

Introduzione al Unit Testing

Verifica del corretto funzionamento di singole unità di codice

-WWW.DUCCIO.ME

- Un'unità è tipicamente una funzione, un metodo o una classe
- Isolato dalle dipendenze esterne (database, API, filesystem)
- Automatizzato e ripetibile
- Benefici:
 - Documentazione eseguibile
 - Prevenzione di regressioni
 - Miglioramento del design del codice
 - o Confidenza nelle modifiche
- Framework popolari:
 - Python: unittest, pytest
 - JavaScript: Jest, Mocha

Anatomia di un Unit Test

- Setup: preparazione dell'ambiente di test
- Esecuzione: chiamata alla funzione o al metodo da testare
- Verifica: controllo che i risultati siano quelli attesi
- Teardown: pulizia dell'ambiente dopo il test

Perché fare Unit Testing?

Vantaggi immediati

- Individuazione precoce dei bug
- Documentazione vivente del comportamento atteso del codice
- Facilita il refactoring e la manutenzione

Vantaggi a lungo termine

- Maggiore fiducia nel codice
- Riduzione del "debito tecnico"
- Sviluppo più rapido di nuove funzionalità
- Onboarding più semplice per nuovi sviluppatori

— WWW.DUKCIO.ME——



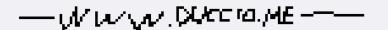
Principi del buon Unit Testing

FIRST

- Fast: i test devono essere veloci
- Independent: ogni test deve essere indipendente dagli altri
- Repeatable: i test devono dare lo stesso risultato ad ogni esecuzione
- Self-Validating: i test devono determinare autonomamente se hanno successo o no
- Timely: i test dovrebbero essere scritti prima o insieme al codice

Le 3 A

- Arrange: preparazione dell'ambiente
- Act: esecuzione del codice da testare
- Assert: verifica dei risultati





Unit Test in Python con unittest

1

廖

```
1 # file: calcoli.py
2 def somma(a, b):
3    return a + b
4
5 def moltiplica(a, b):
6    return a * b
7
8 def dividi(a, b):
9    if b == 0:
10     raise ValueError("Divisione per zero non consentita")
11    return a / b
```

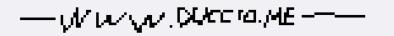


```
2 import unittest
 3 from calcoli import somma, moltiplica, dividi
 5 class TestCalcoli(unittest.TestCase):
       def test_somma(self):
           self.assertEqual(somma(5, 3), 8)
           self.assertEqual(somma(-1, 1), 0)
           self.assertEqual(somma(0, 0), 0)
       def test_moltiplica(self):
           self.assertEqual(moltiplica(4, 3), 12)
           self.assertEqual(moltiplica(0, 5), 0)
           self.assertEqual(moltiplica(-2, 3), -6)
       def test dividi(self):
           self.assertEqual(dividi(10, 2), 5)
           self.assertEqual(dividi(0, 5), 0)
           self.assertAlmostEqual(dividi(5, 3), 1.666666666666667)
       def test_dividi_zero(self):
           with self.assertRaises(ValueError):
               dividi(10, 0)
26 if __name__ == "__main__":
       unittest.main()
```

--- W W JW JW DWGG 10.ME ----

unittest documentazione

Method	Checks that
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	<pre>not isinstance(a, b)</pre>



Unit Test in Python con pytest

```
1 # file: calcoli.py
2 def somma(a, b):
3    return a + b
4
5 def moltiplica(a, b):
6    return a * b
7
8 def dividi(a, b):
9    if b == 0:
10         raise ValueError("Divisione per zero non consentita")
11    return a / b
```

```
• • •
       5 class Calculator:
             def add(self, a, b):
                 return a + b
廖
             def divide(self, a, b):
                 if b == 0:
                     raise ValueError("Impossibile dividere per zero")
                 return a / b
      15 def test_add():
             calc = Calculator()
             assert calc.add(2, 3) == 5
      19 def test_divide():
            calc = Calculator()
             assert calc.divide(10, 2) == 5
      23 import pytest
      25 def test_divide_by_zero():
            calc = Calculator()
             with pytest.raises(ValueError):
                 calc.divide(10, 0)
```

 $-\psi \omega_{
m W}$, $\omega_{
m W}$, $\omega_{
m W}$



Unit Testing in JavaScript/Node.js (Jest)

```
1 // Installazione: npm install --save-dev jest
2
3 // La classe che vogliamo testare - calculator.js
4 class Calculator {
5  add(a, b) {
6  return a + b;
7  }
8
9  divide(a, b) {
10  if (b === 0) {
11  throw new Error("Impossibile dividere per zero");
12  }
13  return a / b;
14  }
15 }
16
17 module.exports = Calculator;
```

- Avvio del testing con:
 - a. npm run test

```
2 const Calculator = require('./calculator');
 4 describe('Calculator', () => {
 5 let calc;
     beforeEach(() => {
   calc = new Calculator();
     test('should add two numbers correctly', () => {
      const result = calc.add(2, 3);
17 expect(result).toBe(5);
     test('should divide two numbers correctly', () => {
      const result = calc.divide(10, 2);
      expect(result).toBe(5);
     test('should throw error when dividing by zero', () => {
      expect(() => calc.divide(10, 0)).toThrow("Impossibile dividere per
   zero"):
29 });
```

—WWW.DUKC10.ME -─—

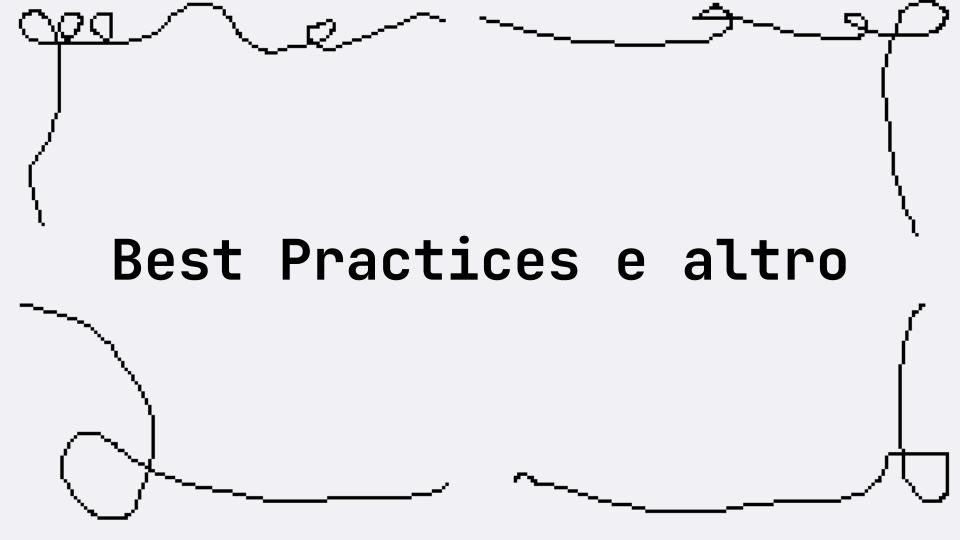
Mocha (test runner) e Chai (libreria di asserzioni)

```
1 // Installazione: npm install --save-dev jest
2
3 // La classe che vogliamo testare - calculator.js
4 class Calculator {
5   add(a, b) {
6    return a + b;
7  }
8
9   divide(a, b) {
10   if (b === 0) {
11    throw new Error("Impossibile dividere per zero");
12  }
13   return a / b;
14  }
15 }
16
17 module.exports = Calculator;
```

- Avvio del testing con:
 - a. npm run test

```
4 const Calculator = require('./calculator');
 5 const expect = require('chai').expect;
 7 describe('Calculator', function() {
 8 let calc:
10 beforeEach(function() {
11 calc = new Calculator():
it('should add two numbers correctly', function() {
const result = calc.add(2, 3);
16 expect(result).to.equal(5);
it('should divide two numbers correctly', function() {
      const result = calc.divide(10, 2);
    it('should throw error when dividing by zero', function() {
      expect(function() {
       calc.divide(10, 0);
}).to.throw("Impossibile dividere per zero");
 30 });
```

— WWW.DUKCIAME——



Best Practices

Cosa testare

- Logica di business complessa
- Edge cases e gestione degli errori
- Funzionalità critiche dell'applicazione

Cosa non testare

- Codice di terze parti
- Getter e setter semplici
- Implementazioni triviali

Organizzazione dei test

- Struttura dei file di test (mirror della struttura del codice)
- Naming convenzionale (test_*.py, *.test.js, *_test.go)
- Raggruppamento logico dei test





Test-Driven Development (TDD)

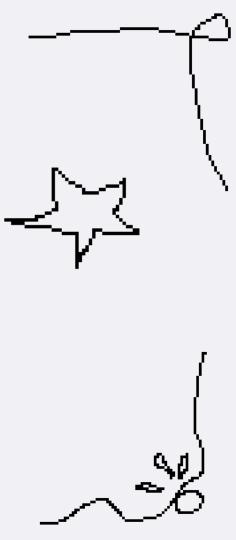
Il ciclo Red-Green-Refactor

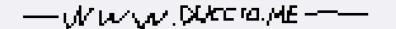
- 1. Red: Scrivi un test che fallisce
- 2. Green: Scrivi il minimo codice necessario per far passare il test
- 3. **Refactor:** Migliora il codice mantenendo i test in verde



Esempio di TDD in Python

```
• • •
 2 def test_validate_password():
       assert validate password("abc123") == False # troppo corto
       assert validate_password("abcdefgh") == False # nessun numero
       assert validate_password("12345678") == False # nessuna lettera
       assert validate password("abcd1234") == True # valida
 11 def validate_password(password):
       if len(password) < 8:</pre>
           return False
       if not any(c.isdigit() for c in password):
           return False
       if not any(c.isalpha() for c in password):
           return False
       return True
 21 def validate_password(password):
       has min length = len(password) >= 8
       has_digit = any(c.isdigit() for c in password)
       has_letter = any(c.isalpha() for c in password)
       return has_min_length and has_digit and has_letter
```





Conclusioni e Risorse

Riassunto

- I test unitari sono fondamentali per la manutenzione del codice
- Python e JavaScript offrono diversi framework con approcci simili
- Il TDD è una pratica che migliora la qualità del codice

Passaggi successivi

- Integrare i test in CI/CD
- Esplorare altre tipologie di test (integrazione, sistema, end-to-end)
- Misurare la copertura del codice

Risorse utili

- <u>Documentazione ufficiale unittest</u>
- <u>Documentazione Jest</u>

