Introduction

Last week, you applied black box testing techniques. This meant that you had no access to the source and had to write your tests based solely on the specification. This week, you will apply white box testing techniques. Therefore, I am providing you with the code you will be testing. Also, since white box testing doesn't care about the specification, the function under test is purely contrived and serves no purpose! Well, that isn't entirely true, the purpose is to give you an opportunity to flex your coverage muscles, but it doesn't *do* anything.

Course Learning Outcome(s):

- **Apply** testing techniques, including black-box and white-box techniques, automatic testing activities, and regression testing (CLO 4)

Module Learning Outcome(s):

- **Apply** white box testing techniques

Description

For this assignment you will have to achieve 100% Branch and Condition Coverage. For this assignment, this means that for each conditional statement, you need to have a test case for each combination of conditions. So if your conditional statement was `a or b` you would need to have a test case for each row of the following truth table:

`a`	`b`	**outcome**
T	T	T
T	F	T
F	T	T
F	F	F

If you recall what I said in the exploration, Branch and Condition coverage is usually not done due to the number of tests required. The reason we are doing it here, is we are

only testing a single function, not an entire program. The function under test is defined below.

Behold! *The Source*!

```
def contrived_func(val):
    # This function serves no logical purpose
    # DO NOT try to make sense of it!
    # Just make sure your tests cover everything requested
    # val is a numerical value
    if val < 150 and val > 100:
        return True
    elif val * 5 < 361 and val / 2 < 24:
        if val == 6:
            return False
        else:
            return True
    elif (val > 75 or val / 8 < 10) and val**val % 5 == 0:
        return True
    else:
        return False
```

Again, this function's only purpose is as a learning aid, it doesn't *do* anything. DO NOT try to rationalize it's behavior!

You need to write a series of unit tests that attempt to meet 100% Branch and Condition Coverage. Once submitted to Gradescope, the autograder will run the tests against a modified version of the above code. The `contrived_func` on Gradescope is functionally equivalent but includes `print` statements when each condition combination is triggered. There are 12 such triggers, labled C1-C12. You will only receive full credit for the autograded portion if your test suite triggers all 12 `print` statements.

Outside of the autograded portion, your test suite will also be graded based on how many tests were required to uncover all 12 triggers. Full points will only be rewarded if you can do it with 7 or fewer test cases. Please note, that if your tests do not pass the autograder with full marks, you will not be eligible for these Efficient Testing points. Also, for the purposes of the efficiency points, each **assert** counts as a single test, so no putting in multiple asserts in each test case.

Finally, as was the case last week, your test suite needs to be free of linting errors.

Hints

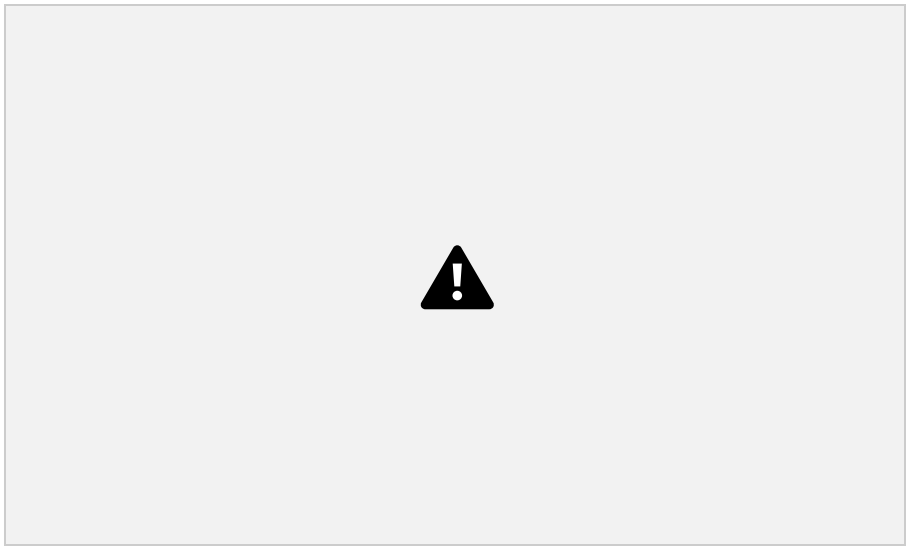
Truth Tables

I highly recommend you write out, by hand, the truth tables for each conditional statement. I believe in this so much I have provided you with the truth table for the first conditional below. Notice that the third row has T for the second condition. This is because the conditions are linked by `and`, which means if the first condition is false, then the second one does not execute. This makes the third and forth rows logically equivalent, so there is no need to write a test case for the forth row (it is also impossible for both conditions to be `False` at the same time).

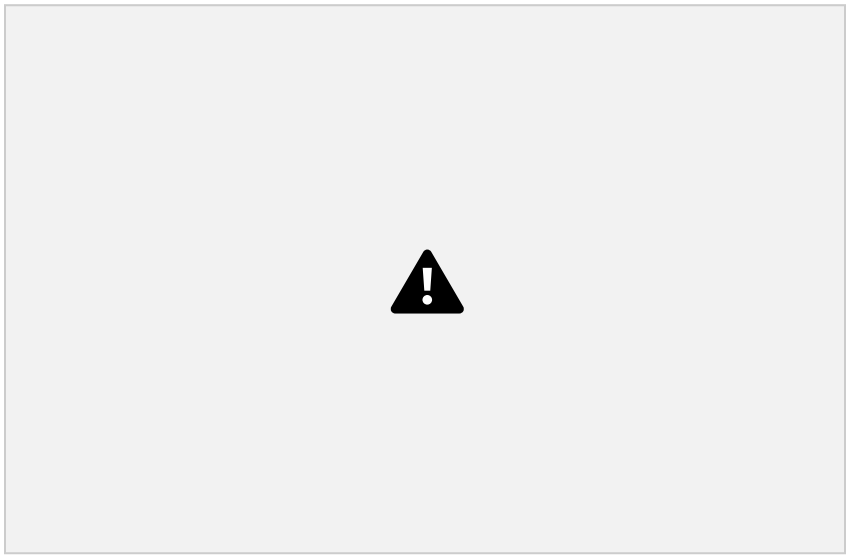
`val < 150`	`val > 100`	**outcome**	**Condition Label**
T	T	T	C1
T	F	F	C2
F	T	F	C3
F	F	F	C3

PyCharm Tools

PyCharm includes tools to track statement and branch coverage. These need to be enabled and/or installed. You can do this by going to your project settings and searching for "Coverage", which is found under "Build, Execute, Deployment". Where it says "When new coverage is gathered" ensure "Activate Coverage View" is checked. Also, if you want to examine branch coverage, ensure that it is checked under "Python coverage"



You can then right click on your testing file and select "Run...with Coverage". If this is the first time you have run with coverage, PyCharm may prompt you at the bottom to install the coverage module. You can do so simply by clicking "install" in the prompt.



Once run, it will open up a new panel where you can see the statement coverage for each file. You can also look in the "gutter" to see which lines haven't been executed. In case you are not familiar, the gutter is the area to the left of the code that you can put break points and often contains the line numbers.

Lines that are marked in red haven't been executed and those marked in yellow indicate branches that haven't been triggered.

To see which branch hasn't executed, left click on the yellow in the gutter. You will be told that the line "was hit", but which line wasn't jump to (i.e. the branch).

Unfortunately, there is no way for PyCharm to track condition coverage, so you will have to rely on testing on Gradescope.

Misc

You will need to include

```
if __name__ == '__main__':
    unittest.main()
```

-
- Feel free to take the provided source above and create your own `contrived_func.py` to run your tests against to help find errors with your testing file

What to turn in

Submit to Gradescope your testing suite; it must be named `tests.py`

- This file must be free of PEP8 linting errors