

UNIT 6.2 GRADED ASSIGNMENT

Group members

Ifra Saleem (2303.khi.deg.003)

Umaina Siddiqui (2023.KHI.DEG.033)

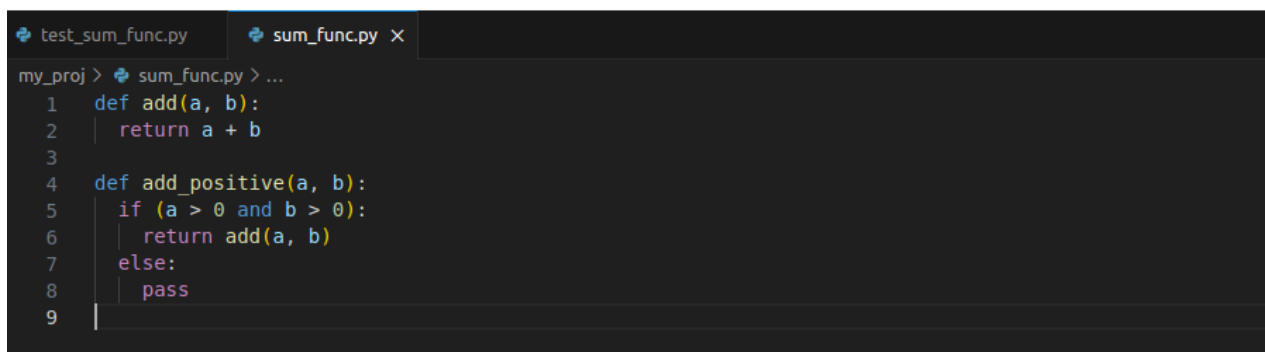
UNIT 6.2 GRADED ASSIGNMENT

Task:

Implement tests for

tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj project (there is a tests/test_sum_func.py file ready, missing the tests only).

Aim for 100% code coverage, branch coverage and condition coverage. Run the tests with pytest. Then run the tests with a flag producing coverage reports (you'll need to install pytest-cov at <https://pypi.org/project/pytest-cov/>).



```
test_sum_func.py  sum_func.py x
my_proj > sum_func.py > ...
1  def add(a, b):
2      return a + b
3
4  def add_positive(a, b):
5      if (a > 0 and b > 0):
6          return add(a, b)
7      else:
8          pass
9
```

Solution:

For the above program, in which we have two functions, we wrote 5 test cases. For the add function, we need to write only one test case which will give two numbers and the sum of those two numbers.

For the second function, we wrote four test cases. One test case will test the function for positive values, second test will test the function by giving a=-2, which means that here in this test case -2<0 so this condition is false, and the control will jump to the else branch and in the else branch we don't have any calculations so we will simply pass None to it.

In the 3rd test case of the second function b is 0 so this condition is also false as we need to give the values greater than 0.

In the 4th test case of the second function b is 0 and a is also 0 so this condition is also false.

So, these test cases have 100% statement, branch and condition coverage.

Note: We can achieve 100% coverage with 4 test cases only. The fifth one with $a=0, b=0$ is not needed but we can add it for feasibility.

Testing all combinations can be beneficial in some cases, while in others it may be unnecessary or impractical. Here are a few considerations to help you decide:

Function complexity: If the function has a small number of inputs and outputs, testing all combinations may be feasible. However, as the number of inputs and outputs increases, the number of combinations grows exponentially, making it impractical to test them all.

```
test_sum_func.py X sum_func.py
my_proj > tests > test_sum_func.py > test_add_a_zero_b_zero
1 from my_proj.sum_func import add, add_positive
2
3 def test_add_1_2_returns_3():
4     assert add(1, 2) == 3
5
6 def test_add_positive():
7     assert add_positive(2, 3) == 5
8
9 def test_add_a_negative_b_positive():
10    assert add_positive(-2, 3) == None
11
12 def test_add_b_zero():
13    assert add_positive(2, 0) == None
14
15 def test_add_a_zero_b_zero():
16    assert add_positive(0, 0) == None

(base) all@localhost:~/data_engineering_bootcamp_2303/tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj$ pytest tests/test_sum_func.py
===== test session starts =====
platform linux -- Python 3.10.6, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/all/data_engineering_bootcamp_2303/tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj
plugins: anyio-3.6.2, cov-4.0.0
collected 5 items

tests/test_sum_func.py ..... [100%]

===== 5 passed in 0.01s =====
(base) all@localhost:~/data_engineering_bootcamp_2303/tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj$

(base) all@localhost:~/data_engineering_bootcamp_2303/tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj$ pytest --cov=my_proj tests/
===== test session starts =====
platform linux -- Python 3.10.6, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/all/data_engineering_bootcamp_2303/tasks/6_writing_robust_readable_and_maintainable_code/day_2_testing/assignment/my_proj
plugins: anyio-3.6.2, cov-4.0.0
collected 5 items

tests/test_sum_func.py ..... [100%]

----- coverage: platform linux, python 3.10.6-final-0 -----
Name                Stmt    Miss  Cover
-----
__init__.py           0         0  100%
sum_func.py           6         0  100%
tests/__init__.py     0         0  100%
tests/test_sum_func.py 11         0  100%
TOTAL                 17         0  100%

===== 5 passed in 0.05s =====
```