Clase 4: Manejo de Excepciones en Python 🚀

Introducción a las Excepciones 📚

Cuando trabajas con código, es común enfrentar errores. Todos los programadores pasan por esto, y saber cómo manejarlos es crucial para un buen desarrollo de software. Manejar errores de manera efectiva distingue a un programador competente de uno excelente.

¿Qué son las excepciones? ❓

Las excepciones son eventos que alteran el flujo normal de ejecución del programa. En Python, se utilizan para manejar errores y permitir que el programa continúe funcionando, en lugar de cerrarse abruptamente.

Importancia del Manejo de Excepciones 💡

Manejar excepciones permite:

- **Prevenir la interrupción del programa.**
- **Proporcionar retroalimentación al usuario.**
- **Permitir que el programa se recupere de errores y siga funcionando.**

Estructura del Manejo de Excepciones 🛠️

La estructura básica para manejar excepciones en Python utiliza las palabras clave `try`, `except`, y opcionalmente `finally` y `else`.

```
try:
    # Código que puede causar un error
except TipoDeError:
    # Código para manejar el error
```

Uso de `pass` ➡️

La palabra reservada `pass` se utiliza en Python como un marcador de posición. Permite que un bloque de código se ejecute sin hacer nada, lo cual es útil si deseas dejar un bloque vacío.

```
try:
    # Código que puede causar un error
except ValueError:
    pass  # No se hace nada, pero se maneja el error
```

Ejemplo Práctico: División por Cero ⚖️

Vamos a crear un programa que pida al usuario un número divisor y maneje posibles excepciones.

Código Ejemplo:

```
while True:
    try:
        divisor = int(input("Ingresa un número divisor: "))
        resultado = 100 / divisor
        print(f"El resultado es: {resultado}")
        break  # Sale del bucle si todo va bien
    except ZeroDivisionError:
        print("Error: El divisor no puede ser cero. Intenta d
```

```
e nuevo.")
    except ValueError:
        print("Error: Debes introducir un número entero.")
```

Desglose del Código:

1. `while True`: Permite repetir la solicitud hasta que se ingrese un número válido.
2. `try`: Intenta ejecutar el bloque de código.
3. `int(input(...))`: Solicita al usuario un número entero.
4. `resultado = 100 / divisor`: Intenta dividir 100 por el número ingresado.
5. `break`: Sale del bucle si no hay excepciones.
6. `except ZeroDivisionError`: Captura el error si el divisor es cero.
7. `except ValueError`: Captura el error si la entrada no es un número entero.

Manejo de Múltiples Excepciones 🔄

Es buena práctica manejar múltiples excepciones para ofrecer una mejor experiencia al usuario.

Ejemplo Mejorado:

```
while True:
    try:
        divisor = int(input("Ingresa un número divisor: "))
        resultado = 100 / divisor
        print(f"El resultado es: {resultado}")
        break
    except (ZeroDivisionError, ValueError) as e:
        print(f"Error: {e}. Intenta de nuevo.")
```

Ejercicios de Práctica 📝

Ejercicio 1: Ingreso de Edad 👶👧

Pide al usuario que ingrese su edad y maneja excepciones si se ingresa un valor no numérico.

```
while True:
    try:
        edad = int(input("Ingresa tu edad: "))
        print(f"Tienes {edad} años.")
        break
    except ValueError:
        print("Error: Debes introducir un número entero.")
```

Ejercicio 2: Cálculo de Promedio 📊

Pide al usuario que ingrese varias calificaciones y calcula el promedio, manejando excepciones si se ingresan valores no válidos.

```
calificaciones = []
while True:
    entrada = input("Ingresa una calificación (o 'fin' para t
erminar): ")
    if entrada.lower() == 'fin':
        break
    try:
        calificacion = float(entrada)
        calificaciones.append(calificacion)
    except ValueError:
        print("Error: Debes introducir un número.")
if calificaciones:
    promedio = sum(calificaciones) / len(calificaciones)
    print(f"El promedio es: {promedio}")
else:
    print("No se ingresaron calificaciones.")
```

Conclusión 🔚

Manejar excepciones es una habilidad fundamental para cualquier programador. Al entender cómo funcionan las excepciones y cómo manejar errores adecuadamente, puedes crear programas más robustos y amigables para el usuario. Recuerda practicar con ejemplos de la vida real para solidificar tu comprensión.