Python Cheatsheet – Testing in Python

> **Curso:** Google IT Automation with Python
> **Módulo:** Testing in Python
> **Tema:** Pruebas automatizadas de código con `unittest` y buenas prácticas

Introducción

El **testing** en Python permite validar que el código funcione como se espera antes de desplegarlo.
Principales beneficios:

- Detectar errores temprano
- Garantizar estabilidad al cambiar el código
- Facilitar mantenimiento y refactorización

Módulo clave: `unittest` (incluido en la biblioteca estándar)

Estructura Básica de un Test con `unittest`

```
import unittest

def suma(a, b):
    return a + b

class TestSuma(unittest.TestCase):
    def test_suma_positivos(self):
        self.assertEqual(suma(2, 3), 5)

    def test_suma_negativos(self):
        self.assertEqual(suma(-1, -1), -2)

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

- `TestCase` → clase base para pruebas
- Métodos que comienzan con `test_` son automáticamente detectados
- `assertEqual(a, b)` → verifica igualdad de valores

Métodos de Assert Más Usados

Método	Descripción	Ejemplo
`assertEqual(a, b)`	a == b	`self.assertEqual(suma(2,3),5)`
`assertNotEqual(a, b)`	a != b	`self.assertNotEqual(2+2,5)`

Método	Descripción	Ejemplo
`assertTrue(x)`	x es True	`self.assertTrue(5>2)`
`assertFalse(x)`	x es False	`self.assertFalse(2>5)`
`assertIs(a, b)`	a es b	`self.assertIs(obj1, obj2)`
`assertIsNone(x)`	x es None	`self.assertIsNone(valor)`
`assertRaises(Exception, func, *args)`	Lanza excepción	`self.assertRaises(ValueError, int, "a")`

◈ Configuración y Limpieza

- `setUp()` → se ejecuta antes de cada test
- `tearDown()` → se ejecuta después de cada test

```
class TestEjemplo(unittest.TestCase):
    def setUp(self):
        self.lista = [1,2,3]

    def tearDown(self):
        self.lista = []

    def test_len(self):
        self.assertEqual(len(self.lista), 3)
```

◈ Ejecutar Tests

Desde terminal:

```
python -m unittest test_script.py
```

Opciones:

- `python -m unittest discover` → busca tests automáticamente
- `python -m unittest -v` → modo verbose (detalla cada test)

◈ Tests Automatizados y Buenas Prácticas

- Cada función o módulo debe tener **tests propios**
- Nombrar archivos de test con `test_*.py`
- Probar tanto **casos positivos como negativos**
- Usar `assertRaises` para validar errores
- Ejecutar tests frecuentemente durante desarrollo

- Mantener tests **independientes**: no deben depender del orden

---

Ejemplo Completo

```
import unittest

def dividir(a, b):
    if b == 0:
        raise ValueError("No se puede dividir por cero")
    return a / b

class TestDivision(unittest.TestCase):
    def test_division_normal(self):
        self.assertEqual(dividir(10,2), 5)

    def test_division_por_cero(self):
        with self.assertRaises(ValueError):
            dividir(5,0)

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

Referencias oficiales

- Python `unittest` module
- Real Python – Python Testing with `unittest`
- Python Testing Tutorial – GeeksforGeeks