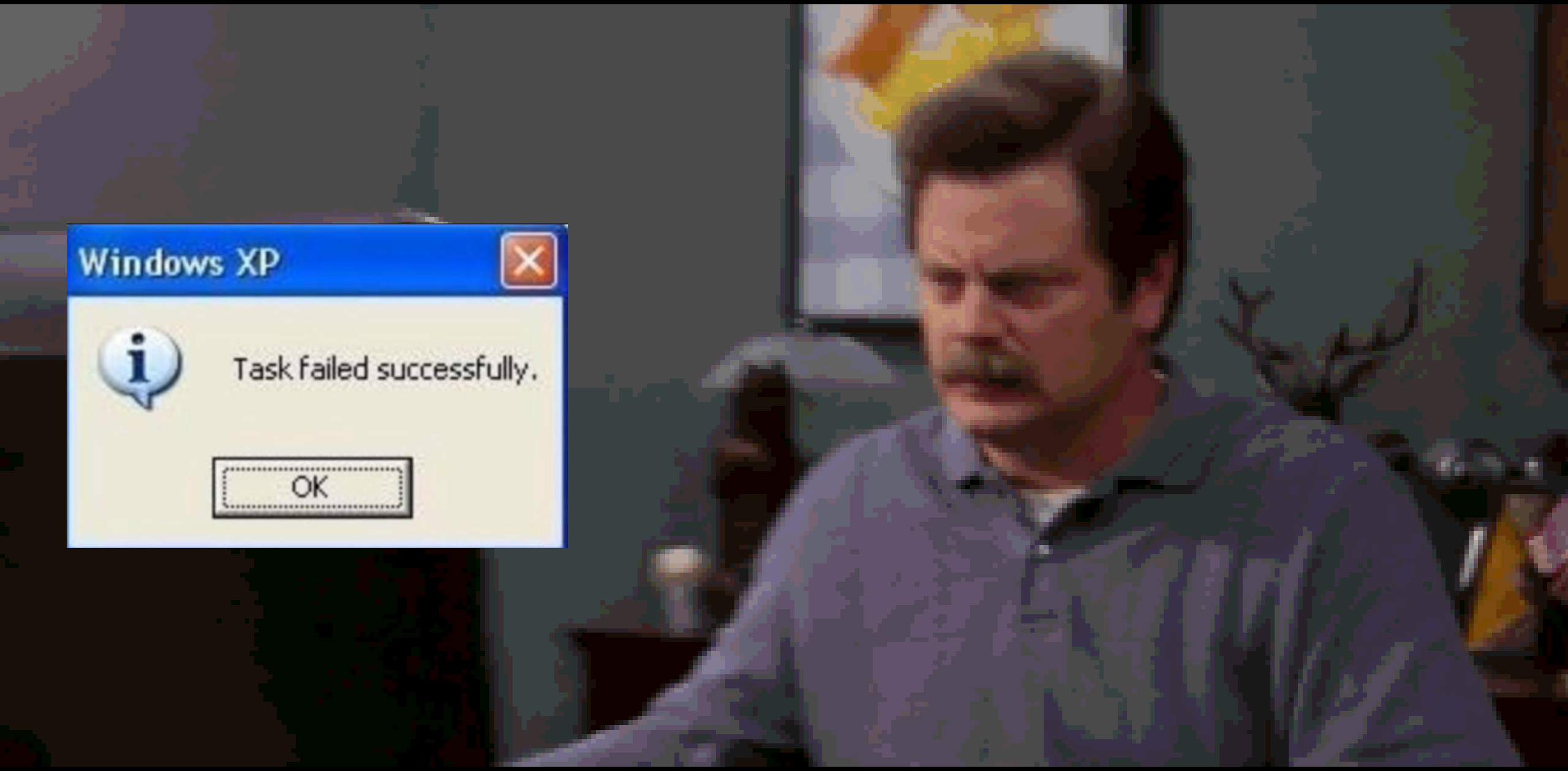

Windows XP
Task failed successfully.
OK

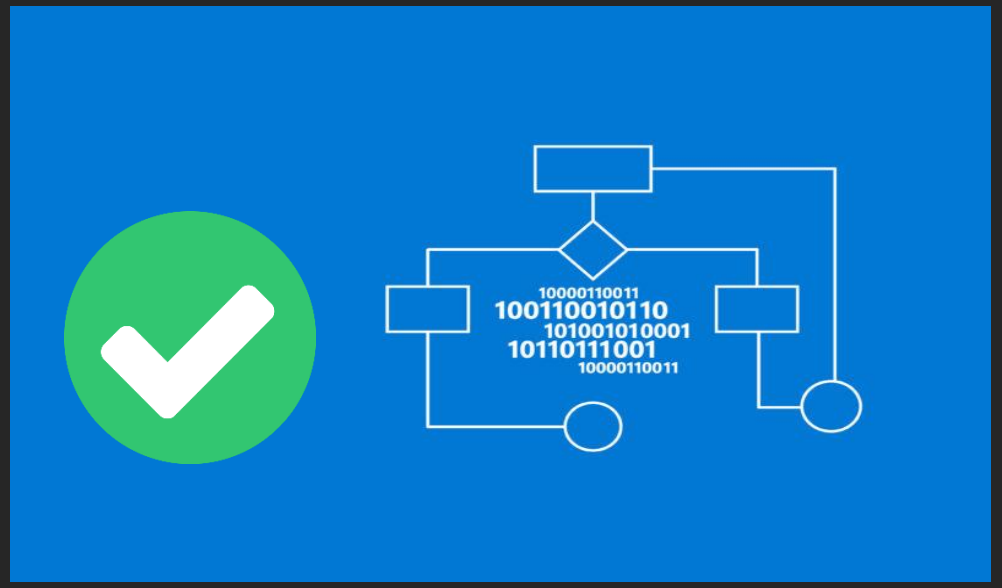
If your programs don’t work well,
they’ll be thrown away like that!

So, how do we know if our program is fine?

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿

There are many ways to verify the correctness of a program

TESTING



CORRECTNESS PROOF

OR COMBINE THEM! ;)

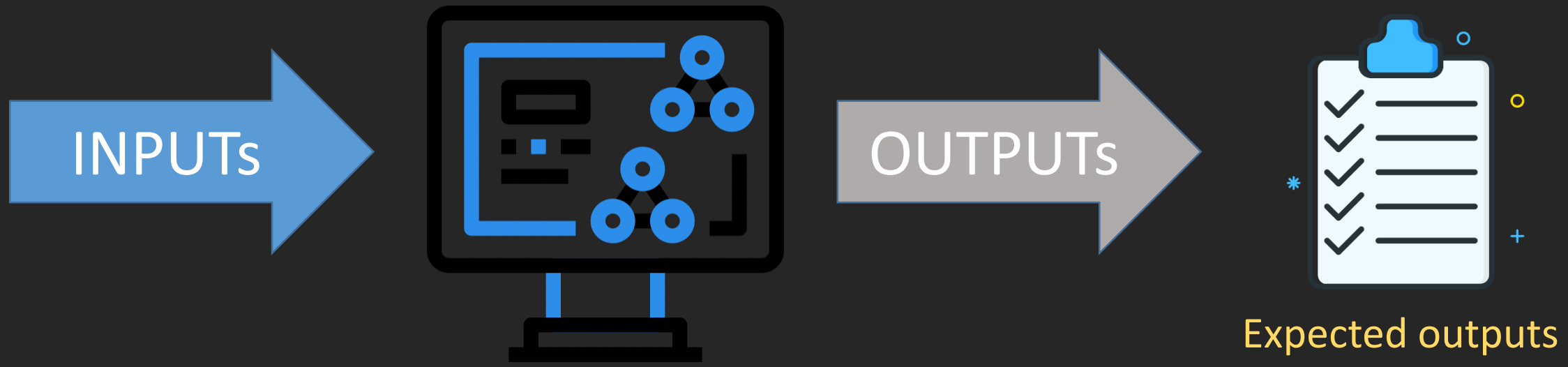
TESTING

Using Test Cases

What is Test Cases?

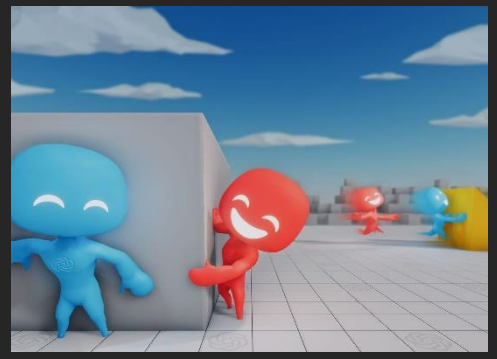
Test case is a set of data include our INPUTs and attached OUTPUTs.

We use test case to verify that our program return expected results.



How to write Test cases for a program?

- Test cases should be saved in text file.
- One test case is used for one test purpose.
- We start with writing small, simple test cases.
- Then we continue with some test cases including special values or abnormal values. Programs often return wrong results because of those kinds of value.
- If we understand how our code work, just write some test cases for “corner” cases.
- We also need some big test cases to check the program’s ability.

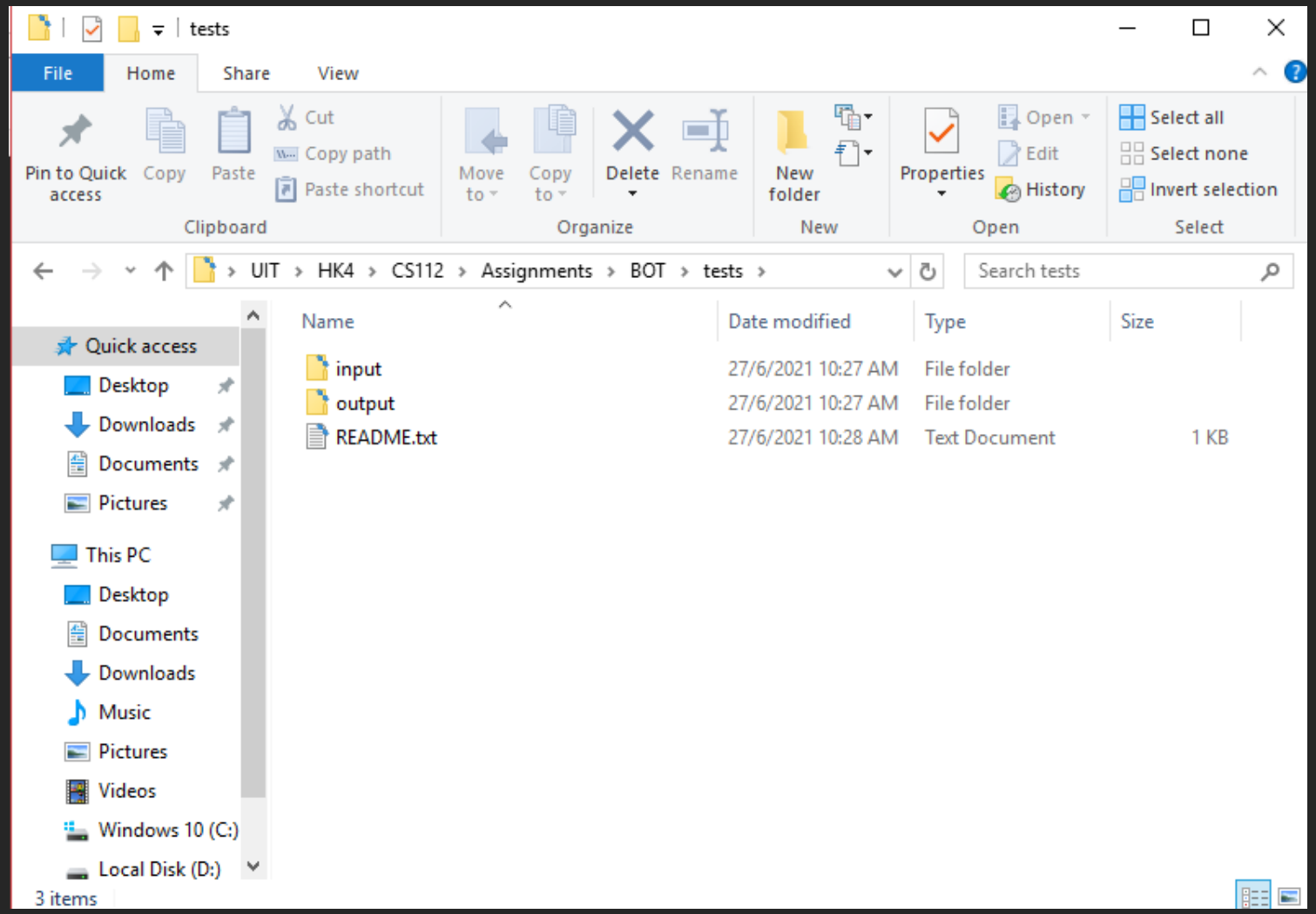


Basic format of test case

- Test ID: such as “input1.txt” – “output1.txt”, “001.txt” (in input folder) – “001.txt” (in output folder).
- Test data: we save input data in input files and expected output in output files.
- Divide test cases into independent folder, remember, One test case is used for One purpose.
- Don’t forget to describe what to be verified for test cases.

Example: write test cases for BOT program and execute test by Python

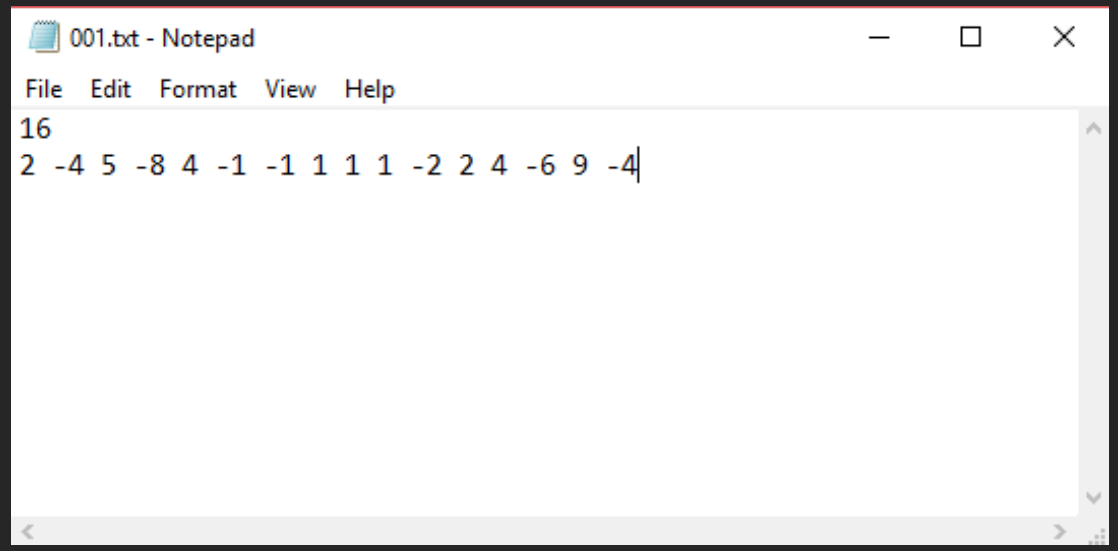
- Create “tests” folder including “input” and “output”.



Example: write test cases for BOT program and execute test by Python

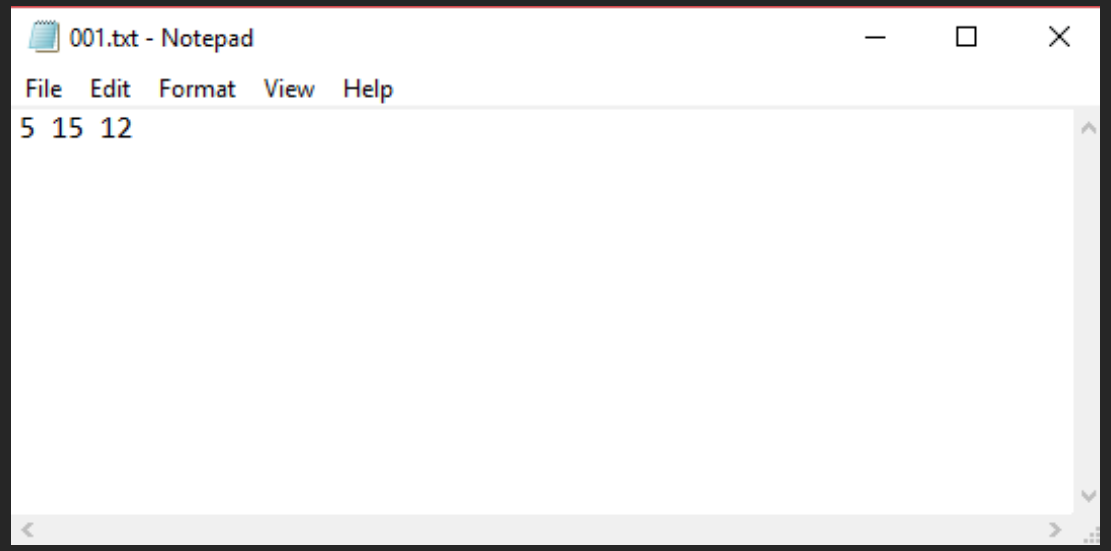
- Write test data:

INPUT



Save in “input” folder as “001.txt”

EXPECTED OUTPUT



Save in “output” folder as “001.txt”

Example: write test cases for BOT program and execute test by Python

- Create simple test runner:

C: > Users > Admin > Desktop > UIT > HK4 > CS112 > Assignments > BOT > unittest.py > ...

```python
import bot

def test(input, output):
    passed = 'PASSED'
    failed = 'FAILED'
    if bot.max_subarray(input) == output:
        return passed
    return failed

def main():
    count = int(0)
    numOfTest = int(5)
    for i in range(numOfTest):
        inpPath = 'C:/Users/Admin/Desktop/UIT/HK4/CS112/Assignments/BOT/tests/input/00' + str(i+1) + '.txt'
        outPath = 'C:/Users/Admin/Desktop/UIT/HK4/CS112/Assignments/BOT/tests/output/00' + str(i+1) + '.txt'
        inpFile = open (inpPath, 'r')
        outFile = open (outPath, 'r')
        n = int(inpFile.readline())
        temp1 = inpFile.readline()
        input = list(map(int, temp1.split()))
        temp2 = outFile.readline()
        output = list(map(int, temp2.split()))

        result = test(input, output)
        if result == 'PASSED':
            count += 1
        print('TEST ' + str(i+1) + ': ' + result)

    print('RESULT: ' + str(count) + '/' + str(numOfTest))

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

Example: write test cases for BOT program and execute test by Python

- RESULT:

```
PS C:\Users\Admin\Desktop\UIT\HK4\CS112\Assignments\BOT>  c:; cd 'c:\Users\Admin\Desktop\UIT\HK4\CS112\Assignments\BOT'; & 'C:\Users\Admin\Ap
pData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Admin\.vscode\extensions\ms-python.python-2021.5.926500501\pythonFiles\lib\python\
debugpy\launcher' '57804' '--' 'c:\Users\Admin\Desktop\UIT\HK4\CS112\Assignments\BOT\unittest.py'
TEST 1: PASSED
TEST 2: FAILED
TEST 3: PASSED
TEST 4: PASSED
TEST 5: PASSED
RESULT: 4/5
PS C:\Users\Admin\Desktop\UIT\HK4\CS112\Assignments\BOT>
```

We can use unittest in Python!

- A unit test is a smaller test, one that checks that a single component operates in the right way. A unit test helps you to isolate what is broken in your application and fix it faster.

- `unittest` has been built into the Python standard library since version 2.1. You'll probably see it in commercial Python applications and open-source projects.

- `unittest` contains both a testing framework and a test runner.

Run test cases using UnitTest in Python

```python
# test_simple_unittest.py
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('python'.upper(), 'PYTHON')

    def test_isupper(self):
        self.assertTrue('PYTHON'.isupper())
        self.assertFalse('Python'.isupper())

    def test_islower(self):
        self.assertTrue('PYTHON'.islower())
        self.assertFalse('Python'.islower())

    def test_split(self):
        test_string = 'python is a best language'
        self.assertEqual(test_string.split(),
                         ['python', 'is', 'a', 'best', 'language'])
        # check that test_string.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            test_string.split(2)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Run test cases using UnitTest in Python

```
> python .\test_simple_unittest.py
test_islower (__main__.TestStringMethods) ... FAIL
test_isupper (__main__.TestStringMethods) ... ok
test_split (__main__.TestStringMethods) ... ok
test_upper (__main__.TestStringMethods) ... ok

======================================================================
FAIL: test_islower (__main__.TestStringMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
File ".\test_simple_unittest.py", line 14, in test_islower
self.assertTrue('PYTHON'.islower())
AssertionError: False is not true

----------------------------------------------------------------------
Ran 4 tests in 0.002s

FAILED (failures=1)
```

Thank you! <3