UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

CENTRO DE CIENCIAS EXACTAS E INGENIERIAS

DIVISION DE INGENIERIAS

Daniel Padilla Mora - 221350184

Computación Tolerante a Fallas

Otras herramientas para el manejar errores (Par. 1)

-Asterión: Son declaraciones que se supone que siempre son verdaderas en un punto particular del código. Si la afirmación resulta ser falsa, se lanza una excepción.

```
def dividir(a, b):
    assert b != 0, "Error: División por cero"
    return a / b
resultado = dividir(10, 0)
```

-Logging: Te permite registrar información sobre errores y otros eventos importantes en tu aplicación.

```
import logging

logging.basicConfig(level=logging.ERROR)

def dividir(a, b):
    try:
        resultado = a / b
    except ZeroDivisionError:
        logging.error("Intento de división por cero.")
        return None
    return resultado
```

-Manejo de errores HTTP: En el desarrollo web, es común manejar errores HTTP para indicar el resultado de las solicitudes. Por ejemplo, en una API REST, podrías devolver un código de estado HTTP específico para indicar si una operación fue exitosa o si ocurrió un error.

```
from flask import Flask, jsonify
```

```
app = Flask(__name__)


@app.route('/dividir/<int:a>/<int:b>')
def dividir(a, b):
    try:
        resultado = a / b
        return jsonify({"resultado": resultado})
    except ZeroDivisionError:
        return jsonify({"error": "División por cero"}), 400
    except Exception as e:
        return jsonify({"error": str(e)}), 500


if __name__ == '__main__':
    app.run(debug=True)
```

-Pruebas unitarias: puede ayudar a identificar y corregir errores antes de que lleguen a la producción, ya que . Herramientas como unittest en Python, JUnit en Java, y Jest en JavaScript son comunes para escribir pruebas unitarias.

```
import unittest


def dividir(a, b):
    if b == 0:
        raise ValueError("División por cero no permitida.")
    return a / b


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

```
    def test_division_por_cero(self):
        with self.assertRaises(ValueError):
            dividir(10, 0)

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

Conclusión

En la práctica, todos funcionan de manera similar, pero en la teoría, cada uno cumple con una función específica. Uno permite tener un registro de los errores que tiene, otro se usa para saber si las solicitudes de una página web funcionan correctamente, otro para detectar si un bloque pequeño de código tiene algún fallo, y otro simplemente te avisa si hay un error. Cada uno tiene una función para cada tipo de situación

Bibliografía

7. Simple statements. (n.d.). Python Documentation.

https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement

HOWTO hacer registros (Logging). (n.d.). Python Documentation.

https://docs.python.org/es/3/howto/logging.html

Códigos de estado de respuesta HTTP - HTTP | MDN. (2022, November 25). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTTP/Status

unittest — Infraestructura de tests unitarios — documentación de Python - 3.9.17. (n.d.). https://docs.python.org/es/3.9/library/unittest.html