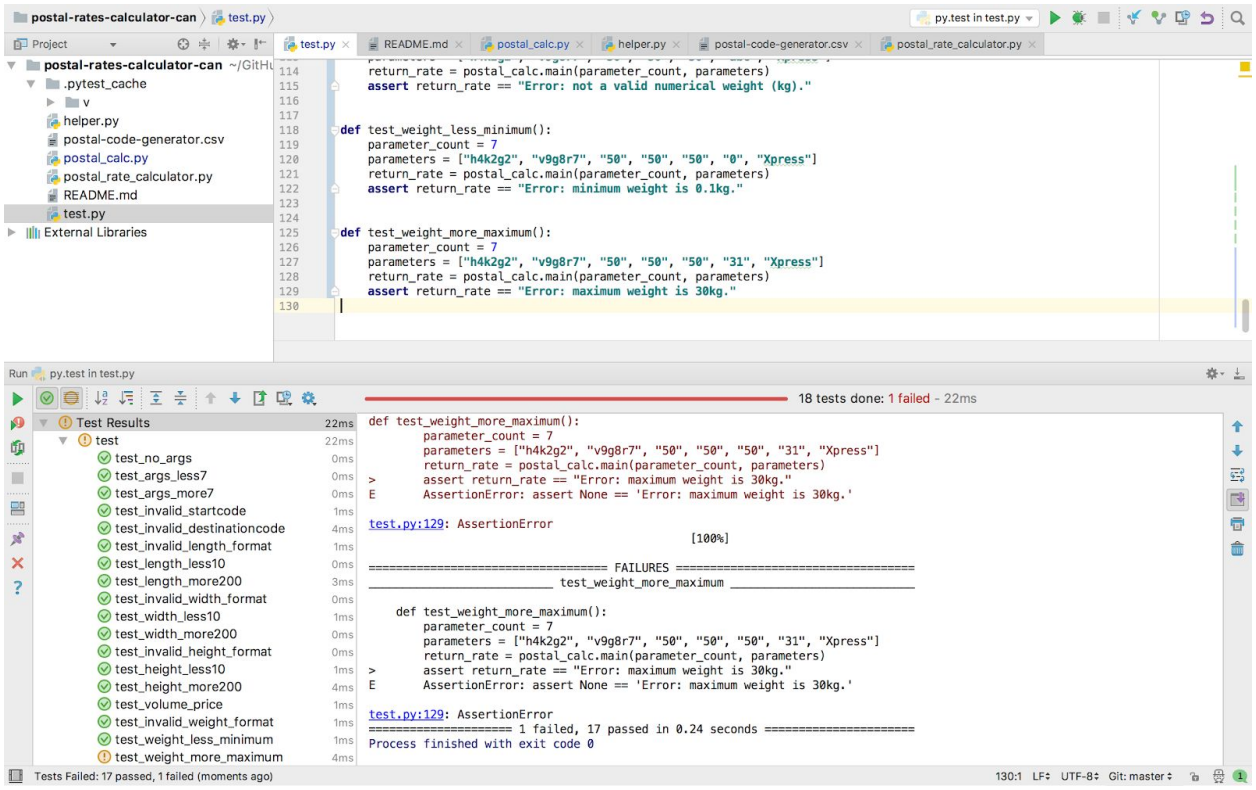
Test 17 Code with changes:



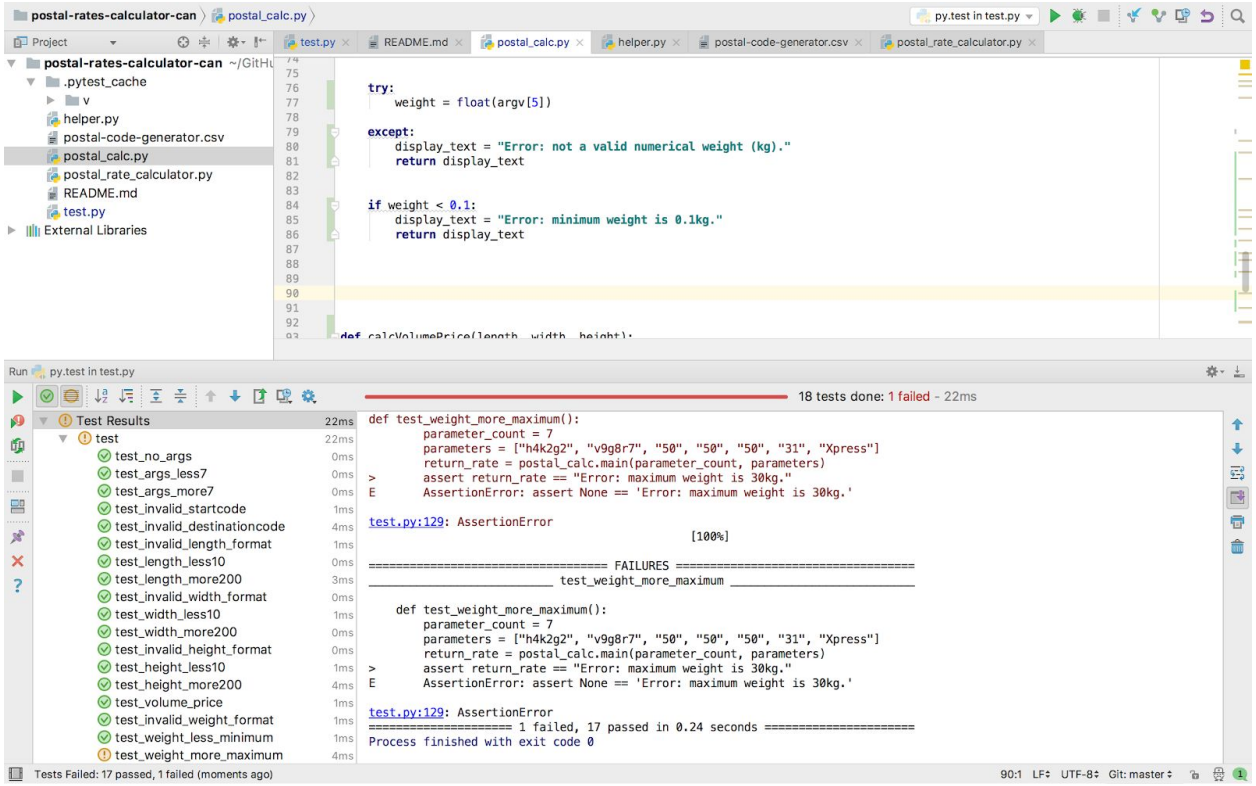
Test 18: Weight too high

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.

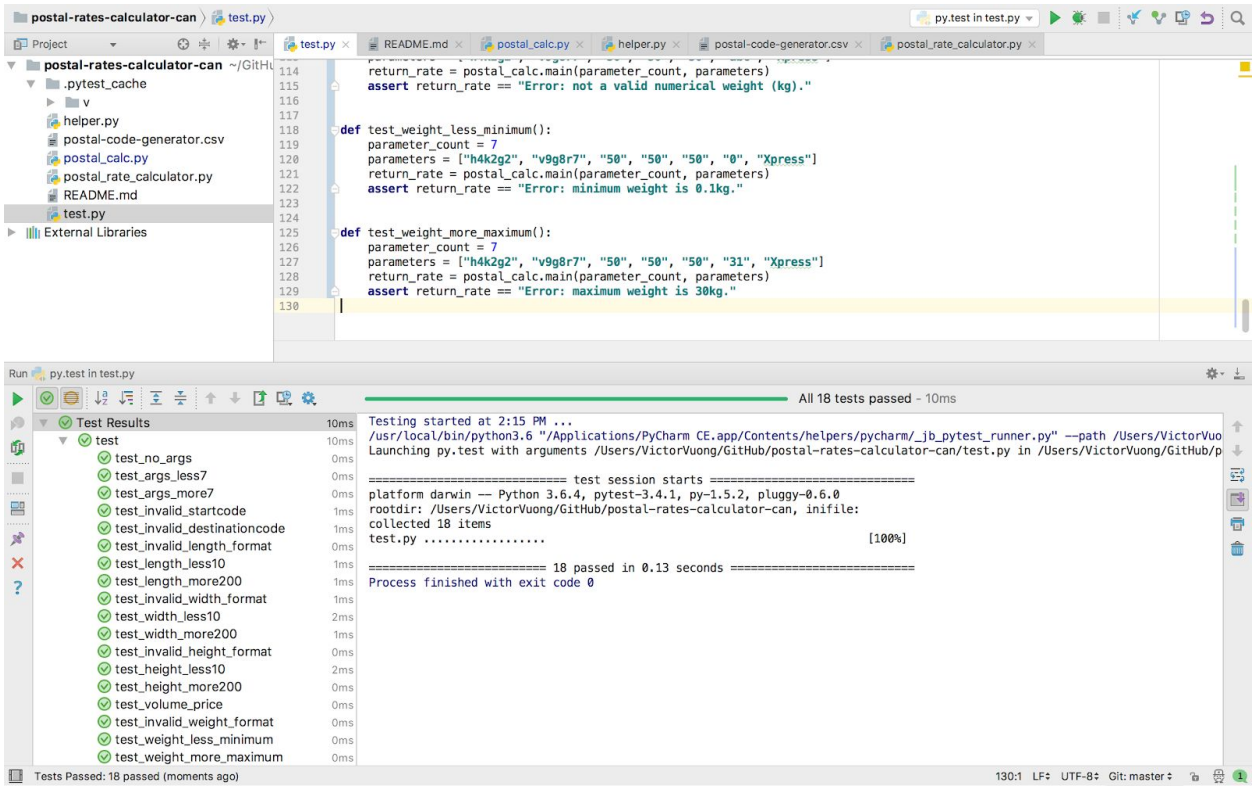
Test 18 Fail:



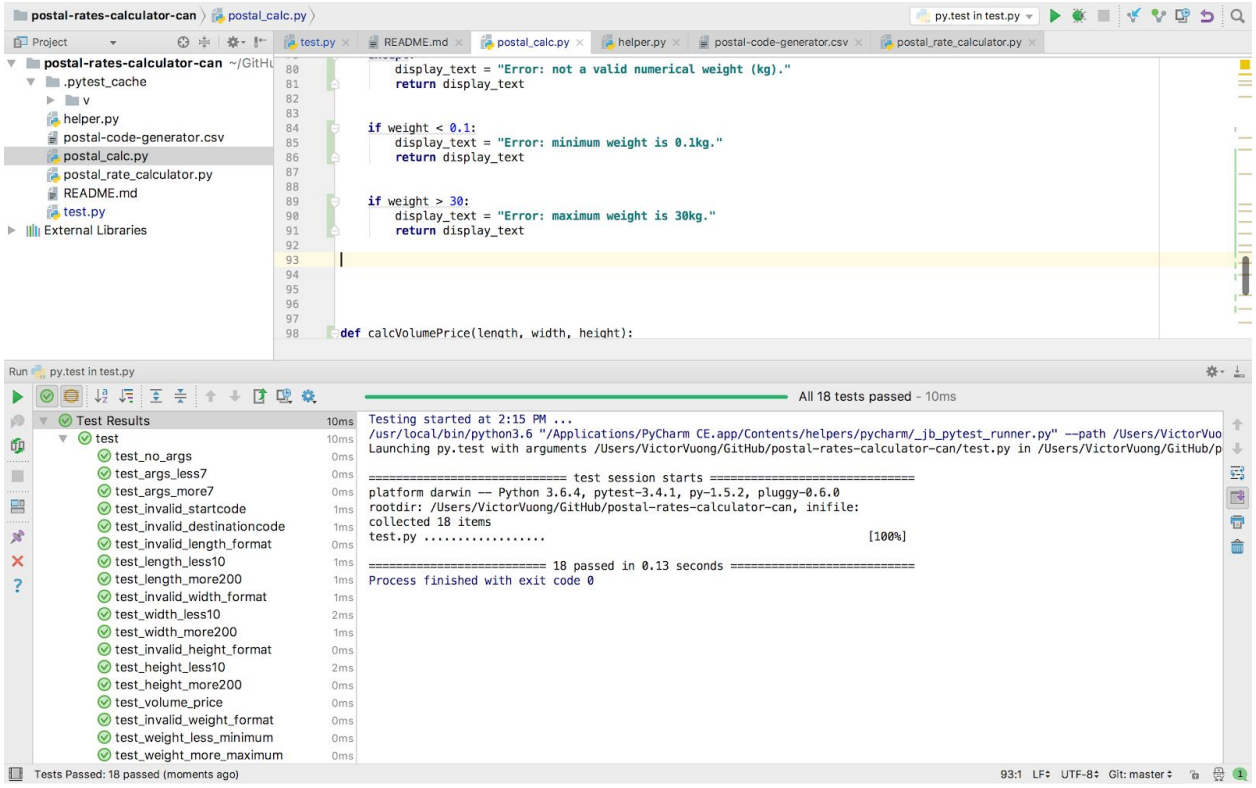
Test 18 Code:



Test 18 Succeed:



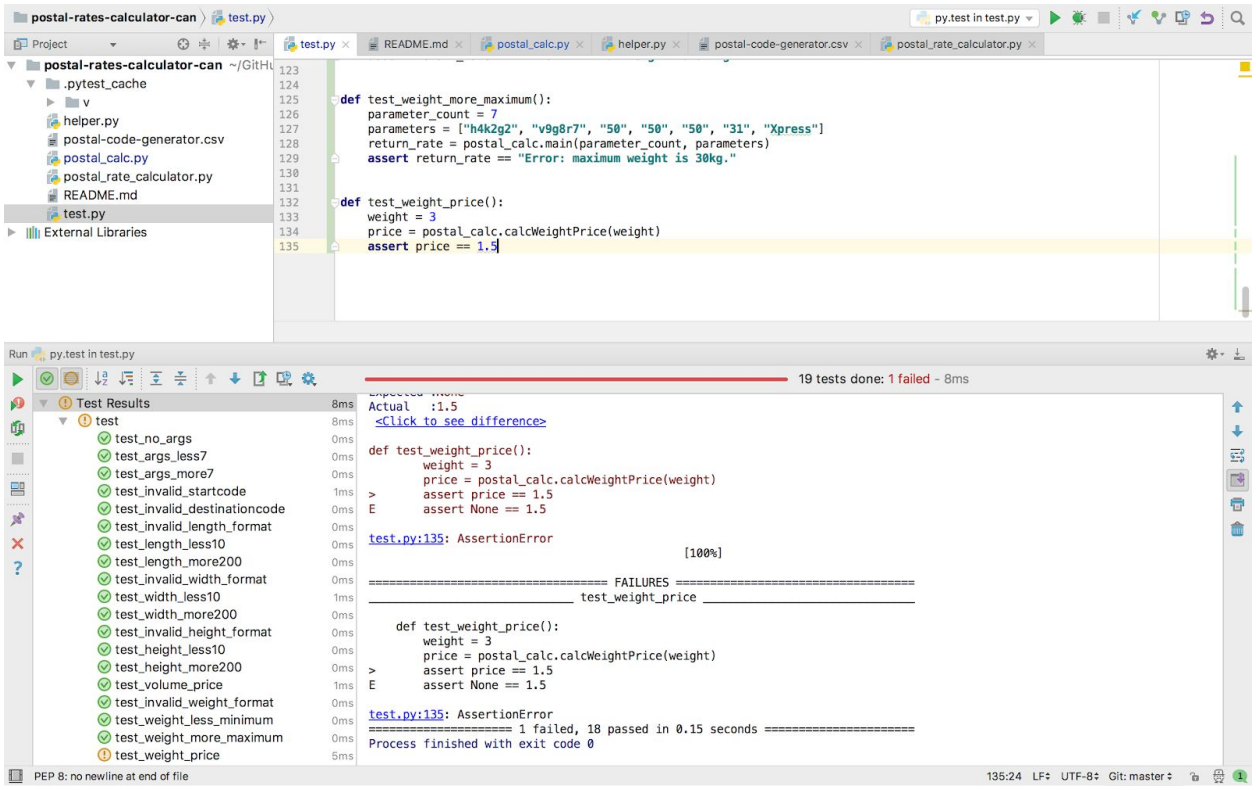
Test 18 Code with changes:



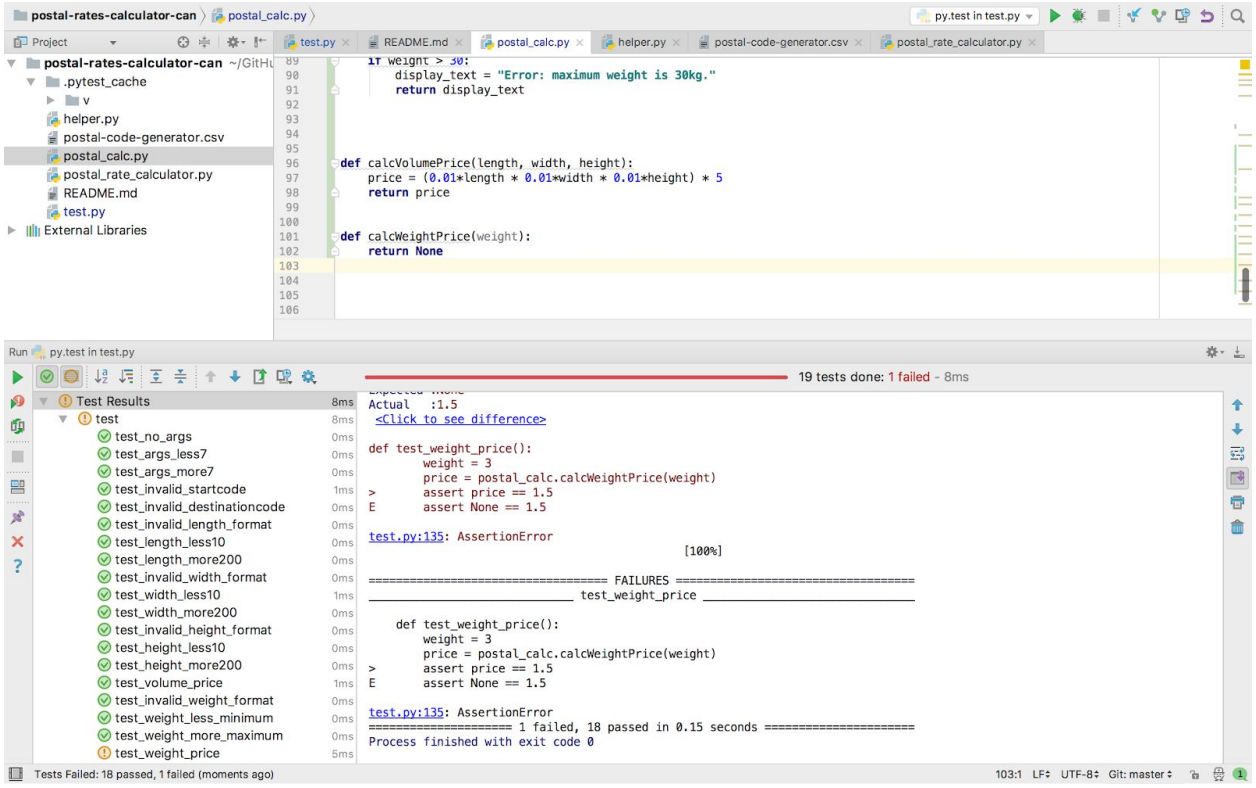
Test 19: Correct weight price

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.

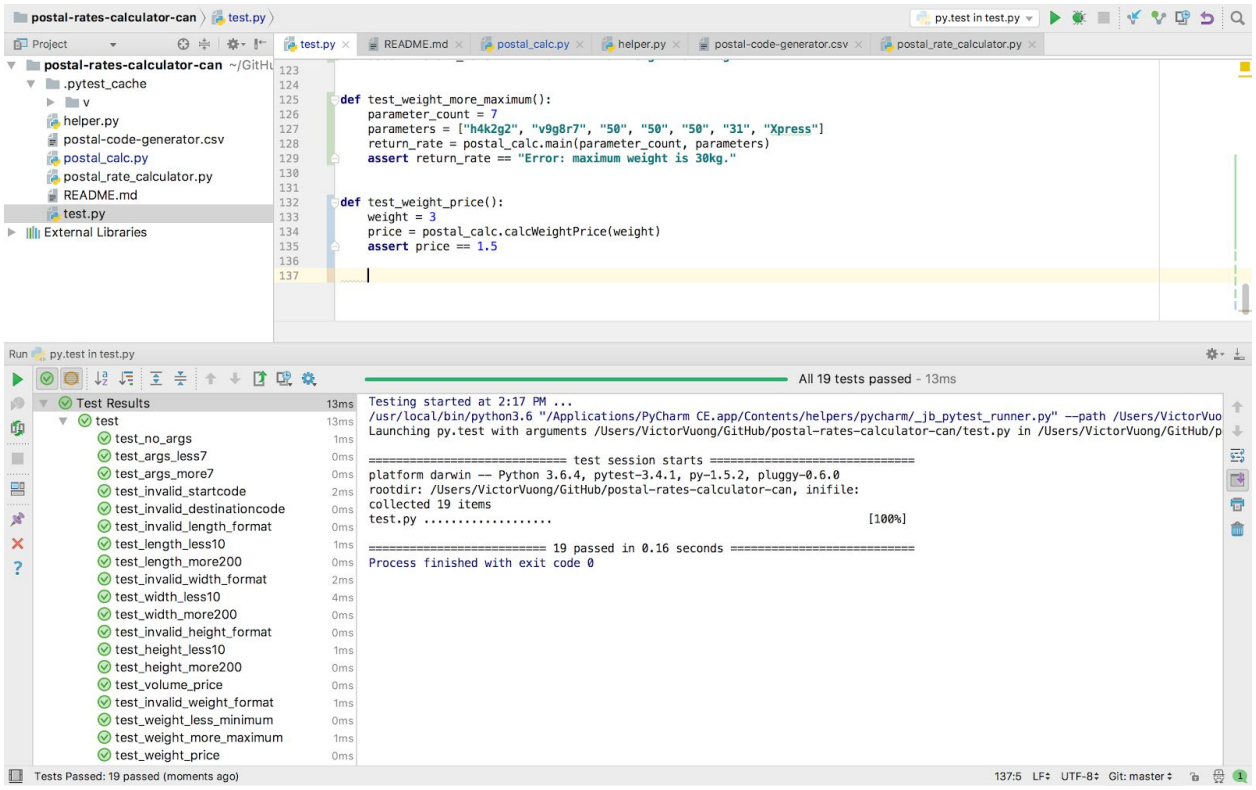
Test 19 Fail:



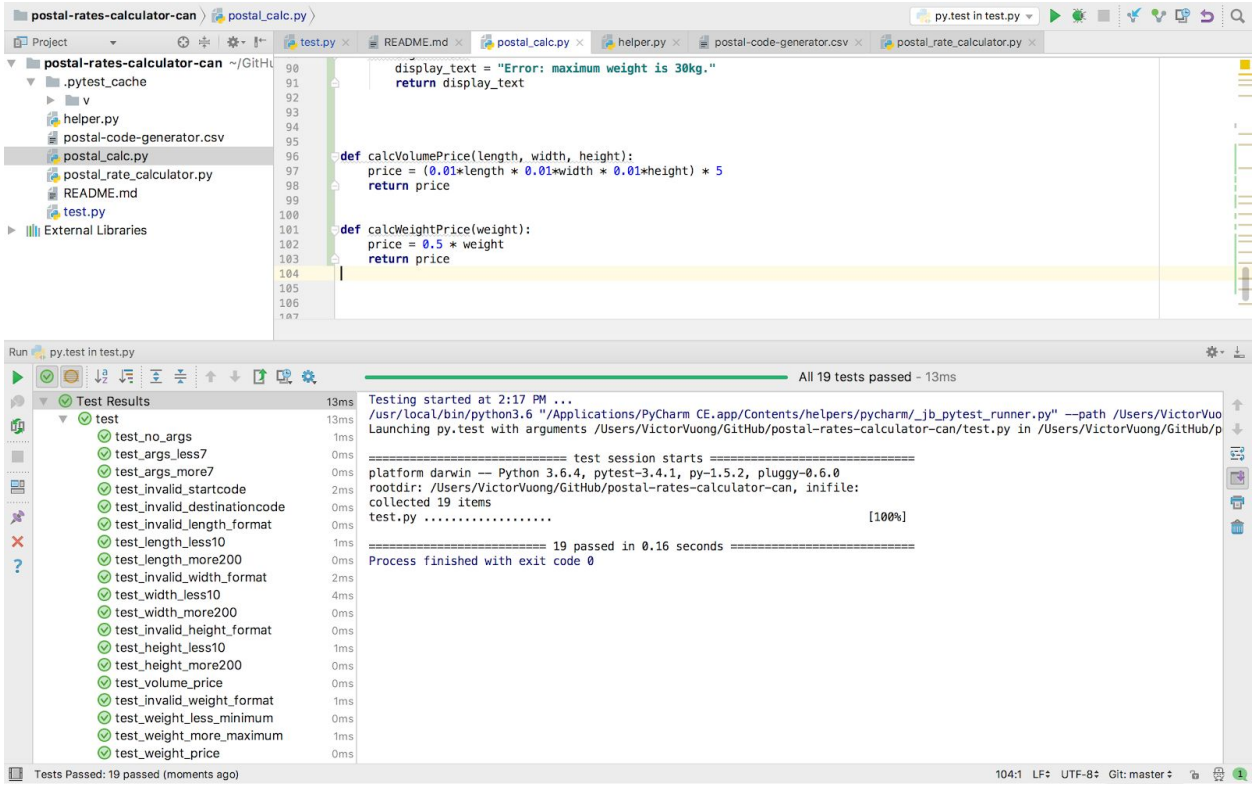
Test 19 Code:



Test 19 Succeed:



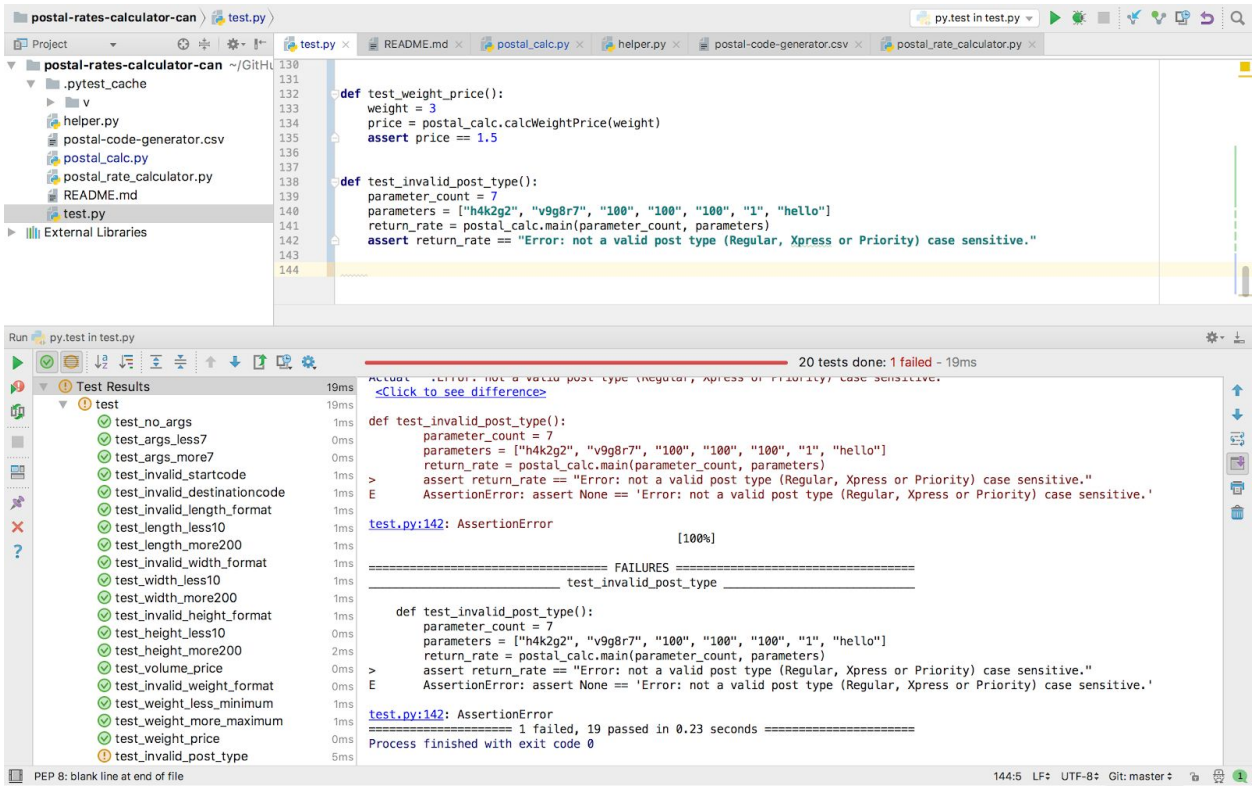
Test 19 Code with changes:



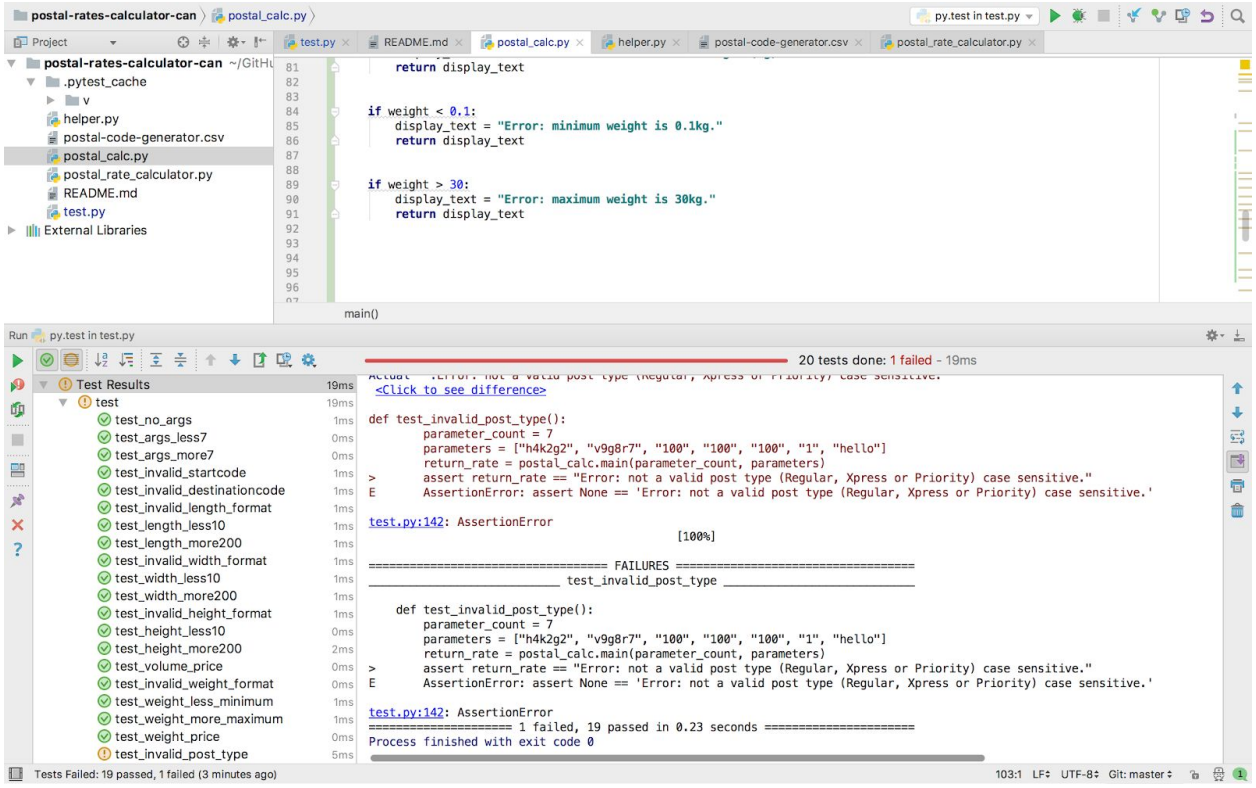
Test 20: Invalid post-type

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.

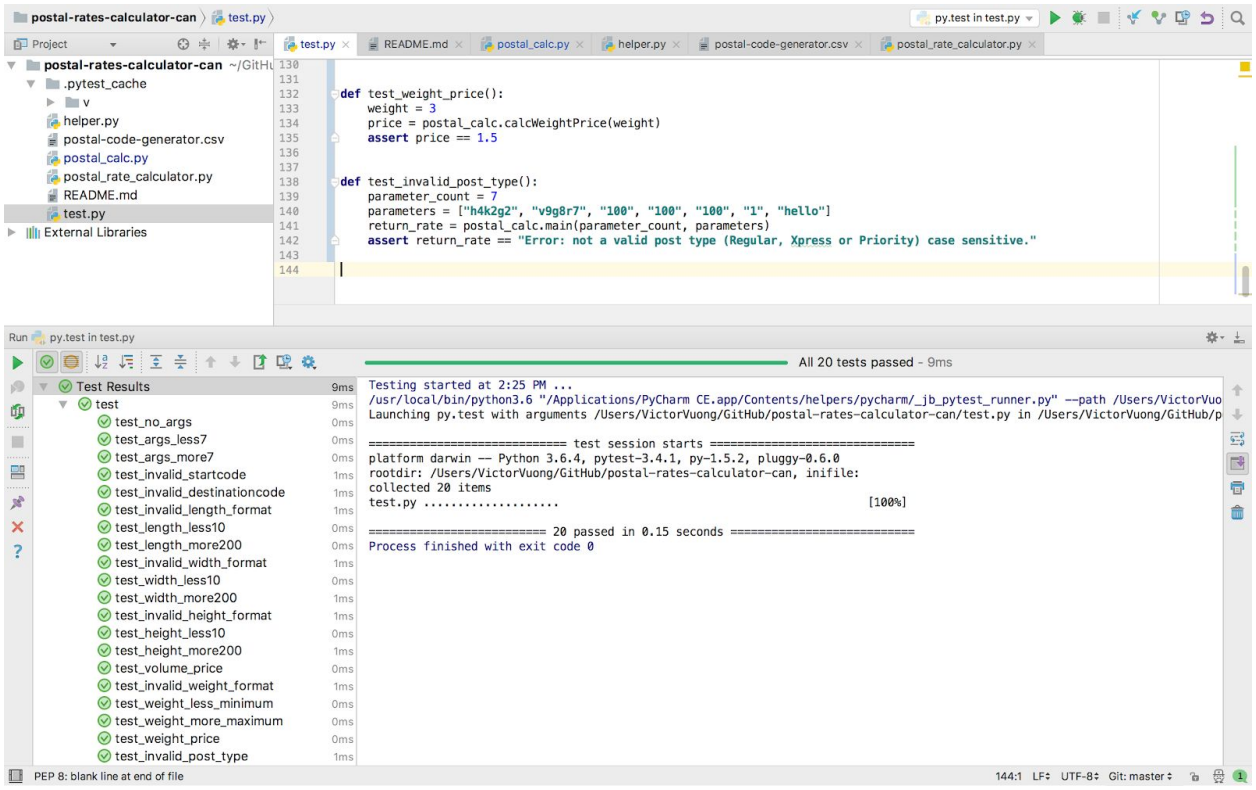
Test 20 Fail:



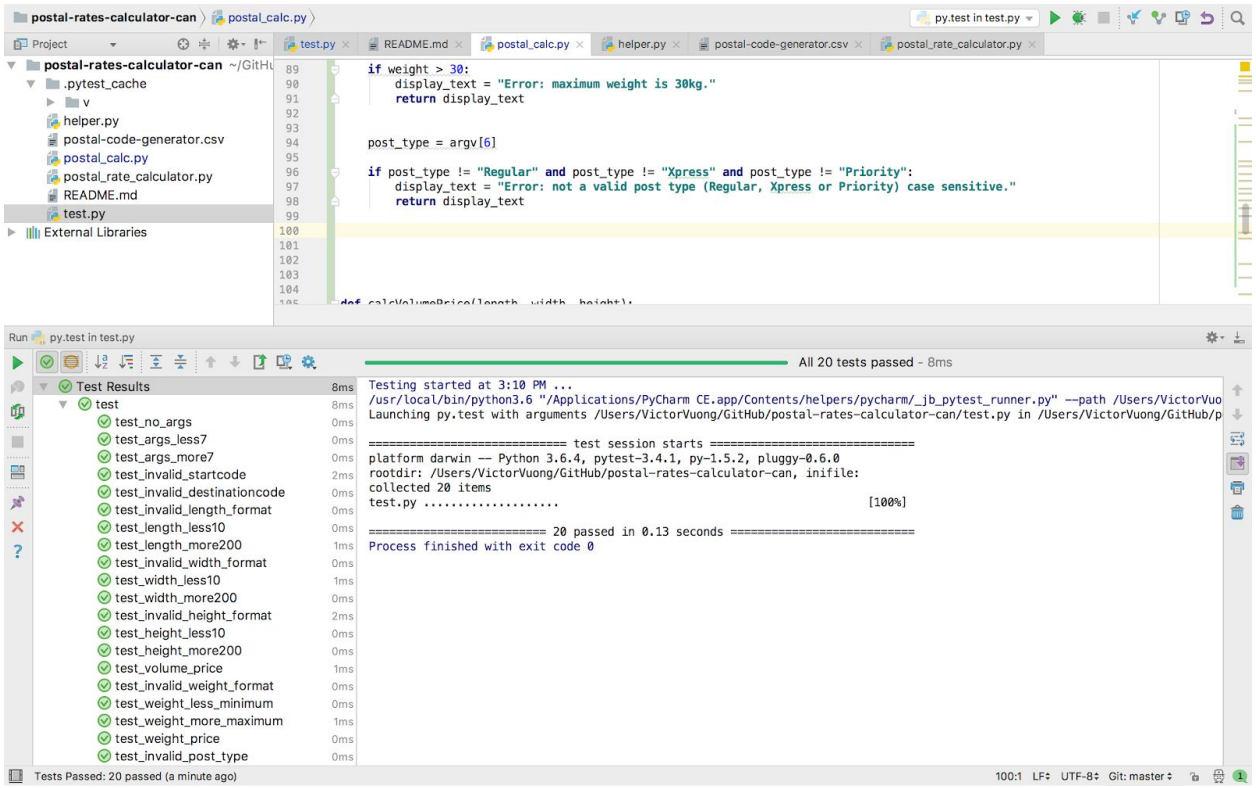
Test 20 Code:



Test 20 Succeed:



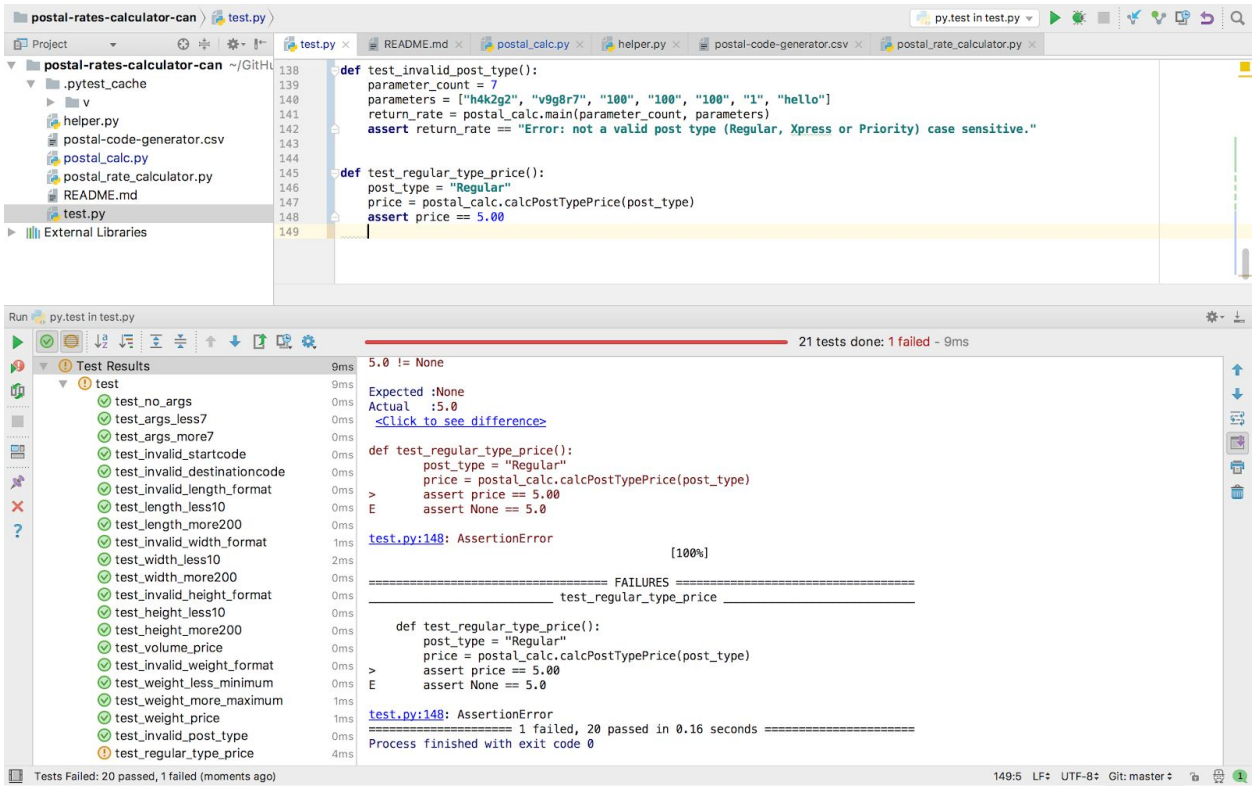
Test 20 Code with changes:



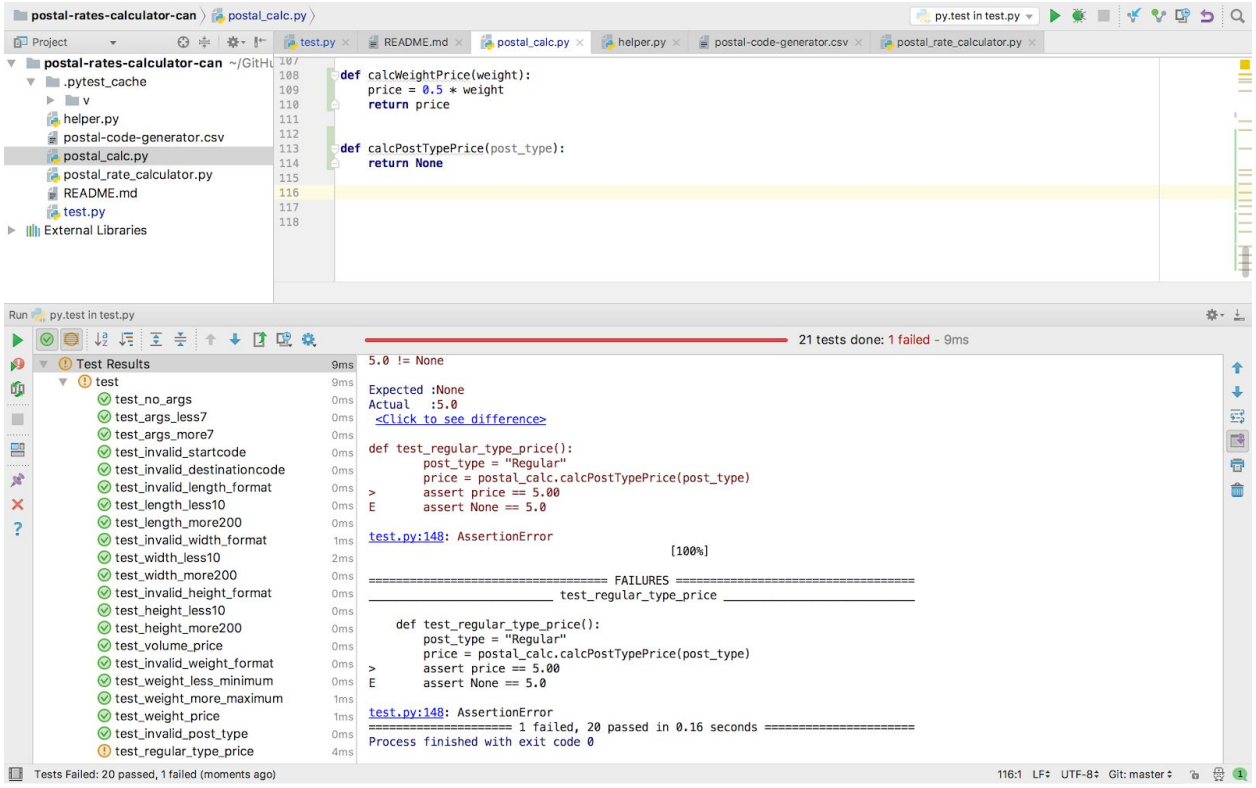
Test 21: Correct Regular post-type cost

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.

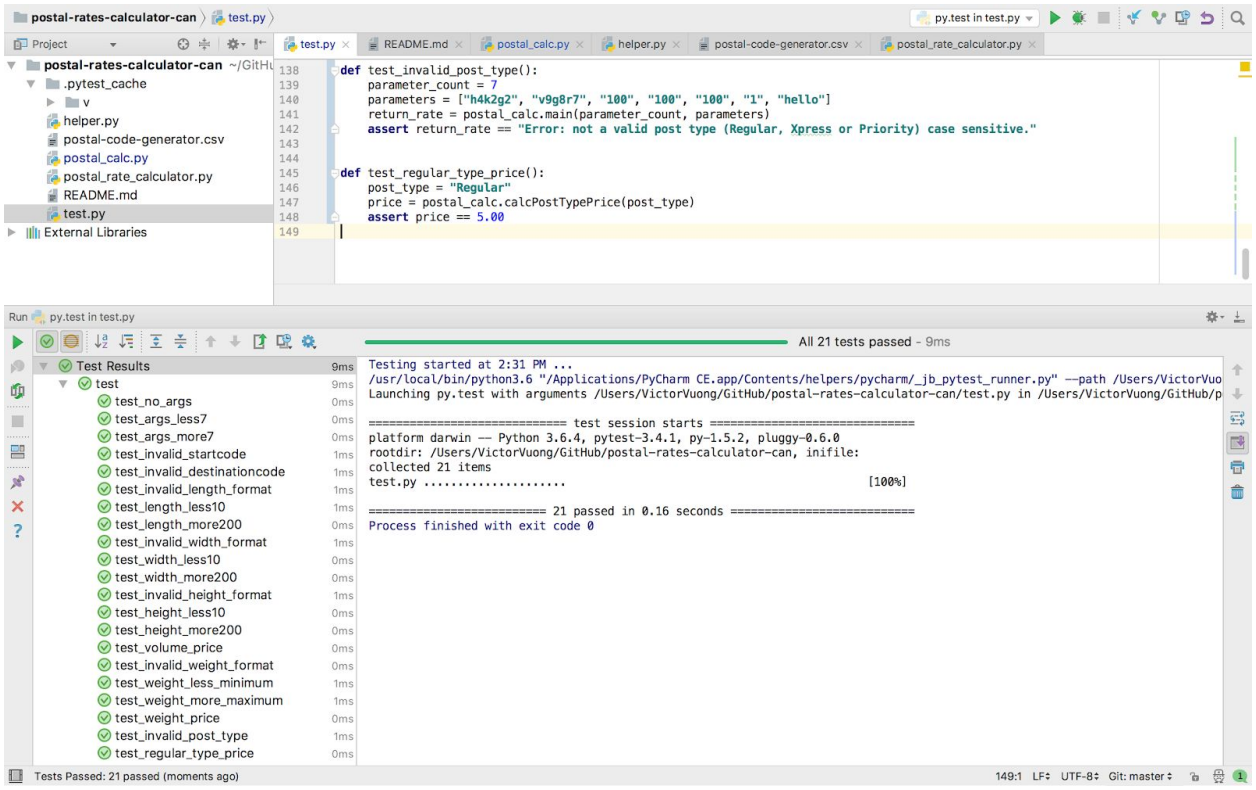
Test 21 Fail:



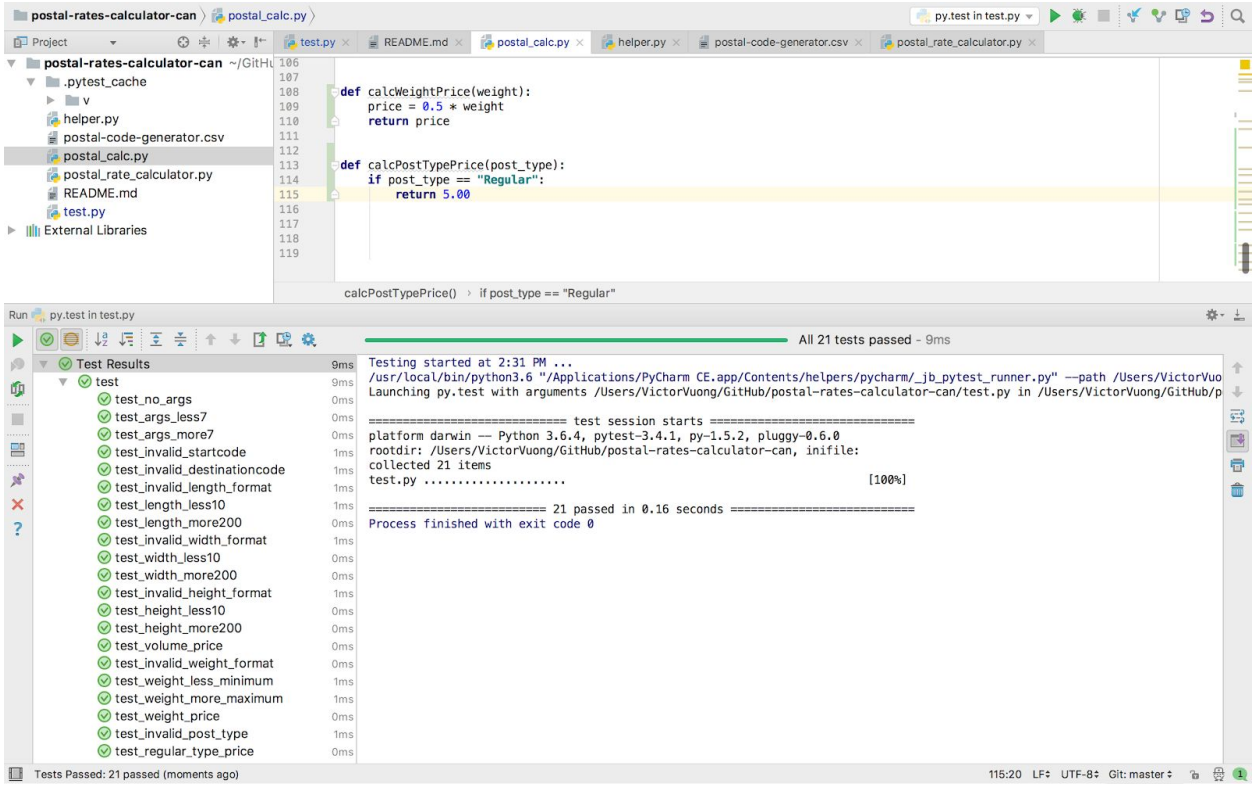
Test 21 Code:



Test 21 Succeed:



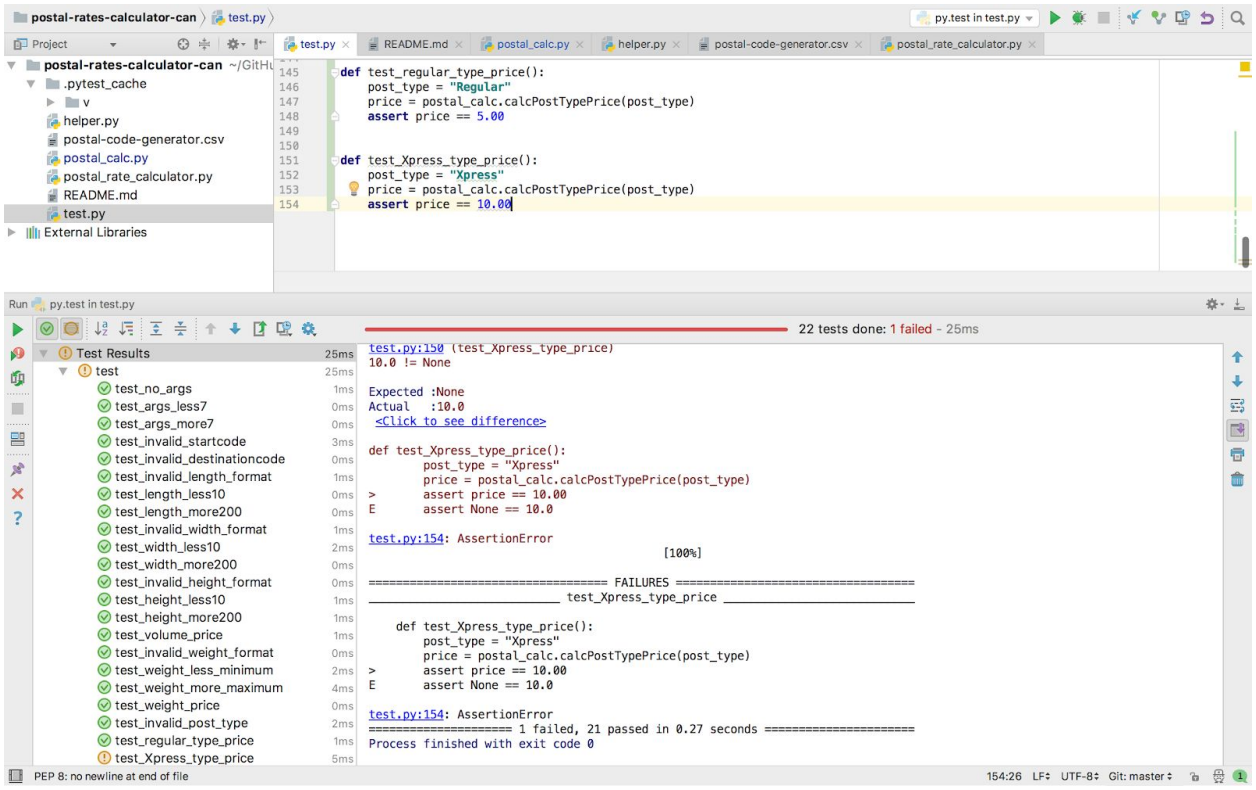
Test 21 Code with changes:



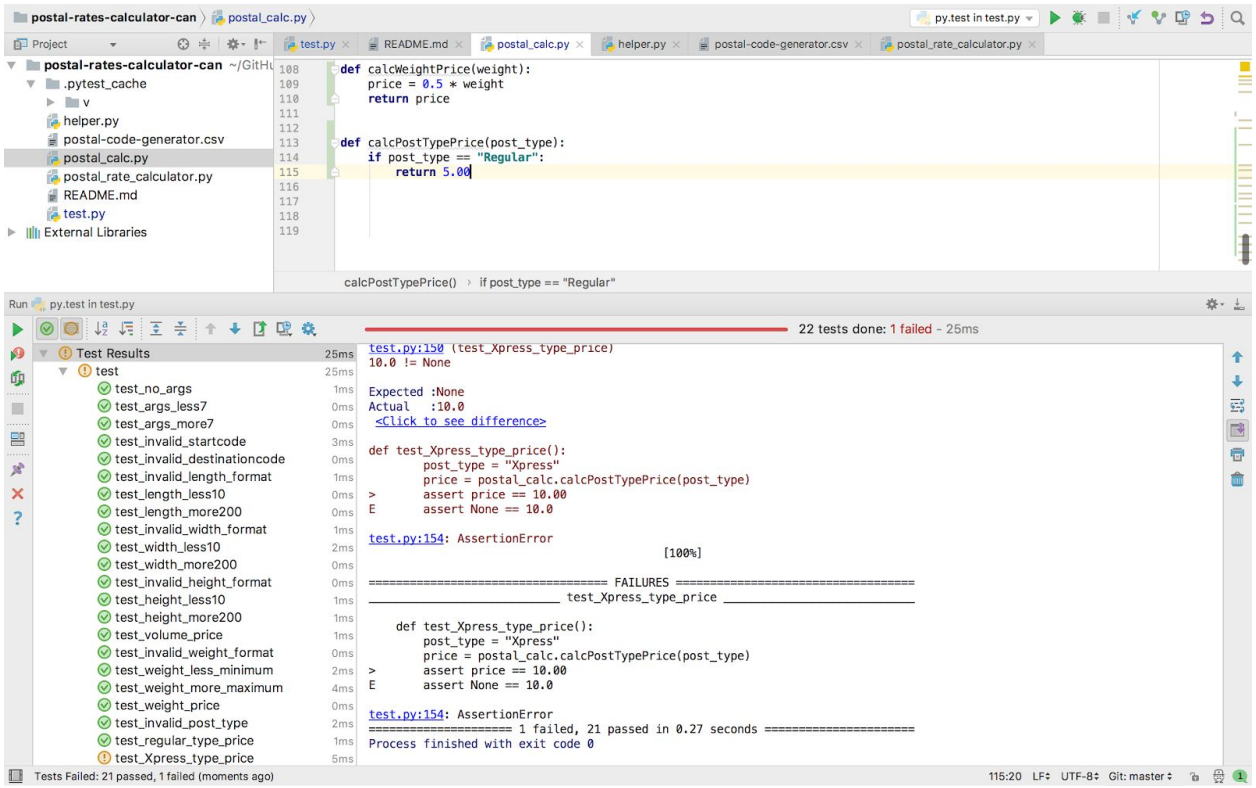
Test 22: Correct Xpress post-type cost

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 the Xpress post-type has a fixed price of $10.00.

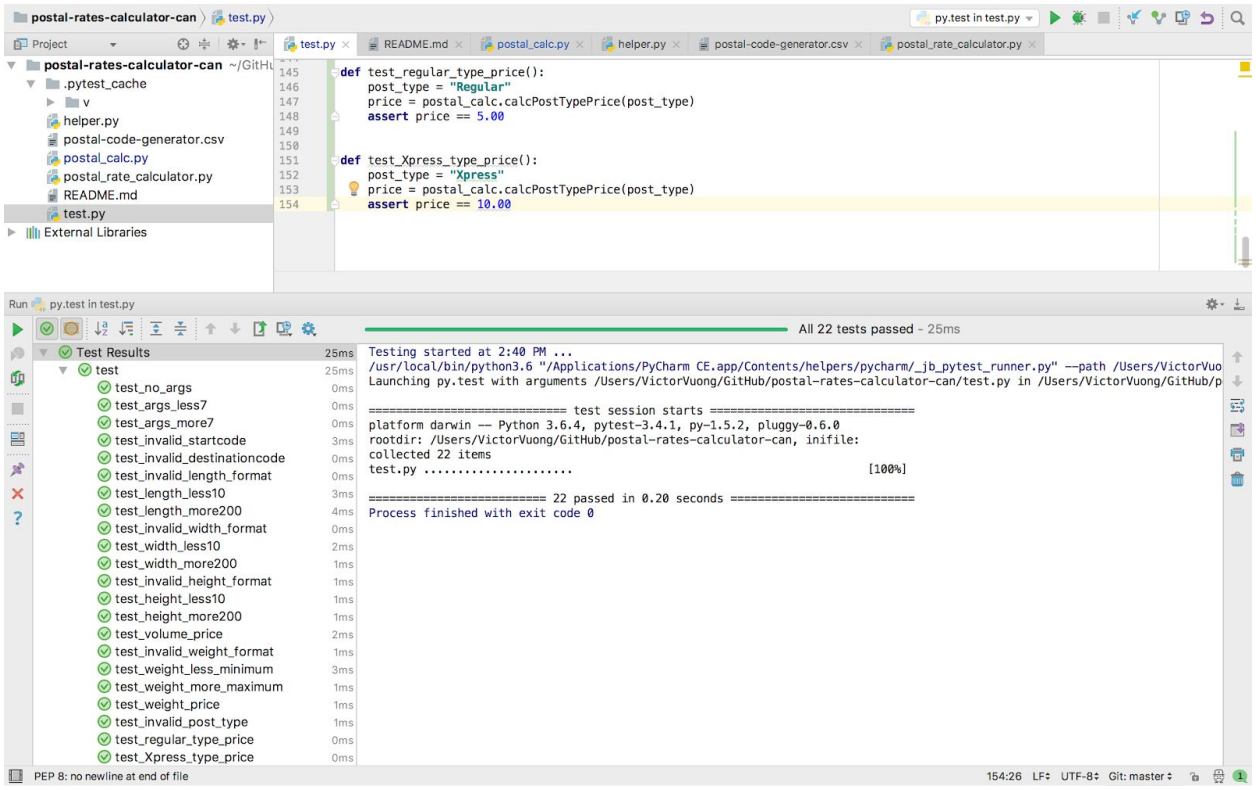
Test 22 Fail:



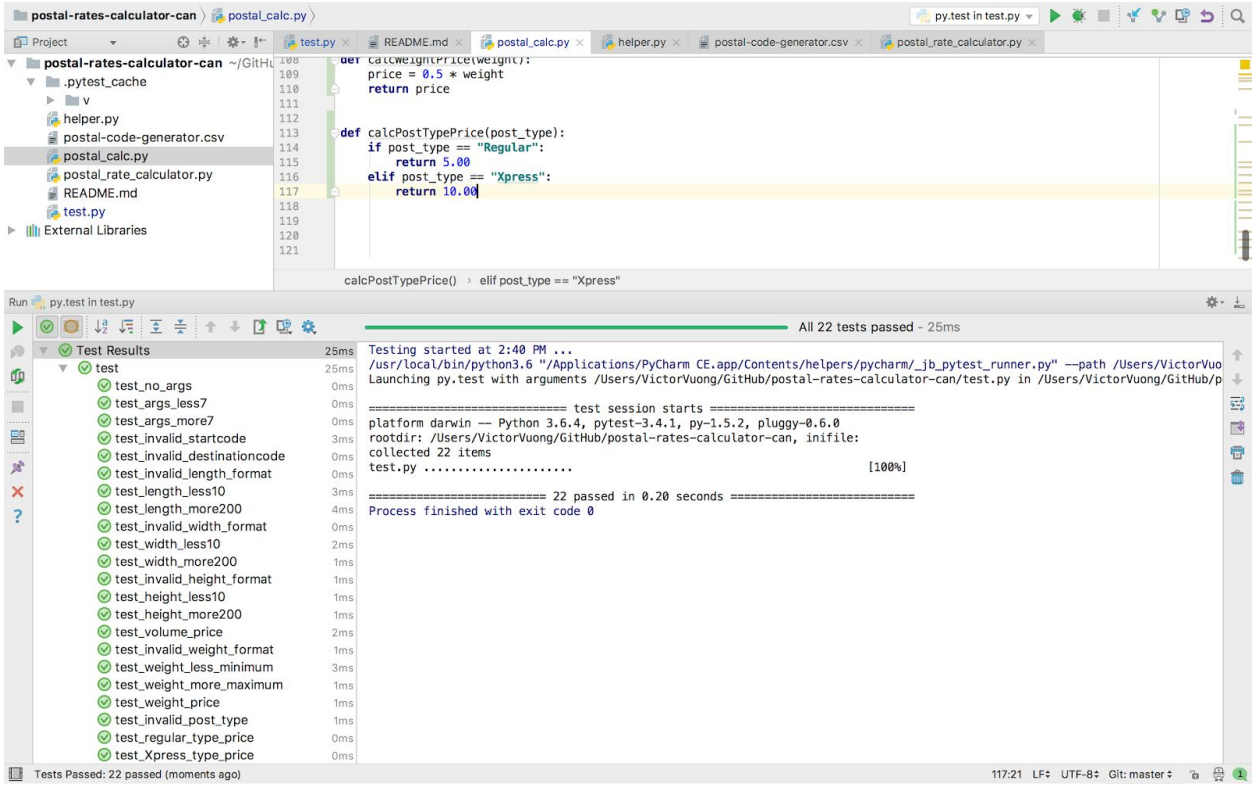
Test 22 Code:



Test 22 Succeed:



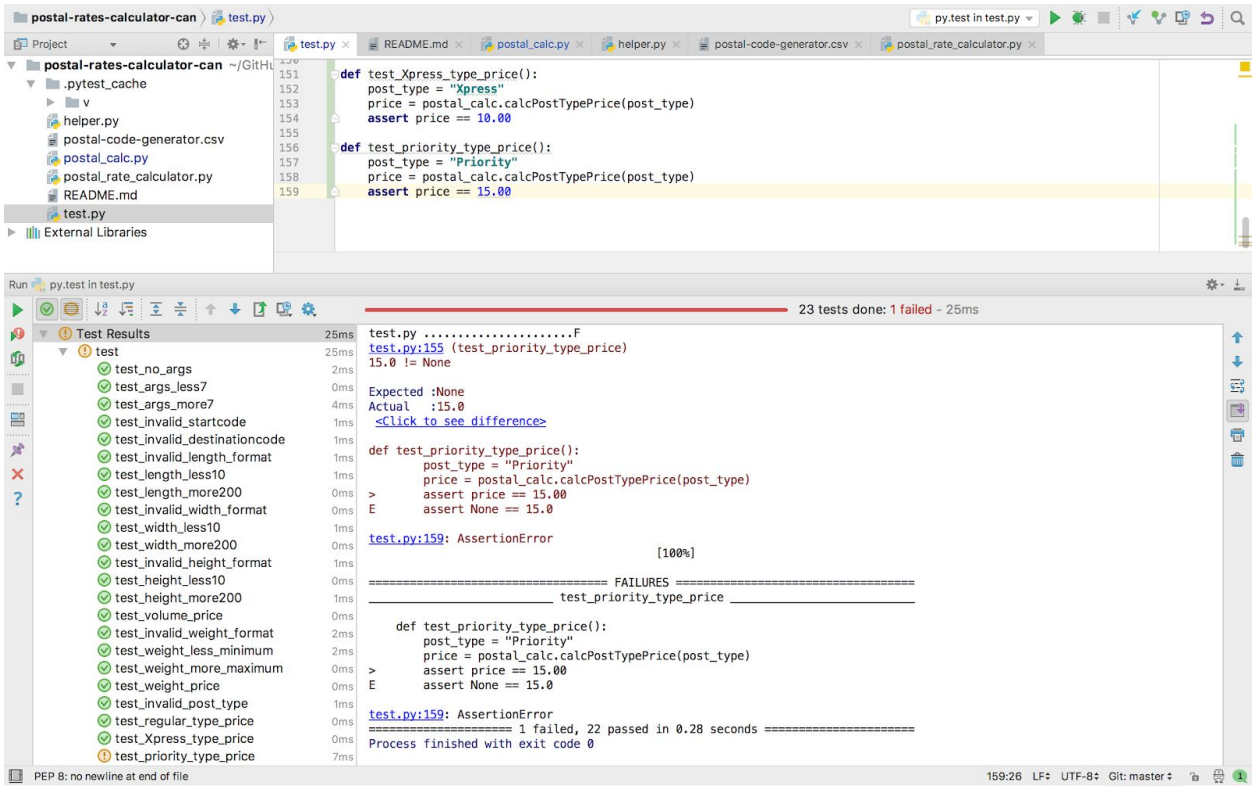
Test 22 Code with changes:



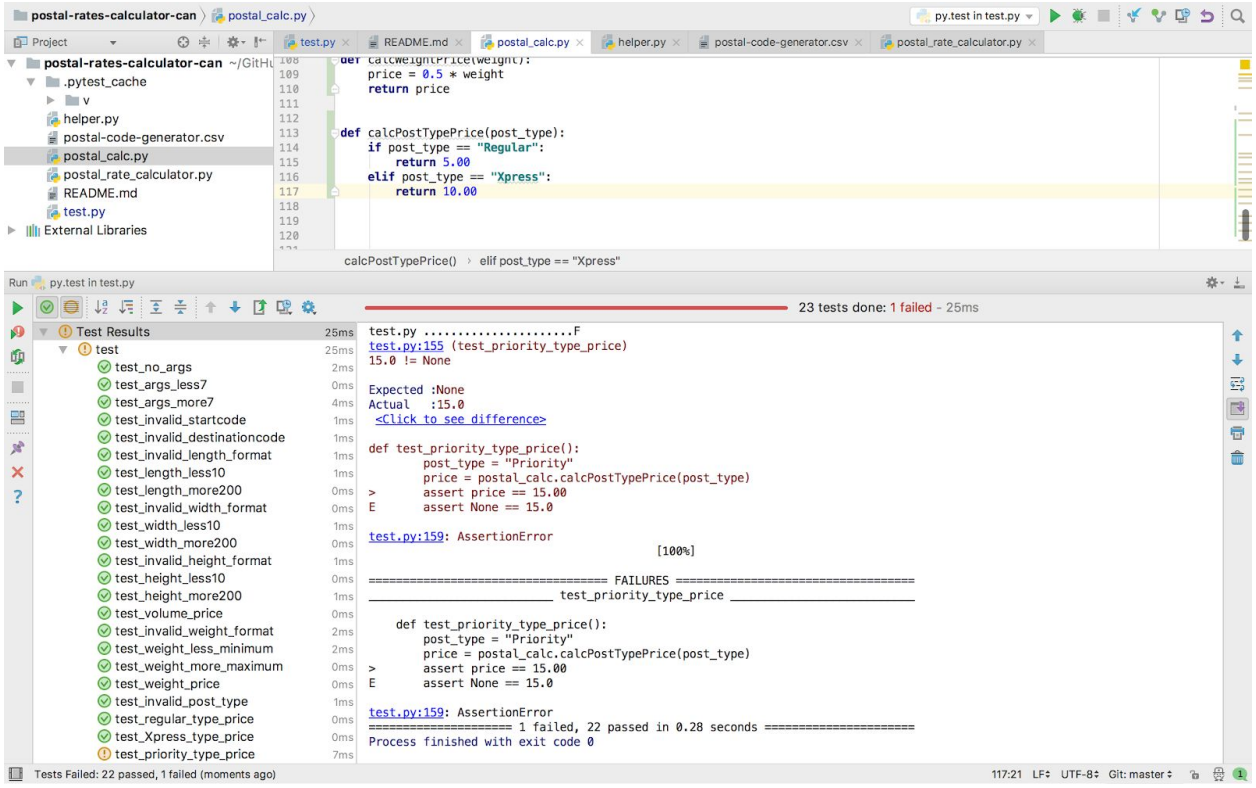
Test 23: Correct Priority post-type cost

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 the Priority post-type has a fixed price of $15.00.

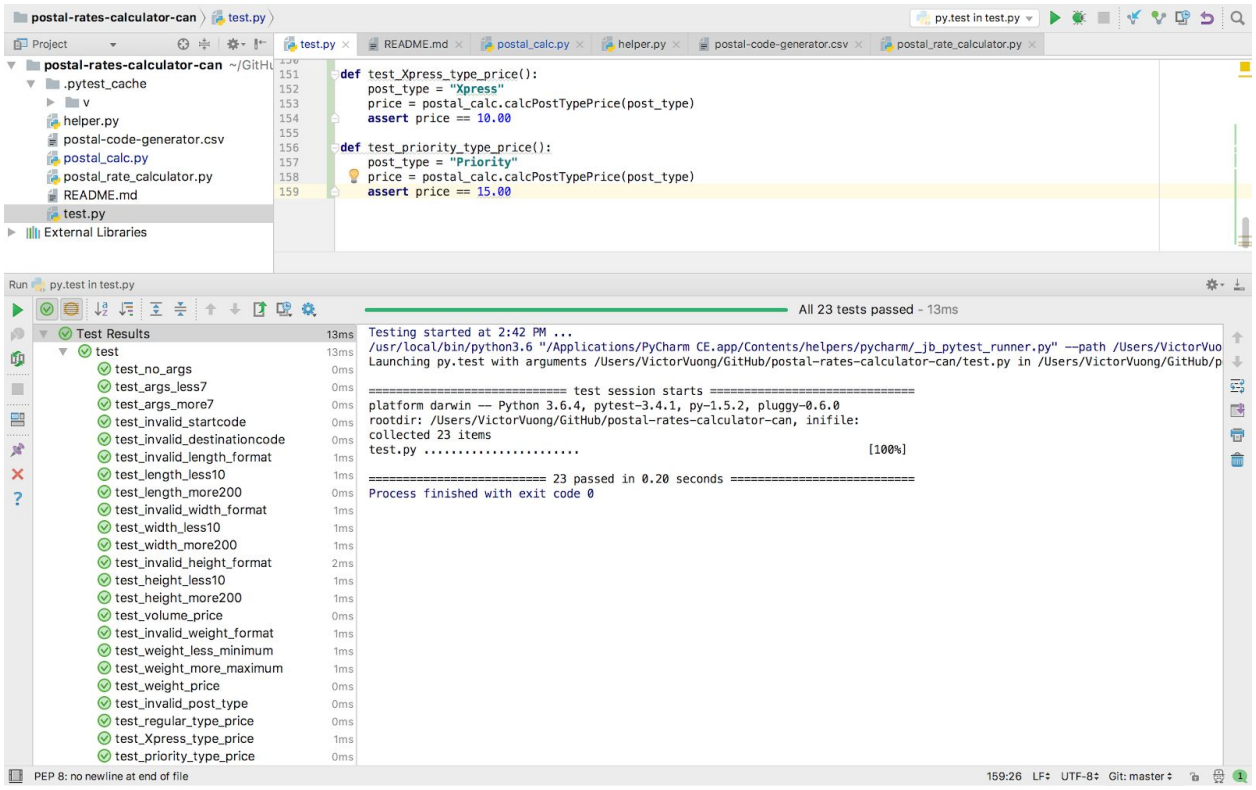
Test 23 Fail:



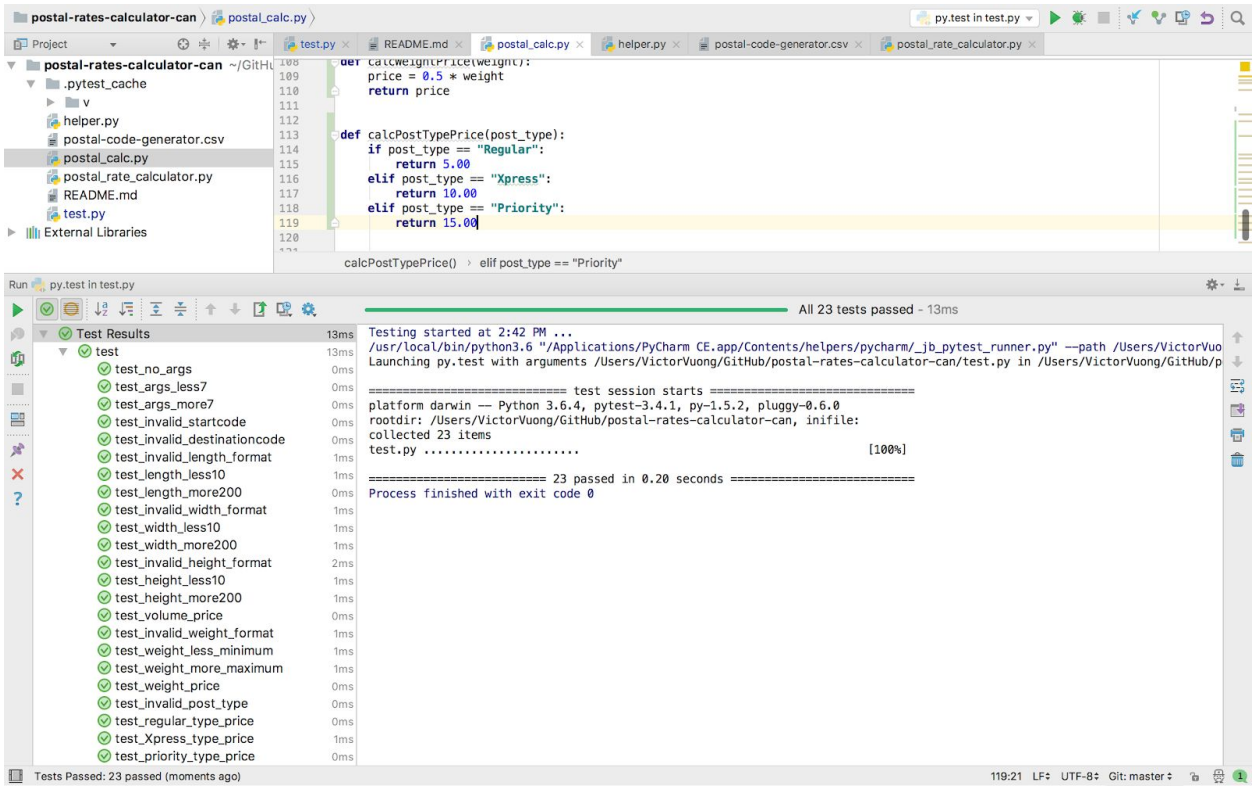
Test 23 Code:



Test 23 Succeed:



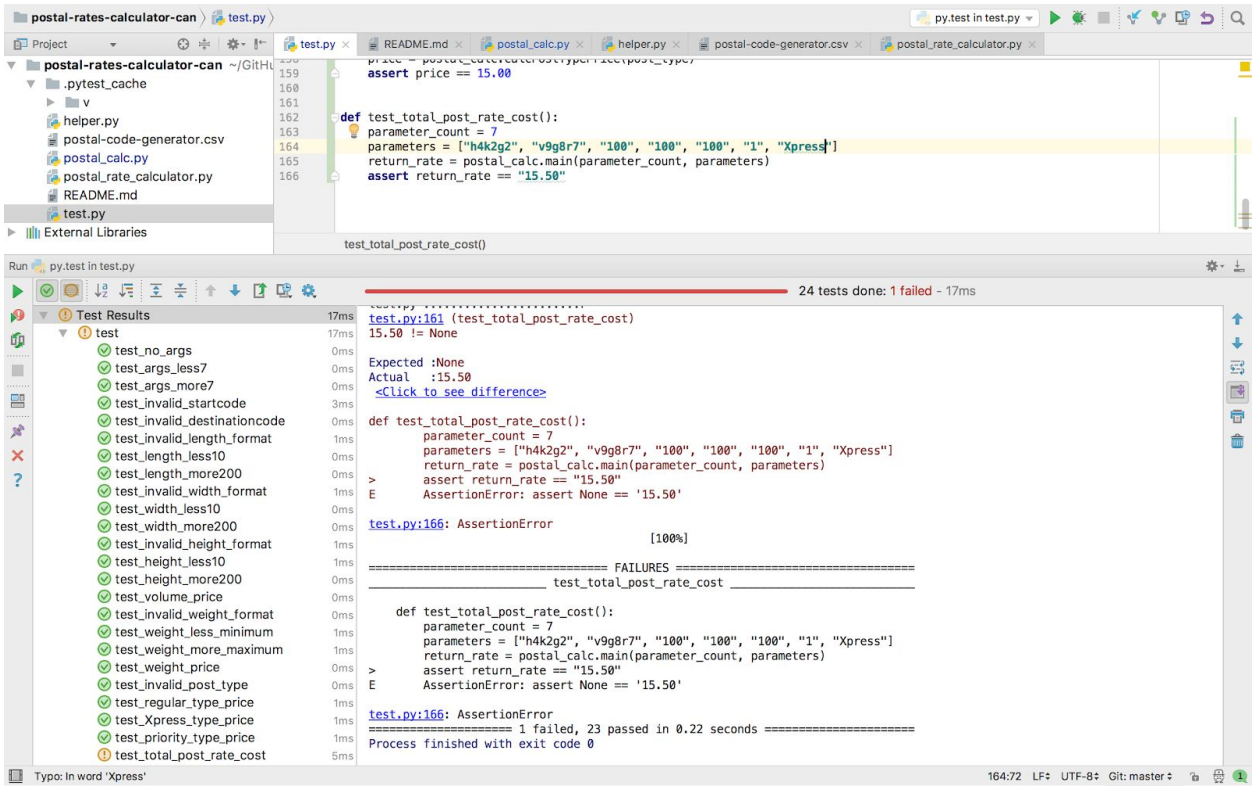
Test 23 Code with changes:



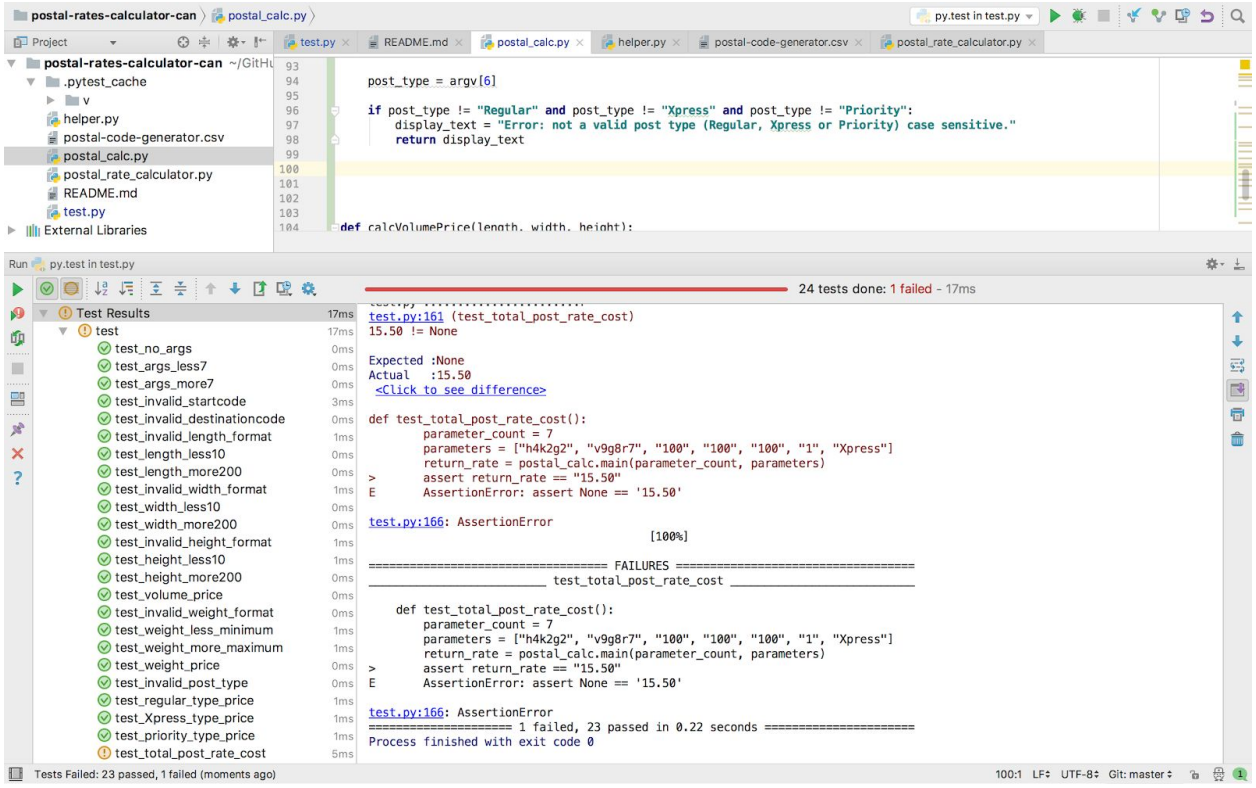
Test 24: Correct post rate cost from calculator (program)

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

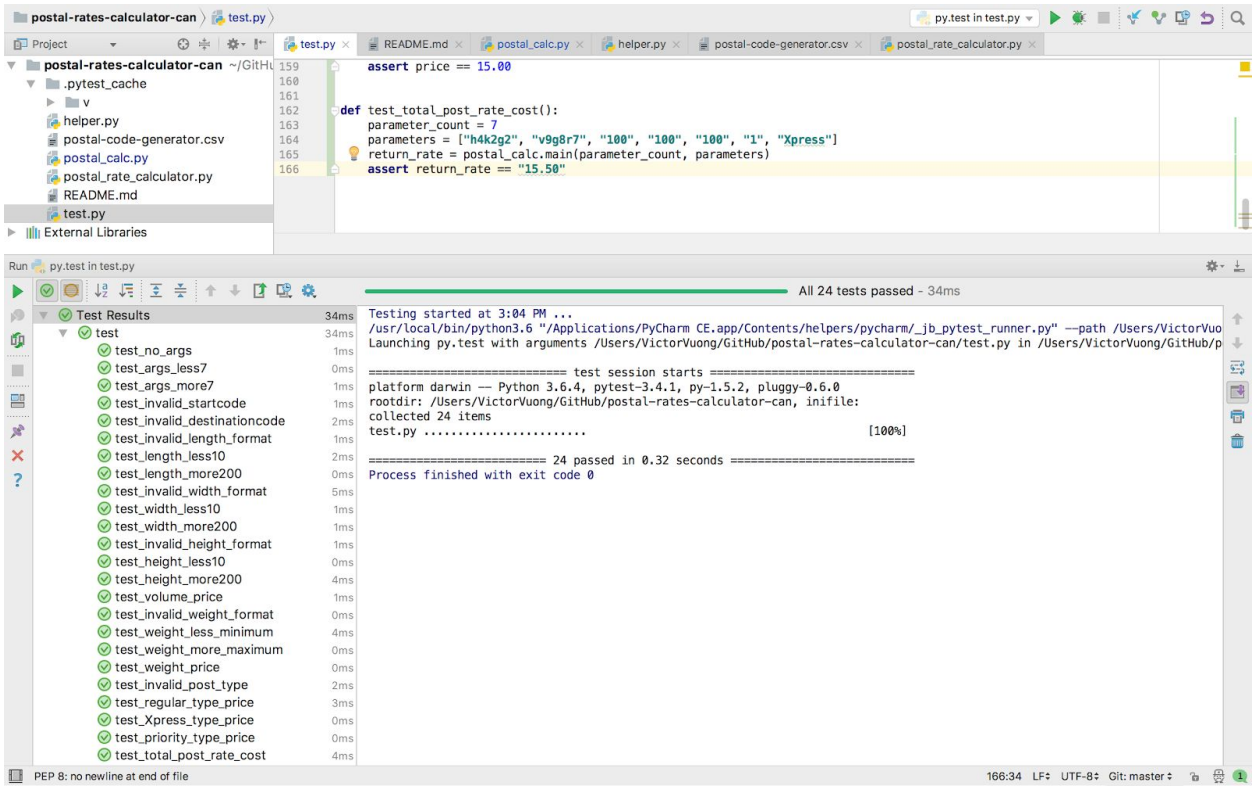
Test 24 Fail:



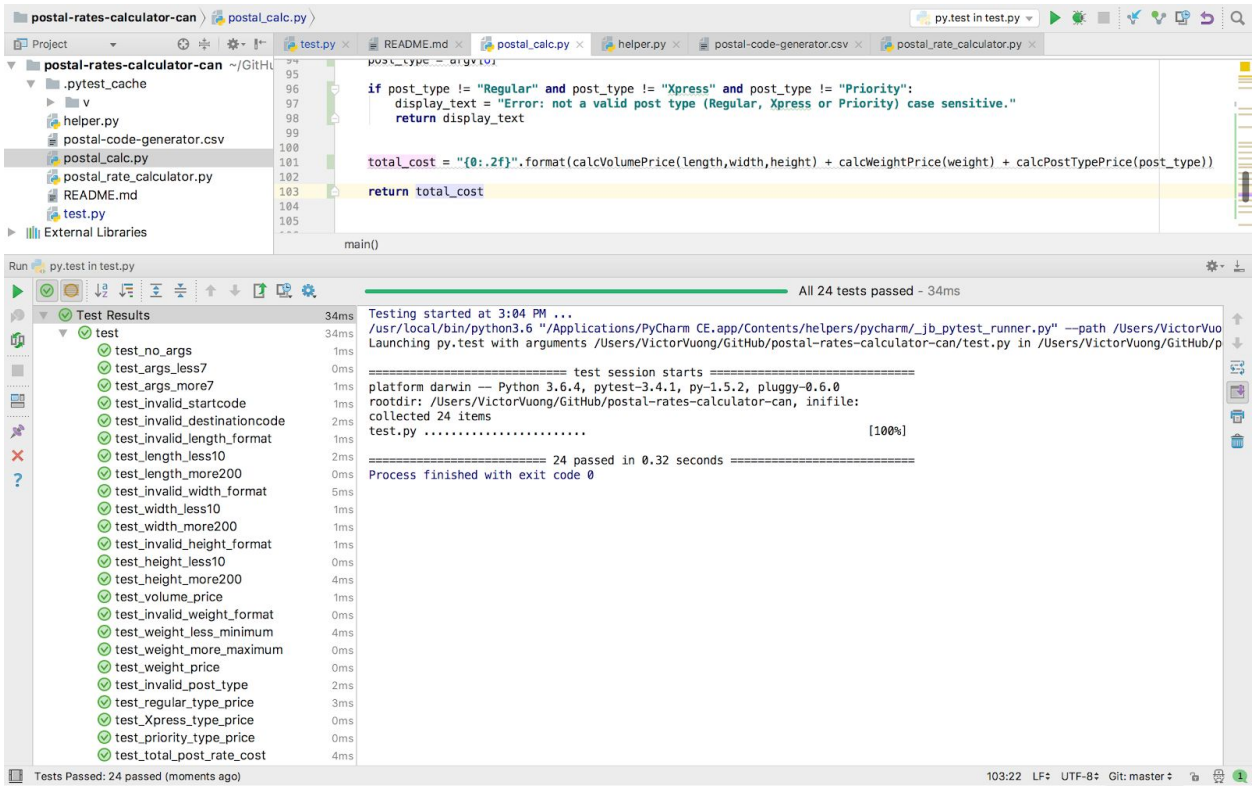
Test 24 Code:



Test 24 Succeed:



Test 24 Code with changes:



Source code

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

```
import sys
import csv

#main method that will run the post rate calculator. argc is the parameters count and argv contains the parameters
def main(argc, argv):

    #CHECK CORRECT NUMBER OF ARGUMENTS
    if (argc < 7 or argc > 7):
        display_text = "Usage: starting_code ending_code length width height weight post_type"
        return display_text     #terminate with error display string

    #CHECK VALID POSTAL CODES
    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

    #CHECK VALID LENGTH
    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
```

```
    #CHECK VALID WIDTH
    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

    #CHECK VALID HEIGHT
    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

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

    #CHECK VALID WEIGHT
    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

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

```
    #CHECK VALID POST TYPE
    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

    #CALCULATE TOTAL POST RATE
    total_cost = "{0:.2f}".format(calcVolumePrice(length,width,height) + calcWeightPrice(weight) + calcPostTypePrice(post_type))

    return total_cost


#function to calculate the volume price (assumed to be 5 dollars per meters cube)
def calcVolumePrice(length, width, height):
    price = (0.01 * length * 0.01 * width * 0.01 * height) * 5
    return price


#function to calculate the weight price (assumed to be 50 cents per kilogram)
def calcWeightPrice(weight):
    price = 0.5 * weight
    return price


#function to return the correct price of the post type (assumed to be 5 for regular, 10 for xpress, 15 for priority)
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
```

Source Code: Test

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

```python
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"]
    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."


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)."

```

```
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
```

```
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"
```