Unit Test

????

- ??????????????????(function)???(method)?????????????????????????????????????
- ????????????????????????????? unittest, Pytest ???????????
- ?????? CI/CD ?????????????

Pytest

- YT: How To Write Unit Tests in Python • Pytest Tutorial - YouTube

unittest

Methods

- `.assertEqual(a, b)` : checks that a == b
- `.assertNotEqual(a, b)` : checks that a != b
- `.assertTrue('FOO'.isupper())` : checks that bool(x) is True
- `.assertFalse('Foo'.isupper())` : checks that bool(x) is False

Example 1: rearrange.py

```
#!/usr/bin/env python3


import re


def rearrange_name(name):
  result = re.search(r"^([\w .]*), ([\w .]*)$", name)
  if result is None:
    return name
  return "{} {}".format(result[2], result[1])
```

rearrange_test.py :

```
#!/usr/bin/env python3
```

```
import unittest

from rearrange import rearrange_name

class TestRearrange(unittest.TestCase):

  def test_basic(self):  # Basic test case
    testcase = "Lovelace, Ada"
    expected = "Ada Lovelace"
    self.assertEqual(rearrange_name(testcase), expected)

  def test_empty(self):  # Edge case, such as zero, blank, negative numbers, or extremely large numbers
    testcase = ""
    expected = ""
    self.assertEqual(rearrange_name(testcase), expected)

  def test_double_name(self):   # Additional test case
    testcase = "Hopper, Grace M."
    expected = "Grace M. Hopper"
    self.assertEqual(rearrange_name(testcase), expected)

  def test_one_name(self):      # Additional test case
    testcase = "Voltaire"
    expected = "Voltaire"
    self.assertEqual(rearrange_name(testcase), expected)

# Run the tests
unittest.main()
```

> Tip: ? Jupyter ???? `unittest.main()` ??????????????? `unittest.main(argv = ['first-arg-is-ignored'], exit = False)` ?

The output of the result:

```
.
----------------------------------------------------------------------
Ran 4 test in 0.000s
```

```
OK
```

Example 2: cakefactory.py

```
#!/usr/bin/env python3

from typing import List

class CakeFactory:
 def __init__(self, cake_type: str, size: str):
   self.cake_type = cake_type
   self.size = size
   self.toppings = []

   # Price based on cake type and size
   self.price = 10 if self.cake_type == "chocolate" else 8
   self.price += 2 if self.size == "medium" else 4 if self.size == "large" else 0

 def add_topping(self, topping: str):
     self.toppings.append(topping)
     # Adding 1 to the price for each topping
     self.price += 1

 def check_ingredients(self) -> List[str]:
     ingredients = ['flour', 'sugar', 'eggs']
     ingredients.append('cocoa') if self.cake_type == "chocolate" else ingredients.append('vanilla extract')
     ingredients += self.toppings
     return ingredients

 def check_price(self) -> float:
     return self.price

# Example of creating a cake and adding toppings
cake = CakeFactory("chocolate", "medium")
cake.add_topping("sprinkles")
cake.add_topping("cherries")
cake_ingredients = cake.check_ingredients()
cake_price = cake.check_price()
```

```
cake_ingredients, cake_price
```

cakefactory_test.py

```
#!/usr/bin/env python3

import unittest
from cakefactory import CakeFactory

class TestCakeFactory(unittest.TestCase):
 def test_create_cake(self):
   cake = CakeFactory("vanilla", "small")
   self.assertEqual(cake.cake_type, "vanilla")
   self.assertEqual(cake.size, "small")
   self.assertEqual(cake.price, 8) # Vanilla cake, small size

 def test_add_topping(self):
     cake = CakeFactory("chocolate", "large")
     cake.add_topping("sprinkles")
     self.assertIn("sprinkles", cake.toppings)

 def test_check_ingredients(self):
     cake = CakeFactory("chocolate", "medium")
     cake.add_topping("cherries")
     ingredients = cake.check_ingredients()
     self.assertIn("cocoa", ingredients)
     self.assertIn("cherries", ingredients)
     self.assertNotIn("vanilla extract", ingredients)

 def test_check_price(self):
     cake = CakeFactory("vanilla", "large")
     cake.add_topping("sprinkles")
     cake.add_topping("cherries")
     price = cake.check_price()
     self.assertEqual(price, 13) # Vanilla cake, large size + 2 toppings



# Running the unittests
unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(TestCakeFactory))
```

This results in the output:

```
..F
======================================================================
===
FAIL: test_check_price (__main__.TestCakeFactory)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-9-32dbf74b3655>", line 33, in test_check_price
    self.assertEqual(price, 13) # Vanilla cake, large size + 2 toppings
AssertionError: 14 != 13


----------------------------------------------------------------------
Ran 4 tests in 0.007s


FAILED (failures=1)
<unittest.runner.TextTestResult run=4 errors=0 failures=1>
```

The program calls the *TextTestRunner()* method, which returns a runner (*TextTestResult*). It says one failure occurred: the statement *self.assertEqual(price, 13)* was incorrect, as it should have been 14. How can we correct that part of the test? Update that part of the code to the following:

```
import unittest



# Fixing the test_check_price method
class TestCakeFactory(unittest.TestCase):
 # ... Other tests remain the same

 def test_check_price(self):
     cake = CakeFactory("vanilla", "large")
     cake.add_topping("sprinkles")
     cake.add_topping("cherries")
     price = cake.check_price()
     self.assertEqual(price, 14) # Vanilla cake, large size + 2 toppings

# Re-running the unittests
unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(TestCakeFactory))
```

And now the program works as expected, as the results provide no failures and are:

```
.
----------------------------------------------------------------------
Ran 4 test in 0.002s


OK
```

Revision #44
Created 20 June 2024 19:47:42 by Admin
Updated 20 December 2024 14:33:03 by Admin