

# Making tests human readable with Generative AI & very little code

Pete Houghton

# What's your problem?

"Normal" people (non techies) keep asking me what my tests do, and that keeps changing

Cucumber is great for creating human readable tests but it does add an overhead

Sometimes I have to work with new tests on a new product – it can take a while to figure out what's going on





```

5 @pytest.mark.parametrize("hand, players, expected_count, expected_score, description", [
6     ((1, "S"), (2, "S")), ["Abi", "Bob"], 3, 0, "Simple ace and 2."),
7     ((4, "S")), ["Abi", "Bob"], 4, 0, "simple 1 card, 4"),
8     ((1, "H"), (10, "S")), ["Abi", "Bob"], 11, 0, "Simple ace and 10."),
9     ((5, "H"), (10, "S")), ["Abi", "Bob"], 15, 2, "fifteen for 2")
10 ])
11 def test_simple_hands1(hand, players, expected_count, expected_score, description):
12
13     calc_count, calc_score, calc_desc = cribbage_scorer.play_score_ongoing(hand)
14     print(calc_count, calc_score, calc_desc)
15     assert calc_score == expected_score, \
16         f"The calculated score was: {calc_score}, the expected score: {expected_score}. " + \
17         f"The calculated count was: {calc_count} and the expected : {expected_count} " + \
18         f"The hand description was: {description}, players: {players} "
19
20
21 @pytest.mark.parametrize("hand, players, last_card, expected_count, expected_score, description", [
22     ((13, "S"), (12, "H"), (10, "S")), ["Abi", "Bob"], True, 30, 1, "Last card at 30"),
23     ((13, "S"), (12, "H"), (9, "S")), ["Abi", "Bob"], True, 30, 1, "Last card at 29"),
24     ((13, "S"), (12, "H"), (9, "S")), ["Abi", "Bob"], False, 30, 0, "Not last card at 29"),
25     ((13, "S"), (12, "H"), (10, "S"), (1, "D")), ["Abi", "Bob"], True, 31, 2, "Last card at 31")
26 ])
27 def test_last_card(hand, players, last_card, expected_count, expected_score, description):
28
29     calc_count, calc_score, calc_desc = cribbage_scorer.play_score_ongoing(hand, last_card)
30     print(calc_count, calc_score, calc_desc)
31     assert calc_score == expected_score, \
32         f"The calculated score was: {calc_score}, the expected score: {expected_score}. " + \
33         f"The calculated count was: {calc_count} and the expected : {expected_count}. " + \
34         f"The hand description was: {description}"
35
36
37 @pytest.mark.parametrize("played_cards, players, expected_last_player", [
38     ([], ["Abi", "Bob"], None),
39     ((1, "S"), (2, "S")), ["Abi", "Bob"], "Bob"),
40     ((4, "S")), ["Abi", "Bob"], "Abi"),
41     ((1, "H"), (10, "S")), ["Abi", "Bob"], "Bob"),
42     ((1, "H"), (10, "S"), (10, "D")), ["Abi", "Bob"], "Abi"),
43     ((5, "H"), (10, "S"), (5, "H")), ["Abi", "Bob"], "Abi"),
44     ((1, "H"), (10, "S")), ["Abi", "Bob", "Charles"], "Bob"),
45     ((5, "H"), (10, "S"), (5, "H")), ["Abi", "Bob", "Charles", "David"], "Charles"),
46     ((5, "H"), (5, "S"), (5, "H"), (5, "C"), (1, "S"), (1, "H")), ["Abi", "Bob", "Charles", "David"], "Bob")
47 ])

```

# The test code

- Parameterized tests can be hard to read
- The test isn't always intuitive
- Code comments have a short half life

# Code to English

- With one API call to OpenAI we can get a human readable explanation of the tests
- No application details or context were provided – just the test code

The code contains a series of tests for a Cribbage scoring system. The tests are organized into different categories, testing various aspects of the scoring system. Below is a summary of the tests and what they test:

1. **test\_duplicate\_cards\_play\_set** and **test\_duplicate\_cards\_play\_ongoing**: These tests check if the scoring system raises a `ValueError` when there are duplicate cards in the played cards set. The tests are parameterized with different sets of played cards and the expected duplicate cards.
2. **test\_simple\_hands1**: This test checks the scoring system for simple hands with different card combinations and expected scores. The test is parameterized with different hands, players, expected counts, expected scores, and descriptions.
3. **test\_last\_card**: This test checks the scoring system for the last card played in a hand. The test is parameterized with different hands, players, last card status, expected counts, expected scores, and descriptions.
4. **test\_last\_player**: This test checks the scoring system for determining the last player in a game. The test is parameterized with different played cards, players, and expected last players.
5. **test\_runs**: This test checks the scoring system for runs (sequences of consecutive cards) in a hand. The test is parameterized with different hands, players, expected counts, expected scores, and descriptions.
6. **test\_multiples**: This test checks the scoring system for multiples (pairs, triples, and quadruples) in a hand. The test is parameterized with different hands, players, expected counts, expected scores, and descriptions.
7. **test\_whole\_play\_scoring**: This test checks the scoring system for the whole play, including scoring for each player. The test is parameterized with different played cards, players, expected counts, expected scores, and descriptions.
8. **test\_whole\_play\_scoring\_with\_go**: This test checks the scoring system for the whole play, including scoring for each player and handling "go" situations. The test is parameterized with different played cards, players, expected counts, expected scores, and descriptions.
9. **test\_cut\_score**: This test checks the scoring system for the cut card and dealer scoring. The test is parameterized with different cut cards, players, dealers, expected scores, and descriptions.

# That's Already in Copilot (Beta)

Copilot

Explain This

# That's *nice* but the results are like hiero... ancient Egyptian writing

```
python -m pytest --tb=no
===== test session starts =====
platform linux -- Python 3.10.4, pytest-7.3.1, pluggy-1.0.0
rootdir: /workspaces/cribbage_scorer
plugins: anyio-3.6.2
collected 1096 items

tests/play/play_scorer_exceptions_test.py ..... [ 0%]
tests/play/play_scorer_test.py ..... [ 4%]
tests/show/show_scorer_impossible_score_test.py ..... [ 9%]
..... [ 18%]
..... [ 28%]
..... [ 37%]
..... [ 47%]
..... [ 56%]
..... [ 66%]
..... [ 75%]
..... [ 85%]
..... [ 94%]
..... [ 95%]
tests/show/show_scorer_exceptions_test.py ... [ 95%]
tests/show/show_scorer_test.py .....FF...F.....F.... [100%]

===== short test summary info =====
FAILED tests/show/show_scorer_test.py::test_his_nobs[starter1-hand1-False-1-One for his nobs (1pt)] - AssertionError: The calculated score was: (2, 'One for his nobs (1pt)'), the expected score: 1.
FAILED tests/show/show_scorer_test.py::test_etc_hands_crib_neutral[starter0-hand0-29-Perfect hand, 29] - AssertionError: The calc score was: (30, "Four of a kind: 5s (12pts)|Made 15 from [(5, 'H')],
FAILED tests/show/show_scorer_test.py::test_etc_hands_crib_neutral[starter4-hand4-23-4th best hand, 23] - AssertionError: The calc score was: (24, "Three of a kind: 5s (6pts)|Double Jacks (2pts)|Ma
FAILED tests/show/show_scorer_test.py::test_etc_hands_crib_neutral[starter13-hand13-18-18, highest with 5pt flush] - AssertionError: The calc score was: (19, "Made 15 from [(5, 'H'), (10, 'H')] (2p
===== 4 failed, 1092 passed in 1.61s =====
```



# Test results Summary

A total of 1096 tests were executed, out of which 1092 tests passed and 4 tests failed.

## Executive Summary of the test results

The majority of the tests executed successfully, with a high pass rate. However, there were 4 test failures further investigation and resolution.

## Detailed Test Results for all tests executed

1. `play_scorer_exceptions_test.py`: This test checks for exceptions in the play scorer.
  - 0 tests failed
2. `play_scorer_test.py`: This test checks the functionality of the play scorer.
  - 0 tests failed
3. `show_scorer__impossible_score_test.py`: This test checks for impossible scores in the show scorer.
  - 0 tests failed
4. `show_scorer_exceptions_test.py`: This test checks for exceptions in the show scorer.
  - 0 tests failed
5. `show_scorer_test.py`: This test checks the functionality of the show scorer.
  - 4 tests failed

## The details of the tests that failed in plain English

1. `test_his_nobs`: The calculated score was 2, while the expected score was 1.
2. `test_etc_hands__crib_neutral` (Perfect hand, 29): The calculated score was 30, while the expected score was 29.
3. `test_etc_hands__crib_neutral` (4th best hand, 23): The calculated score was 24, while the expected score was 23.
4. `test_etc_hands__crib_neutral` (18, highest with 5pt flush): The calculated score was 19, while the expected score was 18.

# Results to English

- With one OpenAI API call the results are now human readable.
- The context did not even need the test code
- We could provide more useful info if we send the test code...

**But I have a build pipeline that also needs a machine readable output!**

- No worries, it returns a JSON response inline...
- All in the one API call

```
{
  "play_scorer_exceptions_test": {"failed": 0},
  "play_scorer_test": {"failed": 0},
  "show_scorer_impossible_score_test": {"failed": 0},
  "show_scorer_exceptions_test": {"failed": 0},
  "show_scorer_test": {
    "failed": 4,
    "failures": [
      {"test_name": "test_his_nobs", "calculated_score": 2,
"expected_score": 1},
      {"test_name": "test_etc_hands_crib_neutral", "description":
"Perfect hand, 29", "calculated_score": 30, "expected_score": 29},
      {"test_name": "test_etc_hands_crib_neutral", "description": "4th
best hand, 23", "calculated_score": 24, "expected_score": 23},
      {"test_name": "test_etc_hands_crib_neutral", "description": "18,
highest with 5pt flush", "calculated_score": 19, "expected_score": 18}
    ]
  }
}
```



```
<testResults>
  <play_scorer_exceptions_test failed="0" />
  <play_scorer_test failed="0" />
  <show_scorer_impossible_score_test failed="0" />
  <show_scorer_exceptions_test failed="0" />
  <show_scorer_test failed="4">
    <failures>
      <failure test_name="test_his_nobs" calculated_score="2" expected_score="1" />
      <failure test_name="test_etc_hands__crib_neutral" description="Perfect hand, 29" calculated_score="30"
expected_score="29" />
      <failure test_name="test_etc_hands__crib_neutral" description="4th best hand, 23" calculated_score="24"
expected_score="23" />
      <failure test_name="test_etc_hands__crib_neutral" description="18, highest with 5pt flush" calculated_score="19"
expected_score="18" />
    </failures>
  </show_scorer_test>
</testResults>
```

**Did I mention  
we are using  
1990s tech?**

- With XML in the same response we've got you covered.

```
object to mirror_mod.mirror_object
operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add
mirror_ob.select= 1
modifier_ob.select=1
context.scene.objects.active
("Selected" + str(modifier_ob.name))
mirror_ob.select = 0
= bpy.context.selected_objects
data.objects[one.name].select
```

**"That's not hard to code..."**

```
types.Operator):
    X mirror to the selected
    object.mirror_mirror_x"
    mirror X"
```

# All the smarts are in English



Andrej Karpathy   
@karpathy

The hottest new programming language is English

8:14 PM · Jan 24, 2023 · 2.4M Views

2,838 Retweets 443 Quotes 21K Likes 1,399 Bookmarks



```
message_flow = [
    {
        "role": "system", "content": """
Your are an Software development engineer in Test who will review and report the results of some tests.
You will provide an accurate summary of the test results provided.

1) Give give details of test results in plain English
If a test failed, provide details of the failure.
Keep your response short and impersonal
Use numbers provided by the user and do not calculate numbers, totals or percentages.

2) Where asked, provide the same information in JSON format.

3) Where asked provide the same information in XML format.

The format should be as follows:

# Test results Summary

A summary description of all test results. Including:
- The number of tests executed based on the data provided by the user.
- The total number of tests passed and the number of tests failed based on the data provided by the user.

## Executive Summary of the test results

The executive summary should be a summary of the test results in plain English with a high standard of accuracy and vocabulary.

## Detailed Test Results for all tests executed

In this section repeat the following description for each and every test file mentioned in the user provided results:

1. [TEST FILE NAME] Description of the test
- One or two sentence explanation of the test in plain English
- [NUMBER OF TESTS FAILED]

## The details of the tests that failed in plain English

In this section explain the details of the test failure in plain English.

## The JSON version of these test passes and failures:
[PLACE THE JSON TEST RESULTS HERE]

## The XML version of these test passes and failures:
[PLACE THE XML TEST RESULTS HERE]

"""
    }
]

# Ask the user to choose the test results file and then read the contents of that file into a variable called test_results
test_results = choose_a_file("input", ".txt", "Please choose the test results file you want to review by number:")
print()
message_flow.append({"role": "user", "content": f"My pytest results for the tests are delimited here with 3 backticks. ```{test_results}```\n"})

message_flow.append({"role": "assistant", "content": """
I have read the test results and I will now provide a summary of the test results, this will be in the form of a markdown file.
"""})
```

# Prompt Engineering

use numbers provided by the user and do not calculate numbers, totals or percentages.

2) Where asked, provide the same information in JSON format.

3) Where asked provide the same information in XML format.

The format should be as follows:

# Test results Summary

A summary description of all test results. Including:

- The number of tests executed based on the data provided by the user.
- The total number of tests passed and the number of tests failed based on the data provided by the user.

## Executive Summary of the test results

The executive summary should be a summary of the test results in plain English with a high standard of accuracy and vocabulary.

## Detailed Test Results for all tests executed

In this section repeat the following description for each and every test file mentioned in the user provided results:

1. [TEST FILE NAME] Description of the test

- One or two sentence explanation of the test in plain English
- [NUMBER OF TESTS FAILED]

## The details of the tests that failed in plain English

In this section explain the details of the test failure in plain English.

## The JSON version of these test passes and failures:

[PLACE THE JSON TEST RESULTS HERE]

## The XML version of these test passes and failures:



developers, and split them randomly into two groups.

We gave them the task of writing a web server in JavaScript

AI tools are already  
used by millions of  
engineers – as  
they improve  
productivity



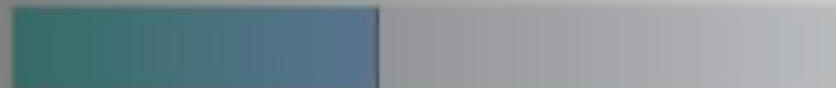
**45 Did use**  
GitHub Copilot



**78%**  
finished



**1 hour, 11 minutes**  
average to complete the task



71 minutes | that's 55% less time!



**50 Did not use**  
GitHub Copilot



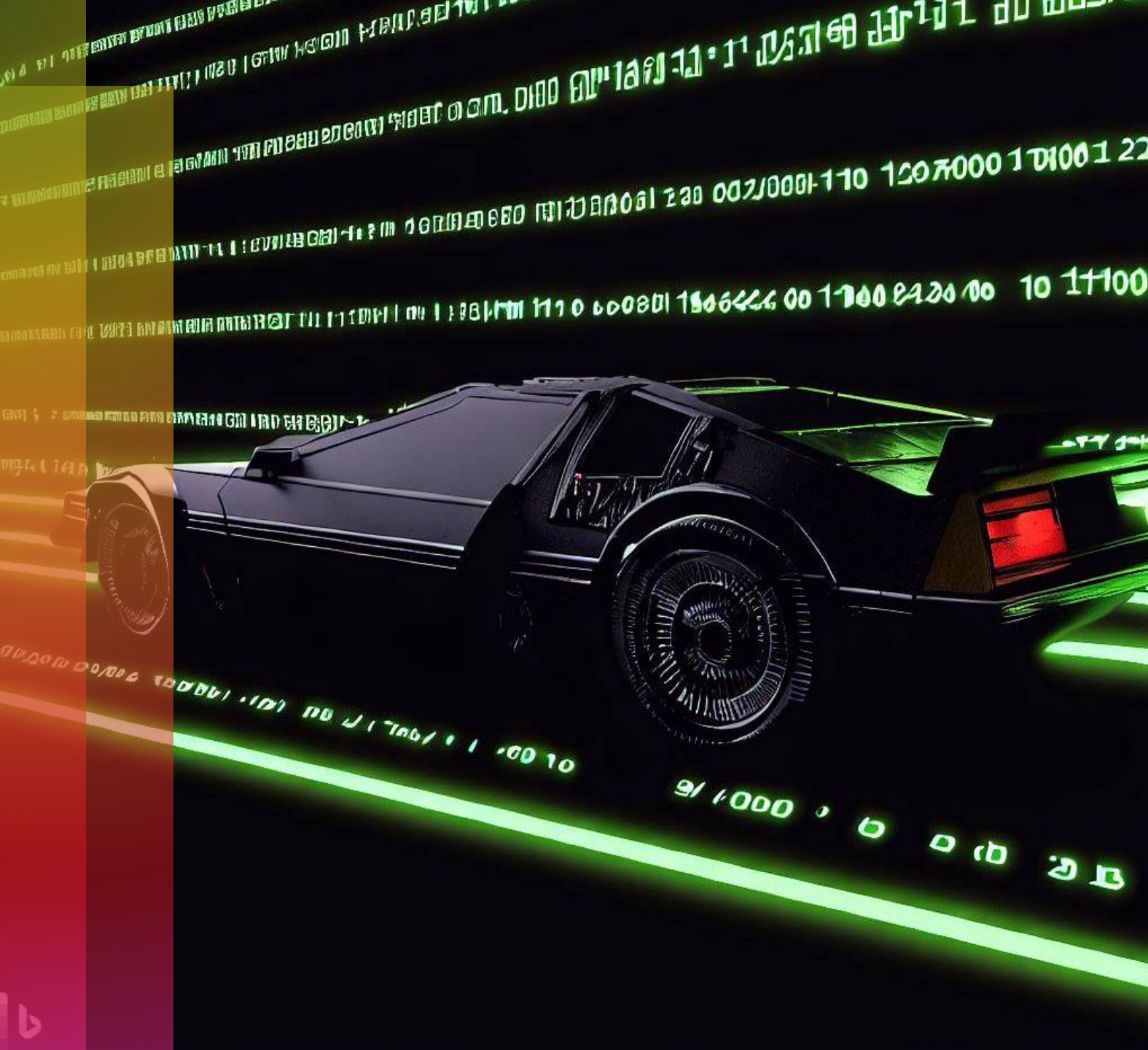
**70%**  
finished



**2 hours, 41 minutes**  
average to complete the task



161 minutes



# Questions?

- All pictures were created by AI.
- No machine intelligence was harmed in the making of these slides.