ChatGPT

5. Manejo de Errores y Excepciones en Python

5.1 Mecanismos de generación de errores en Python

En Python, los errores se clasifican en dos tipos principales: **errores de sintaxis** y **excepciones** de tiempo de ejecución [1] [2]. Un error de sintaxis (por ejemplo, olvidar dos puntos `:` en una estructura de control) es detectado por el intérprete antes de ejecutar el programa [3]. En cambio, una excepción ocurre durante la ejecución cuando sucede algo inesperado (por ejemplo, división por cero o uso de una variable no definida) [1]. Si no se maneja, la excepción produce un mensaje de error (traceback) que incluye el tipo de excepción y la línea de código donde ocurrió el problema [1] [3].

Los principales tipos de excepciones en Python incluyen: - **ZeroDivisionError**: ocurre al dividir un número por cero [4]. - **NameError**: ocurre al referirse a una variable que no existe [5]. - **TypeError**: ocurre al aplicar operaciones a tipos de datos incompatibles [6]. - Otros ejemplos comunes son **ValueError**, **OSError**, **KeyboardInterrupt**, etc. Estas excepciones provienen de la jerarquía de `Exception` (la mayoría de excepciones de usuario heredan de `Exception`, que a su vez deriva de `BaseException`) [7].

5.2 Control de excepciones en Python

Python utiliza las sentencias `try` / `except` (equivalente a try/catch) para capturar excepciones y controlar el flujo del programa [8]. El bloque `try` contiene el código que podría causar errores; si no ocurre ninguna excepción, se ejecuta normalmente. Si aparece una excepción, se busca un bloque `except` compatible y se ejecuta la primera cláusula que coincida [8]. Se pueden tener múltiples cláusulas `except` para distintos tipos de excepción, o incluso capturar varias excepciones a la vez usando una tupla [9]. Por ejemplo:

```
try:
    x = int("abc")
except (TypeError, ValueError) as e:
    print(f"Error de tipo o valor: {e}")  # captura TypeError o ValueError [10]
```

También es posible capturar la instancia de la excepción (`as e`) para obtener más detalles. Por ejemplo:

```
try:
    valor = int(input("Ingrese un entero: "))
except ValueError as e:
    print(f"Entrada no válida ({e}); intente de nuevo.")
```

Opcionalmente, se puede usar `else` para ejecutar código cuando **no** hubo excepción, y `finally` para código que debe ejecutarse siempre (por ejemplo, liberar recursos) [11]. El bloque `finally` es útil para cerrar archivos o conexiones sin importar si hubo errores [11] [12]. Python provee además la declaración `with` para objetos con acciones de limpieza predefinidas. Por ejemplo:

```
with open("datos.txt") as f:
    datos = f.read()
```

Aquí el archivo se cierra automáticamente al salir del bloque `with`, incluso si ocurre un error durante la lectura [13].

Como ejemplo práctico, este programa solicita repetidamente un número entero válido, capturando la excepción `ValueError` hasta que la entrada sea correcta:

```
while True:
    try:
        num = int(input("Ingrese un número entero: "))
        print(f"Se ingresó correctamente: {num}")
        break
    except ValueError:
        print("Error: Debe ingresar un número entero. Intente de nuevo.")
```

Este código utiliza `try`/`except` alrededor de la conversión `int()`. Si la conversión falla, se lanza `ValueError` y el bloque `except` lo captura, evitando que el programa termine abruptamente y permitiendo manejar el error (por ejemplo, mostrando un mensaje) [14]. De esta forma se controla el flujo hasta que el usuario ingresa un valor válido.

5.3 Excepciones personalizadas en Python

Python permite definir **excepciones propias** creando clases que hereden de `Exception` [15]. Por convención se les da nombres terminados en "Error". Por ejemplo:

```
class NumeroNegativoError(Exception):
    """Excepción personalizada para números negativos."""
    pass

def raiz_cuadrada(x):
    if x < 0:
        raise NumeroNegativoError("No se puede calcular la raíz cuadrada de
un número negativo")
    return x ** 0.5

try:
    numero = float(input("Ingrese un número para calcular su raíz cuadrada:
"))
    resultado = raiz_cuadrada(numero)
    print(f"Resultado: {resultado}")
except NumeroNegativoError as e:
    print(f"Error personalizado: {e}")
```

En este código se define la excepción `NumeroNegativoError`. La función `raiz_cuadrada` lanza (`raise`) esta excepción si recibe un número negativo. Luego, al llamar la función, se captura la

excepción personalizada usando `except NumeroNegativoError` y se maneja mostrando un mensaje específico. La sintaxis `raise Excepcion('mensaje')` fuerza la generación de la excepción indicada [16]. Este patrón permite distinguir errores específicos de la aplicación y tratarlos de forma ordenada, mejorando la robustez del programa.

Fuentes: Documentación oficial de Python sobre errores y excepciones [1] [8] [9] [15]. Cada sección citada corresponde al tema pertinente en el tutorial de Python (sintaxis de `try`/`except`, `raise`, excepciones definidas por el usuario, etc.).

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] 8. Errores y excepciones — documentación de Python - 3.14.0
https://docs.python.org/es/3/tutorial/errors.html