* ¿Qué es una excepción en programación y por qué es importante manejarla correctamente?

Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal del mismo, generalmente a causa de un error (por ejemplo, división por cero, acceso a archivo inexistente, etc.).

Manejar las excepciones correctamente es crucial porque:

* Permite que el programa no se detenga abruptamente.
* Facilita una respuesta controlada al error (mostrar un mensaje, reintentar la operación, registrar el fallo, etc.).
* Mejora la robustez, seguridad y experiencia del usuario.
* ¿Cuáles son los tipos de excepciones más comunes?

En Python, algunos tipos comunes de excepciones son:

|  |  |
| --- | --- |
| Excepción | Descripción |
| ZeroDivisionError | División entre cero |
| ValueError | Valor inválido para una operación |
| TypeError | Operación entre tipos incompatibles |
| IndexError | Índice fuera de rango en una lista o tupla |
| KeyError | Clave inexistente en un diccionario |
| FileNotFoundError | Archivo no encontrado |
| AttributeError | Acceso a un atributo que no existe |
| ImportError | Error al importar un módulo |

* ¿Cómo funciona la sentencia try/except y cuándo se debe utilizar?

La estructura try/except permite **atrapar y manejar errores** que puedan ocurrir durante la ejecución del bloque try.

1. try:

2. resultado = 10 / 0

3. except ZeroDivisionError:

4. print("No se puede dividir por cero.")

**Cuándo usarlo:**

* Cuando existe una **posibilidad real de error** en una operación (acceso a archivos, conversiones de datos, conexiones externas).
* Para evitar que el programa se detenga inesperadamente.
* ¿Cómo se pueden capturar múltiples excepciones en un solo bloque de código?

Puedes capturar varias excepciones usando una tupla de excepciones o varios bloques except.

1. try:

2. x = int("abc")

3. except (ValueError, TypeError) as e:

4. print("Error:", e)

O también:

1. try:

2. ...

3. except ValueError:

4. ...

5. except TypeError:

6. ...

* ¿Qué es el uso de raise en Python y cómo se utiliza para generar excepciones en validaciones?

La palabra clave raise se usa para **lanzar manualmente una excepción**, generalmente cuando detectas un error lógico o de validación.

1. def validar\_edad(edad):

2. if edad < 0:

3. raise ValueError("La edad no puede ser negativa.")

Es útil para:

* Aplicar validaciones personalizadas.
* Forzar errores si se incumplen reglas de negocio.
* ¿Cómo se pueden definir excepciones personalizadas y en qué casos sería útil?

Puedes crear excepciones personalizadas heredando de la clase Exception:

1. class SaldoInsuficienteError(Exception):

2. pass

3.

4. def retirar(monto, saldo):

5. if monto > saldo:

6. raise SaldoInsuficienteError("No tienes suficiente saldo.")

Se usan cuando:

* Quieres representar errores específicos de tu aplicación.
* Necesitas diferenciar tipos de fallos lógicos propios de tu dominio (por ejemplo, en un sistema bancario o de reservas).
* ¿Cuál es la función de finally en el manejo de excepciones?

El bloque finally se ejecuta siempre, ocurra o no una excepción. Sirve para realizar tareas de limpieza o cierre de recursos.

1. try:

2. archivo = open("datos.txt")

3. # operaciones

4. except FileNotFoundError:

5. print("Archivo no encontrado.")

6. finally:

7. archivo.close()

* ¿Cuáles son algunas acciones de limpieza que deben ejecutarse después de un proceso que puede generar errores?

Acciones comunes

* Cerrar archivos o conexiones de base de datos.
* Liberar recursos del sistema (memoria, sockets, etc.).
* Eliminar archivos temporales.
* Registrar errores o eventos para auditoría.
* Revertir transacciones si el proceso falló a mitad.