

Debugging With PyCharm

Disclaimer

- No right/wrong way to debug
- Ultimately a skill that takes practice
- Very difficult to debug a fundamental misunderstanding

Why PyCharm?

- Community edition is free to use
 - Professional edition is available for GaTech Students
 - Can download from [here](#)
- High quality debugging and code inspection tools
- Integrates nicely with testing/grading code

PyCharm Project Setup

- Make sure you're using the course Python environment



General Debugging Tips

- **Theoretical understanding is critical!**
- Use public threads to rubber-duck
- Use the project visualizations!
- Tests can be run individually
 - Start with the simplest case to debug (e.g. no noise) if you're failing multiple cases
- Use unit tests to verify basic functionality

Unit Tests?

- A unit test is just a function that verifies your code works
- The more testable your code is (and the more tests you have), the less debugging you have to do!

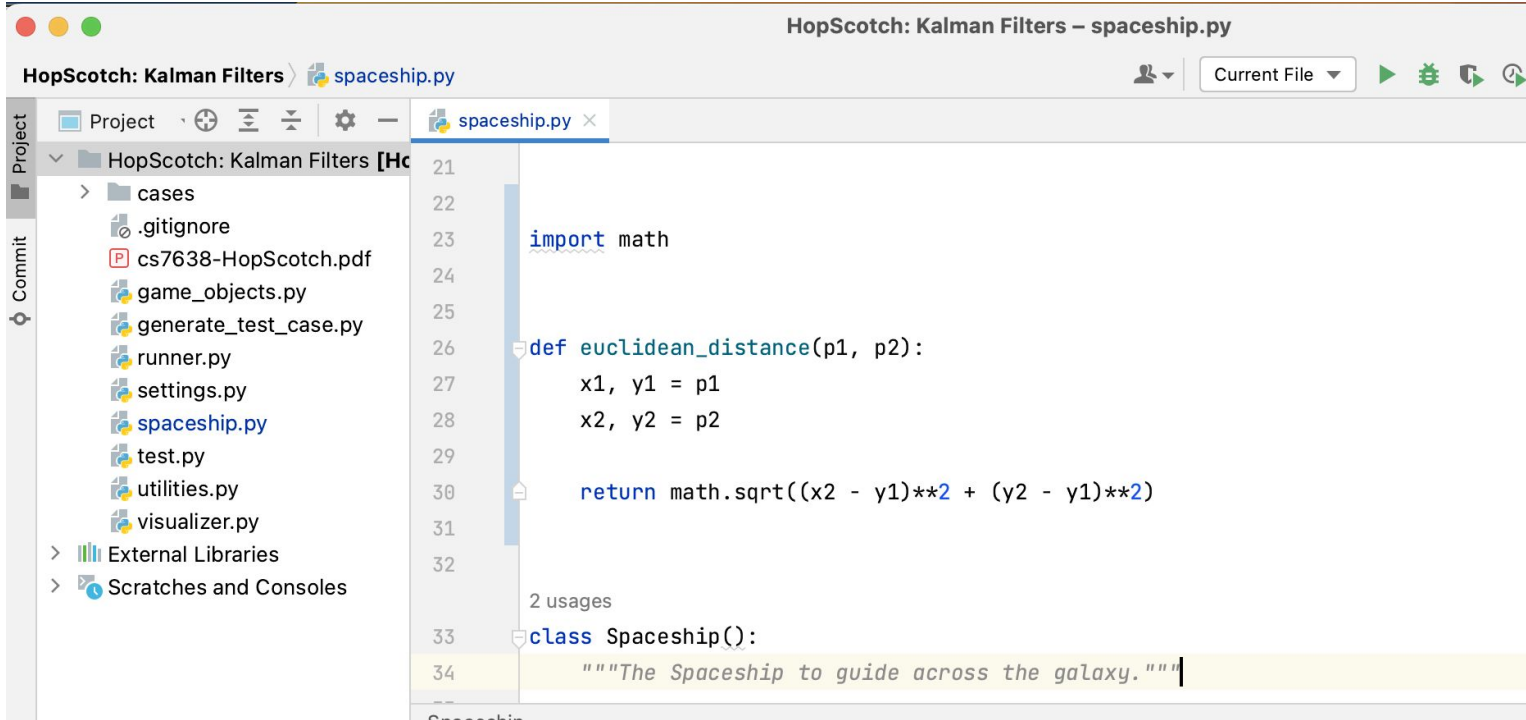
How to Add a Test

- General process
 - Write code, ideally in small snippets that can run in isolation
 - Write tests that exercise this code
 - Don't need to worry about this code working/not working **ever again**

Adding Code

- Say we need to write code that computes the Euclidean distance
- Better to write a separate function or embed this in a project function like **estimate_next_position?**

Sample Code



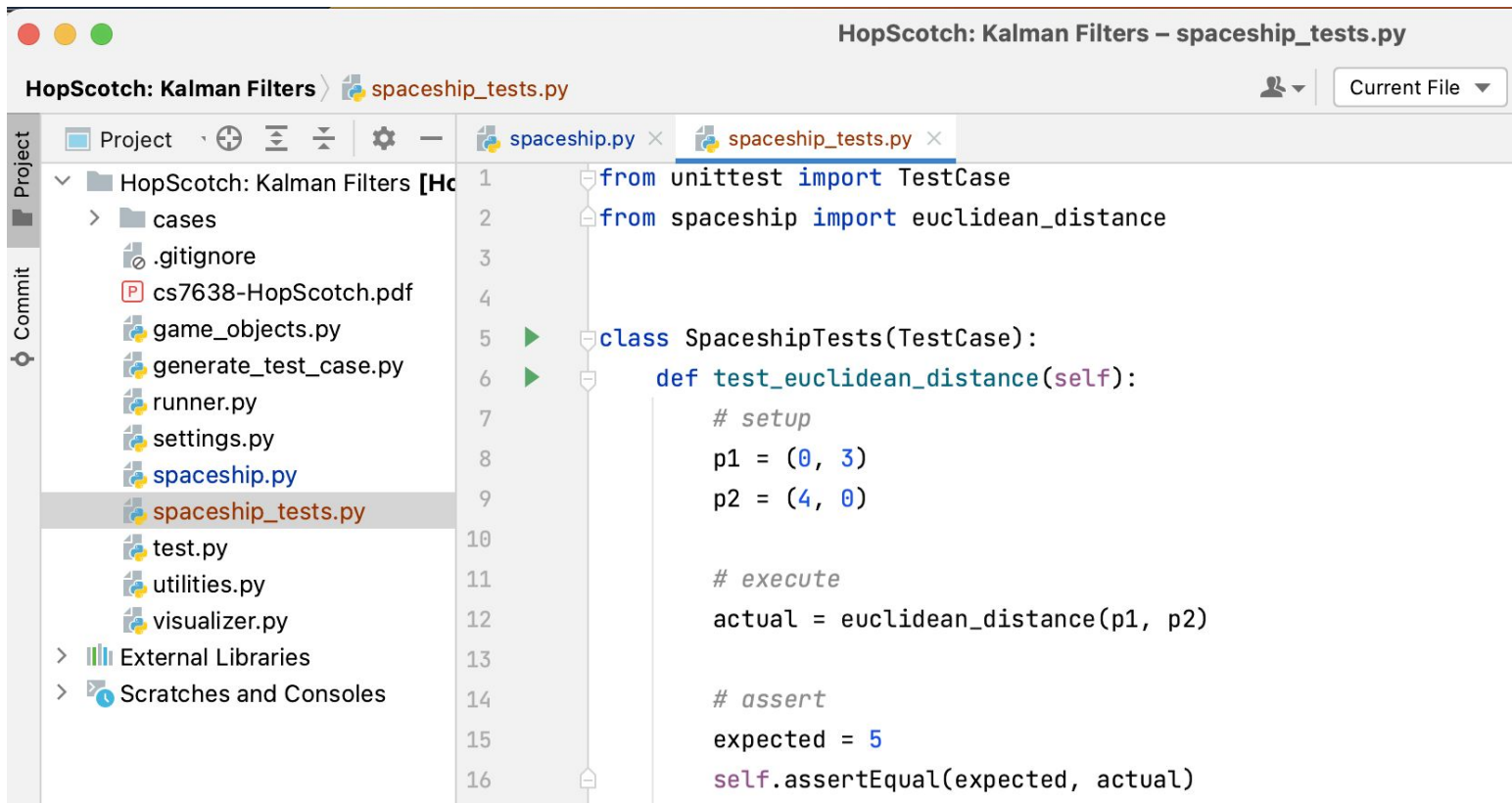
The screenshot shows an IDE window titled "HopScotch: Kalman Filters – spaceship.py". The left sidebar displays a project tree for "HopScotch: Kalman Filters" with files including .gitignore, cs7638-HopScotch.pdf, game_objects.py, generate_test_case.py, runner.py, settings.py, spaceship.py (selected), test.py, utilities.py, and visualizer.py. The main editor area shows the code for spaceship.py, which includes an import statement for math, a function definition for euclidean_distance, and the start of a class definition for Spaceship.

```
21
22
23 import math
24
25
26 def euclidean_distance(p1, p2):
27     x1, y1 = p1
28     x2, y2 = p2
29
30     return math.sqrt((x2 - y1)**2 + (y2 - y1)**2)
31
32
33 class Spaceship():
34     """The Spaceship to guide across the galaxy."""
```

Adding Tests

- Tests are usually written in separate files
 - Follow the naming convention `file_we_want_to_test_tests.py`
 - Convention is to either prefix/suffix with “tests”. Naming this way allows PyCharm to automatically find tests
- Every test has three parts
 - **Setup**: create the data your code will need
 - **Execute**: run your code with the setup data
 - **Assert**: verify your code’s return with the expected output

Sample Test



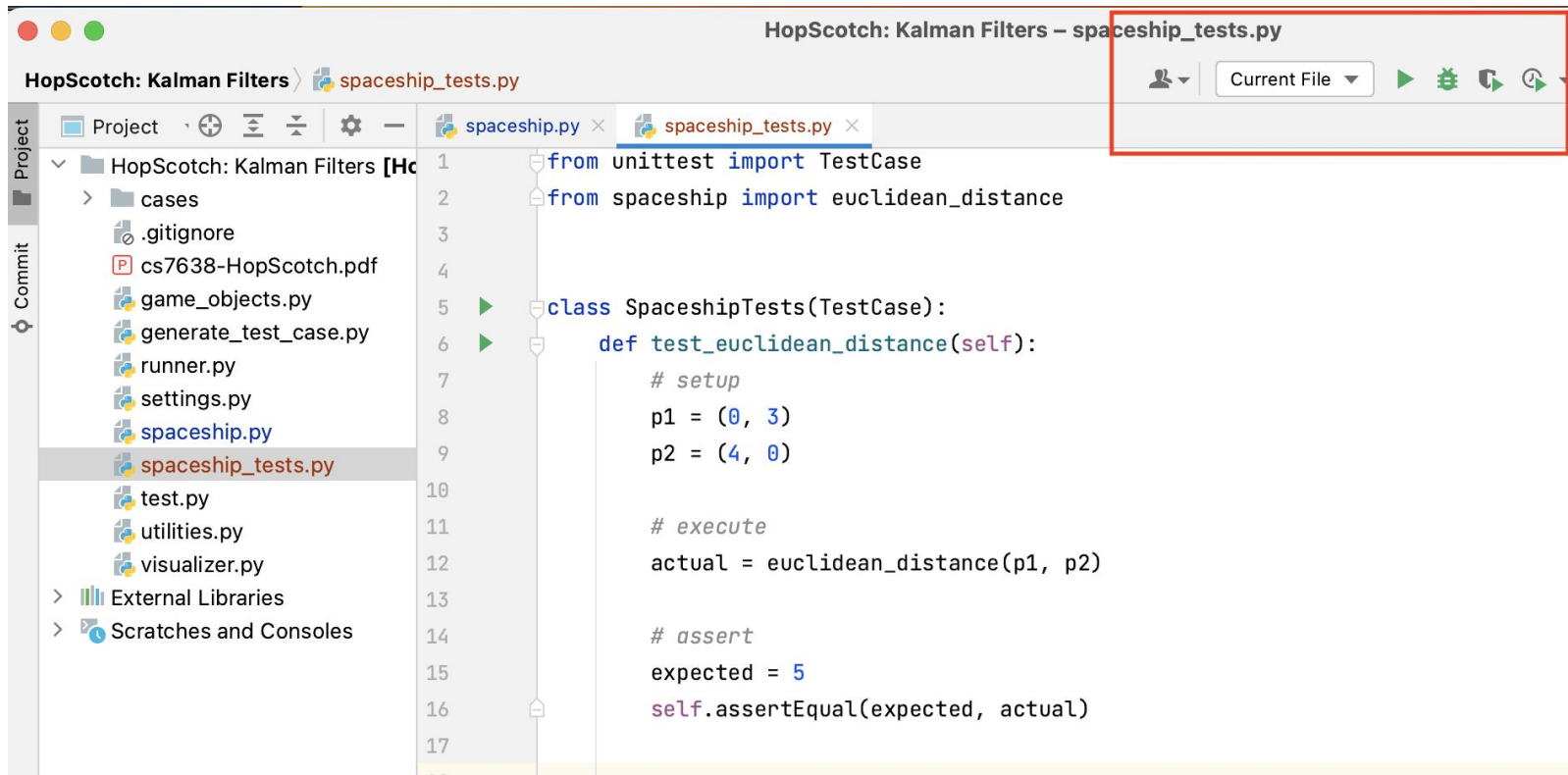
The screenshot shows an IDE window titled "HopScotch: Kalman Filters – spaceship_tests.py". The left sidebar contains a "Project" view with a tree structure: "HopScotch: Kalman Filters [Hc]" (expanded) contains "cases" (expanded), ".gitignore", "cs7638-HopScotch.pdf", "game_objects.py", "generate_test_case.py", "runner.py", "settings.py", "spaceship.py", "spaceship_tests.py" (selected), "test.py", "utilities.py", and "visualizer.py". Below this are "External Libraries" and "Scratches and Consoles". The main editor area shows the code for "spaceship_tests.py" with line numbers 1 through 16. The code imports "TestCase" from "unittest" and "euclidean_distance" from "spaceship". It defines a "SpaceshipTests" class inheriting from "TestCase" with a "test_euclidean_distance" method. The method sets up points p1=(0, 3) and p2=(4, 0), calculates the actual euclidean distance, and asserts it equals the expected value of 5.

```
1 from unittest import TestCase
2 from spaceship import euclidean_distance
3
4
5 class SpaceshipTests(TestCase):
6     def test_euclidean_distance(self):
7         # setup
8         p1 = (0, 3)
9         p2 = (4, 0)
10
11         # execute
12         actual = euclidean_distance(p1, p2)
13
14         # assert
15         expected = 5
16         self.assertEqual(expected, actual)
```

Running Tests

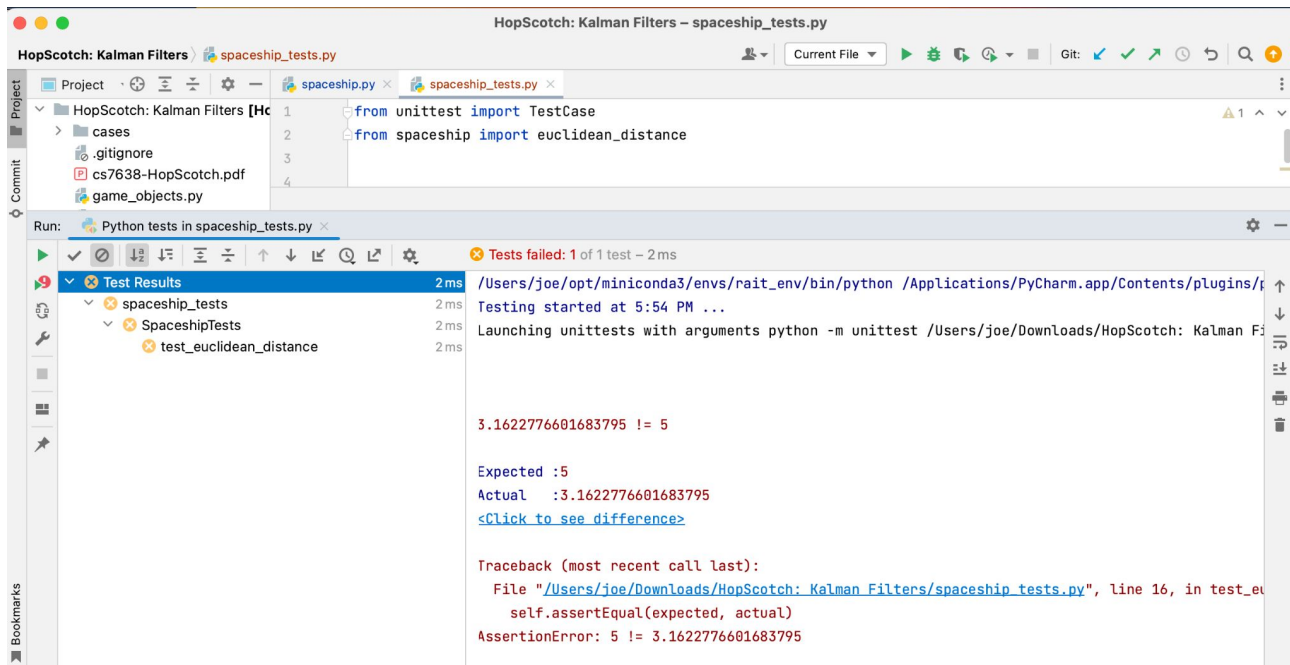
- PyCharm should automatically detect and provide methods to run your test code
- From the test file, you should see the option to create a run or debug configuration

Running Tests



Debugging

- Our first test failed - how can we figure out what went wrong?



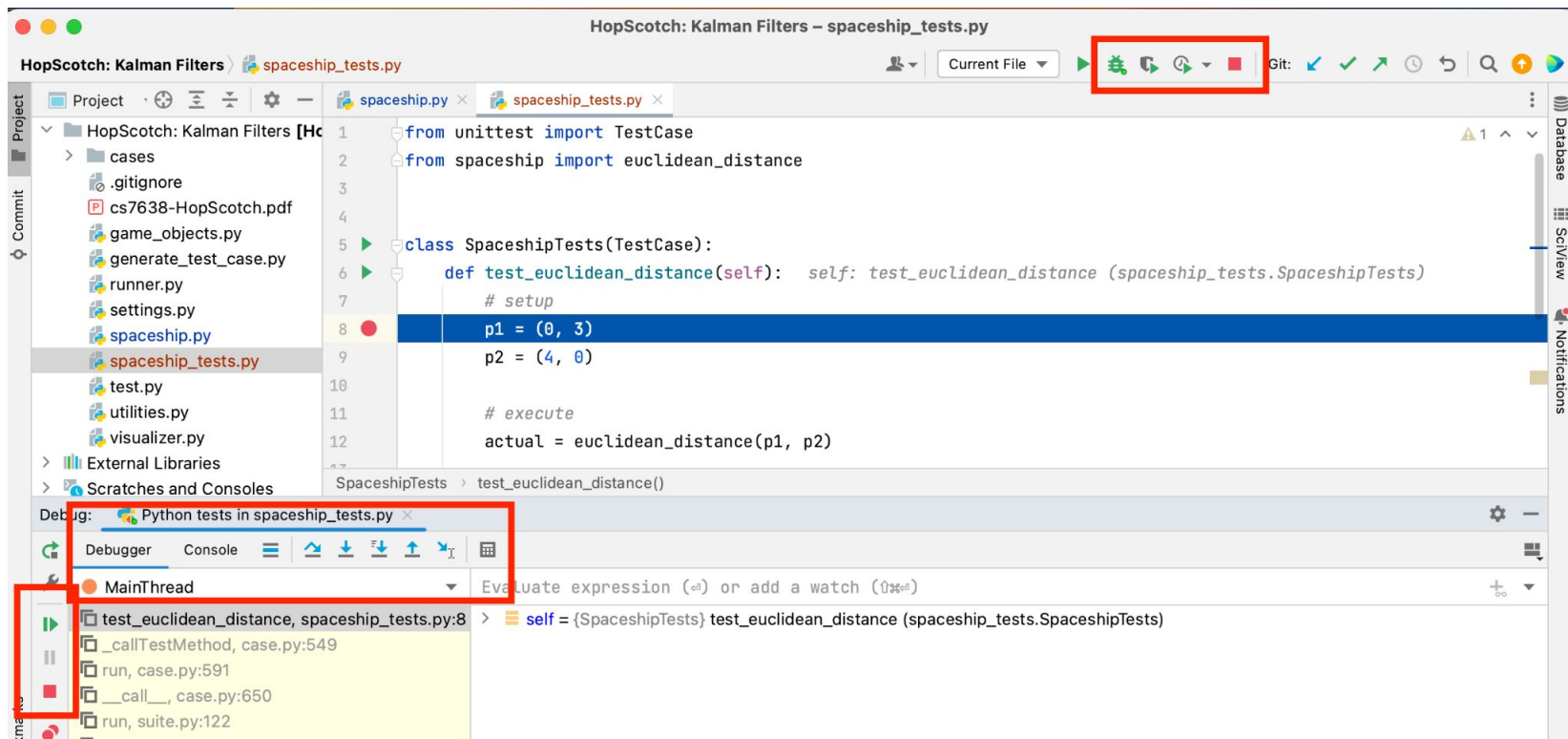
The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for running and debugging. The left sidebar shows the project structure with files like `spaceship.py` and `spaceship_tests.py`. The main editor displays the code for `spaceship_tests.py`, which imports `unittest.TestCase` and `euclidean_distance` from `spaceship`. Below the editor, the 'Run' panel shows the test results for `spaceship_tests.py`. The test `test_euclidean_distance` failed. The output pane displays the following error message:

```
Tests failed: 1 of 1 test - 2 ms
Test Results
  spaceship_tests
    SpaceshipTests
      test_euclidean_distance
        2 ms
        2 ms
        2 ms
        2 ms
        3.1622776601683795 != 5
        Expected :5
        Actual   :3.1622776601683795
        <Click to see difference>
        Traceback (most recent call last):
          File "/Users/joe/Downloads/HopScotch: Kalman Filters/spaceship_tests.py", line 16, in test_euclidean_distance
            self.assertEqual(expected, actual)
          AssertionError: 5 != 3.1622776601683795
```

Debugging

- Use the “Bug” icon in PyCharm to activate Debugging
- Use breakpoints to halt the execution
 - Without a breakpoint, the execution will continue to completion
- Use “Step Over” and “Step Into” to either advance over the current line
- Use the “Play” button to continue to the next breakpoint

Debugging Controls



Finding the Bug

The screenshot shows an IDE window titled "HopScotch: Kalman Filters – spaceship.py". The left sidebar displays a project tree for "HopScotch: Kalman Filters [Hc]". The main editor shows the file "spaceship.py" with the following code:

```
25  
26  
27 def euclidean_distance(p1, p2): p1: (0, 3) p2: (4, 0)  
28     x1, y1 = p1 x1: 0 y1: 3  
29     x2, y2 = p2 x2: 4 y2: 0  
30     return math.sqrt((x2 - y1)**2 + (y2 - y1)**2)  
31  
32
```

A red dot is visible on line 27, and a lightbulb icon is on line 29. The return statement on line 30 is highlighted in blue. The right sidebar shows "Database", "SciView", and "Notifications".

The bottom panel shows the "Debug" window for "Python tests in spaceship_tests.py". The "Debugger" tab is active, showing the "MainThread" and a list of stack frames. The "Evaluate expression (ed) or add a watch (f)" window is open, displaying the following variable values:

```
> p1 = {tuple: 2} (0, 3)  
> p2 = {tuple: 2} (4, 0)  
01 x1 = {int} 0  
01 x2 = {int} 4  
01 y1 = {int} 3  
01 y2 = {int} 0
```

The "Evaluate expression" window is highlighted with a red border.

Finding the Bug

The screenshot shows an IDE window titled "HopScotch: Kalman Filters - spaceship.py". The left sidebar displays a project tree with files like `spaceship.py` and `spaceship_tests.py`. The main editor shows the `euclidean_distance` function in `spaceship.py`, which calculates the distance between two points `p1` and `p2`. The function is called in `spaceship_tests.py`. The debugger is open, showing the `MainThread` and the `euclidean_distance` function. A red box highlights the `Evaluate` button in the debugger. Another red box highlights the `Evaluate` dialog box, which shows the code fragment `math.sqrt((x2 - y1)**2 + (y2 - y1)**2)` and the result "Nothing to show".

spaceship.py

```
25 def euclidean_distance(p1, p2):
26     x1, y1 = p1
27     x2, y2 = p2
28     return math.sqrt((x2 - y1)**2 + (y2 - y1)**2)
```

spaceship_tests.py

```
27 def test_euclidean_distance():
28     p1 = (0, 0)
29     p2 = (4, 0)
30     result = euclidean_distance(p1, p2)
31     assert result == 4
```

Debugger

Python tests in spaceship_tests.py

MainThread

euclidean_distance, spaceship.py:30

test_euclidean_distance, spaceship_tests.py:12

`Evaluate` button highlighted.

Evaluate Dialog

Code fragment:

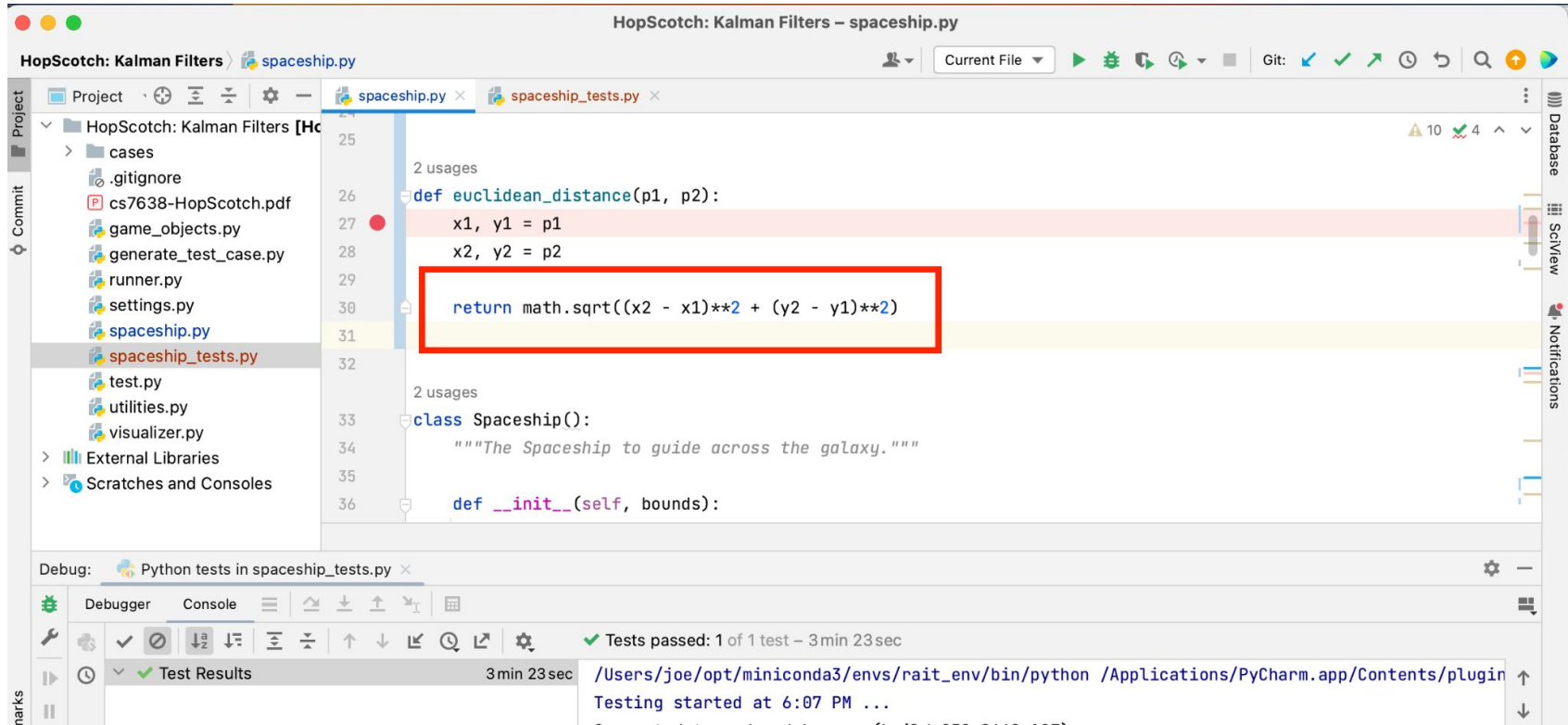
```
math.sqrt((x2 - y1)**2 + (y2 - y1)**2)
```

Result:

Nothing to show

`Evaluate` button highlighted.

Fixing the Bug



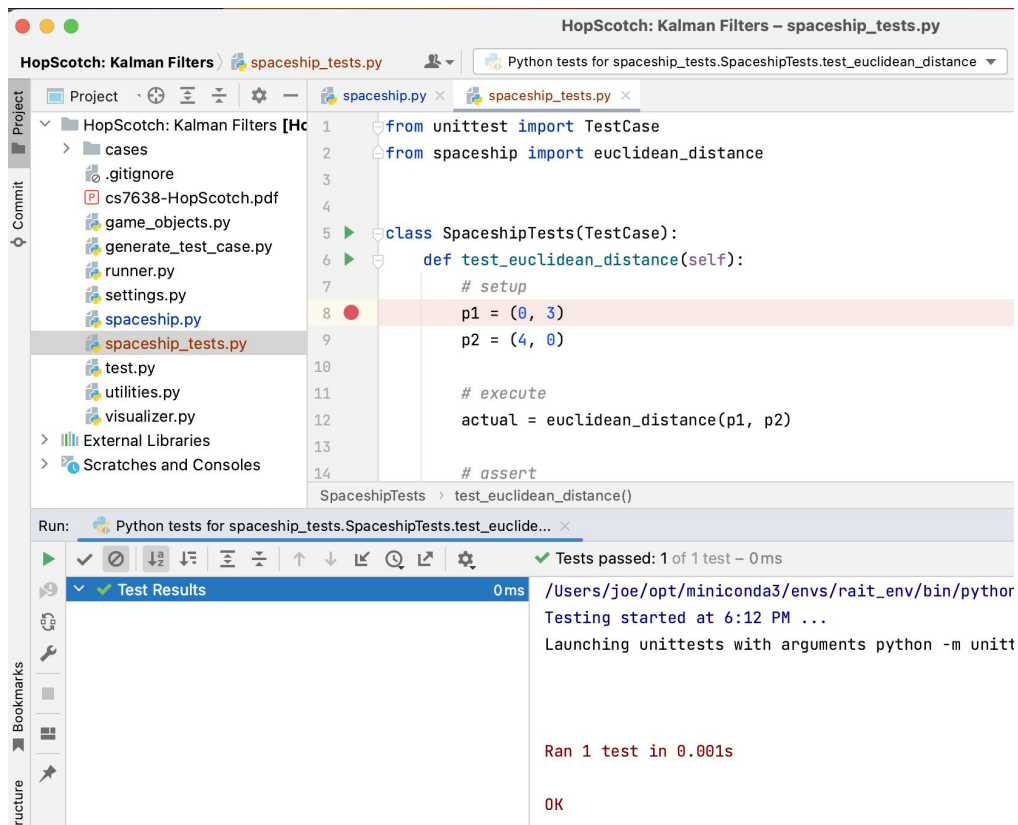
The screenshot shows the PyCharm IDE with the file `spaceship.py` open. The `euclidean_distance` function is defined as follows:

```
def euclidean_distance(p1, p2):  
    x1, y1 = p1  
    x2, y2 = p2  
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
```

The return statement is highlighted with a red box, indicating the location of the bug. The bug is a `TypeError: unsupported operand type(s) for ** or pow(): 'int' and 'str' not supported`, which occurs because the code uses `**2` instead of `**2.0` for floating-point arithmetic.

The bottom panel shows the test results for `spaceship_tests.py`, indicating that 1 of 1 tests passed in 3 minutes 23 seconds. The console output shows the test environment path and the start time of the testing process.

Verifying with Tests



The screenshot shows an IDE window titled "HopScotch: Kalman Filters – spaceship_tests.py". The left sidebar displays a project tree with files like `spaceship.py` and `spaceship_tests.py`. The main editor shows the following Python code:

```
1 from unittest import TestCase
2 from spaceship import euclidean_distance
3
4
5 class SpaceshipTests(TestCase):
6     def test_euclidean_distance(self):
7         # setup
8         p1 = (0, 3)
9         p2 = (4, 0)
10
11         # execute
12         actual = euclidean_distance(p1, p2)
13
14         # assert
```

Below the code editor, the "Run" panel shows the command: `Python tests for spaceship_tests.SpaceshipTests.test_euclidean...`. The "Test Results" panel indicates "Tests passed: 1 of 1 test – 0ms". The output console shows the following text:

```
Testing started at 6:12 PM ...
Launching unittests with arguments python -m unittest

Ran 1 test in 0.001s

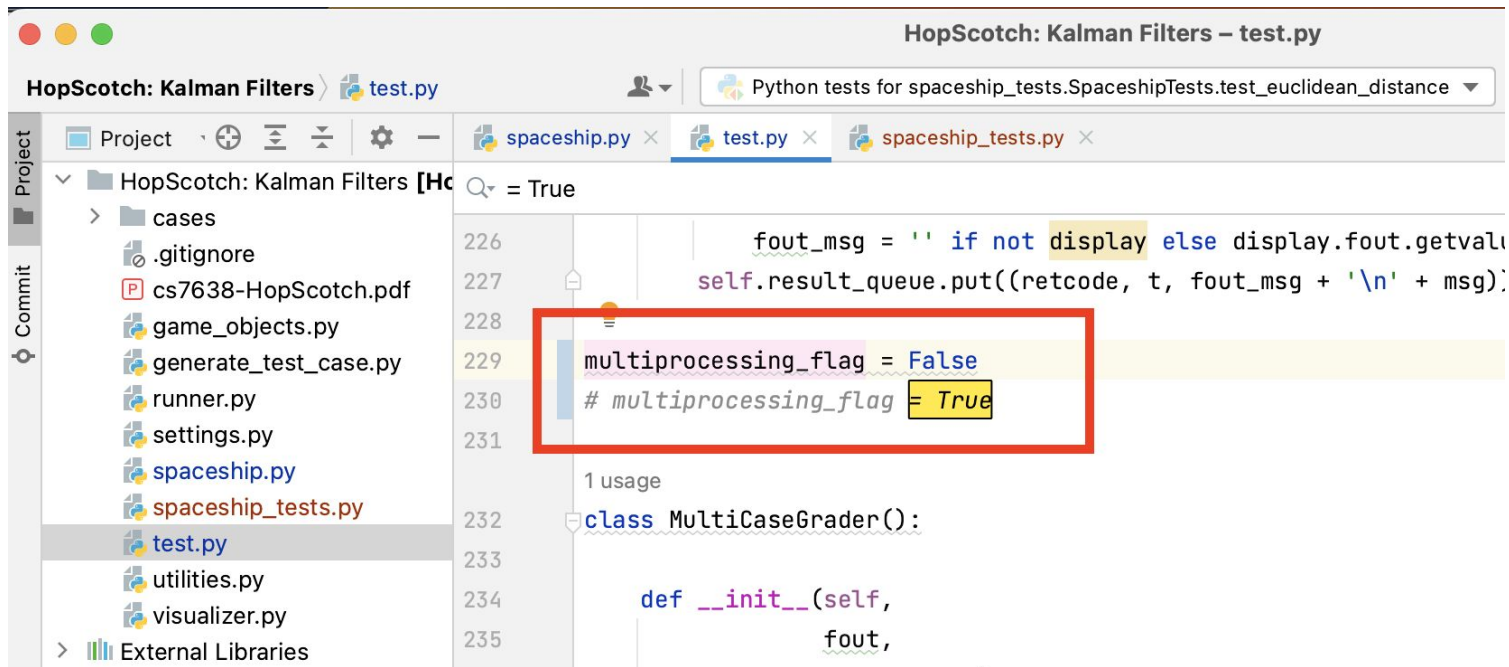
OK
```

Key Things to Remember

- Small functions are easier to test/debug
- Unit tests allow you to keep your code easy to verify
- Debugging can still be difficult

Debugging Project Tests

- Prefer to debug a single test case rather than the whole suite
- Disable multiprocessing, if applicable



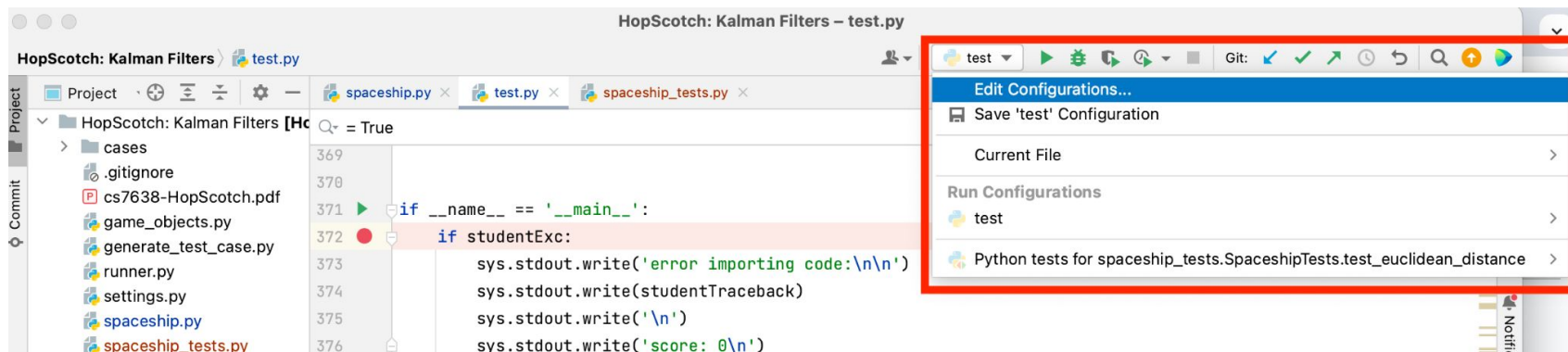
The screenshot shows an IDE window titled "HopScotch: Kalman Filters – test.py". The left sidebar displays a project tree for "HopScotch: Kalman Filters" with files like .gitignore, cs7638-HopScotch.pdf, game_objects.py, generate_test_case.py, runner.py, settings.py, spaceship.py, spaceship_tests.py, test.py, utilities.py, and visualizer.py. The main editor area shows the code for test.py. A red box highlights the following code snippet:

```
229 multiprocessing_flag = False
230 # multiprocessing_flag = True
```

Below this snippet, a tooltip indicates "1 usage" and points to the `class MultiCaseGrader():` definition. The code continues with a `def __init__(self, fout,` line.

Debugging Project Tests

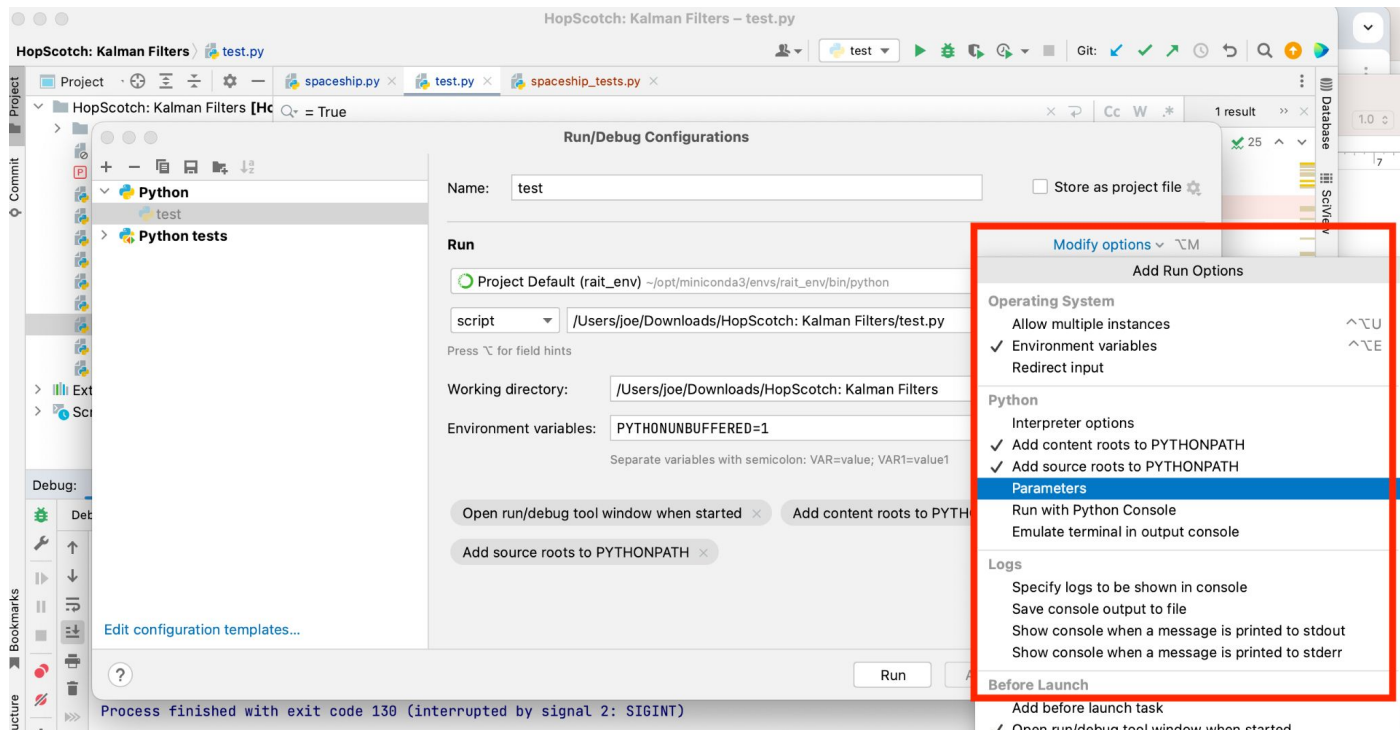
- Project entrypoint run from command line
- Have to pass these arguments to the debugger



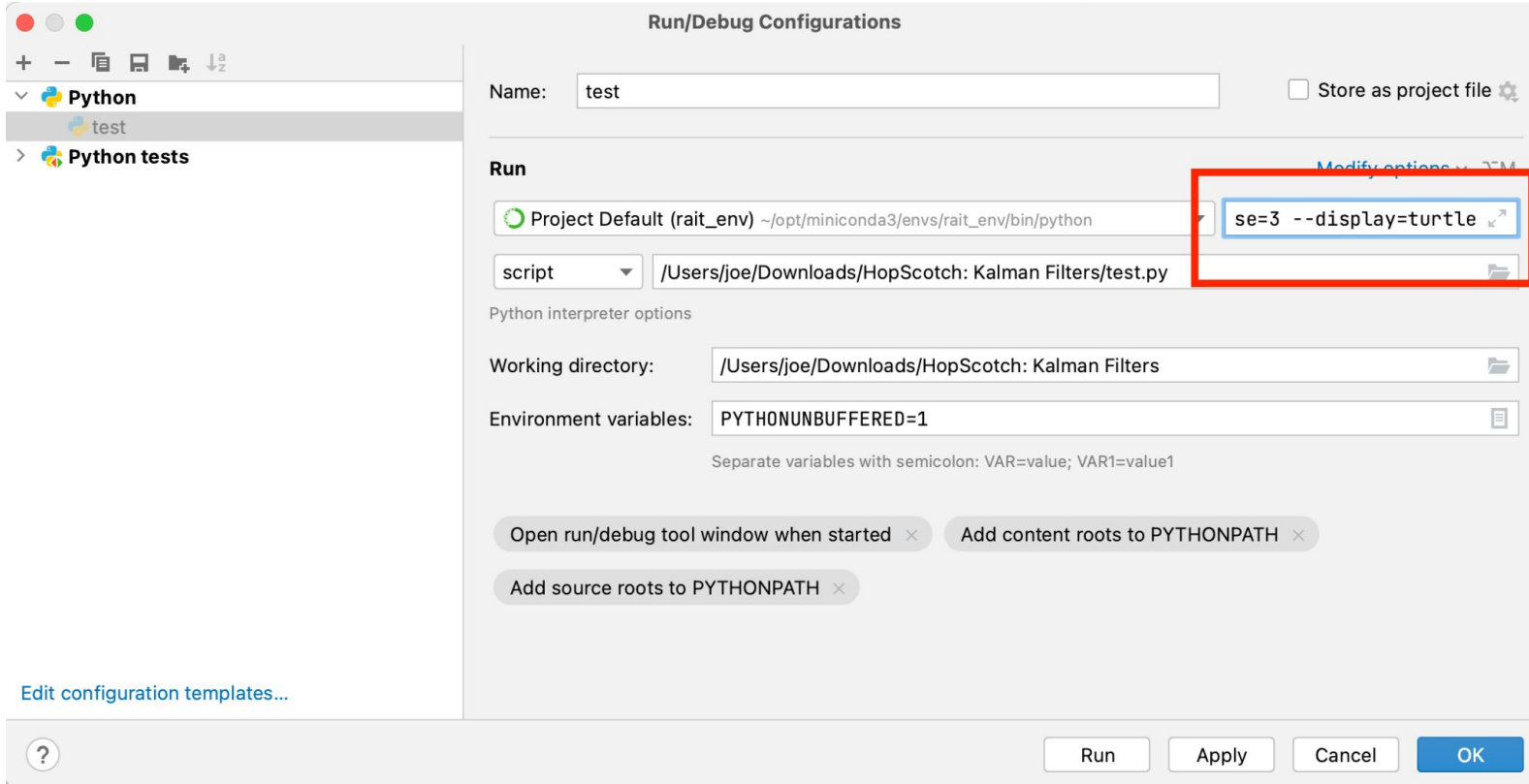
Debugging Project Tests

- Look at the entrypoint to the tests
- Figure out what options are parsed from the command line
 - E.g. KF **--method=estimate --case=3 --display=turtle**

Debugging Project Tests



Debugging Project Tests



Debugging Project Tests

- Set a breakpoint and verify that arguments set for the run configuration are passed/parsed as expected

Debugging Project Tests

The screenshot shows an IDE window titled "HopScotch: Kalman Filters – test.py". The left sidebar displays a project tree with files like `cases`, `.gitignore`, `cs7638-HopScotch.pdf`, `game_objects.py`, `generate_test_case.py`, `runner.py`, `settings.py`, `spaceship.py`, `spaceship_tests.py`, `test.py`, `utilities.py`, and `visualizer.py`. The main editor shows the `test.py` file with a red breakpoint at line 86. A tooltip for the `args` variable is visible, showing its value as `Namespace(method='estimate', case='3', display='turtle')`. The bottom panel shows the debugger with the `MainThread` selected, and the `Evaluate expression` window displaying the current state of the program, including `ESTIMATE_TIMEOUT`, `FAILURE_EXCEPTION`, `FAILURE_TIMEOUT`, `JUMP_TIMEOUT`, `TASKS`, and `args`.

```
382 mcg = MultiCaseGrader(sys.stdout)
383 mcg.run()
384
385 args = parser().parse_args() args: Namespace(method='estimate', case='3', display='turtle')
386 if args: args = (Namespace() Namespace(method='estimate', case='3', display='turtle'))
387     case = {str} '3'
388     display = {str} 'turtle'
389     method = {str} 'estimate'
390     > Protected Attributes
391     Set value F2
392     args.display = 'text'
393
394 main(method_name=args.method,
if __name__ == '__main__': else: if student_id: try: else
```

Debug: test

Debugger Console

MainThread

<module>, test.py:386

Evaluate expression (⌘) or add a watch (⇧⌘)

```
01 ESTIMATE_TIMEOUT = {int} 200
01 FAILURE_EXCEPTION = {str} 'exception_raised'
01 FAILURE_TIMEOUT = {str} 'execution_time_exceeded'
01 JUMP_TIMEOUT = {int} 50
> TASKS = {tuple: 35} (GradingTask(case_num=1, method_name='estimate', weight=1, timeout=200), GradingTask(case_num=... View
> args = {Namespace} Namespace(method='estimate', case='3', display='turtle')
```

Troubleshooting Project Debugging

- Delete the configuration, start over
- Can also modify our code
 - Comment out the command line parsing
 - Run our tests as Python unit tests