Nathalia Rojas
Daniel Arrieta
Cecilia Ramos

Paso 1: Conceptualización y Análisis
Cada equipo debe investigar y responder en un documento las siguientes preguntas:

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

Durante la ejecución de un programa, pueden surgir eventos imprevistos denominados excepciones.
(las excepciones no son improvisadas, son programadas para el control y manejo de errores)

Estas interrumpen el flujo normal del programa y pueden ser provocadas por diversas razones, como errores en los datos de entrada o problemas con los archivos. En lugar de causar un fallo abrupto, las excepciones permiten gestionar estos errores de forma controlada, evitando la interrupción inesperada del programa.

*Una **excepción** es un evento que interrumpe el flujo normal de un programa cuando ocurre un error. Es importante manejarlas para evitar que el programa se detenga inesperadamente y para dar **mensajes claros** al usuario o tomar acciones correctivas.*

¿Cuáles son los tipos de excepciones más comunes?

1. *ZeroDivisionError: Ocurre cuando se intenta dividir un número por cero.*
2. *ValueError: Ocurre cuando una operación recibe un valor inapropiado.*
3. *TypeError: Ocurre cuando una operación o función es aplicada a un objeto de tipo incorrecto.*
4. *FileNotFoundError: Ocurre cuando se intenta abrir un archivo que no existe.*
5. *IndexError: Ocurre cuando se accede a un índice que no existe en una lista o tupla.*

¿Cómo funciona la sentencia try/catch y cuándo se debe utilizar? *Nota: Debiese ser try/except ya que try/catch se utiliza para JavaScript*

La sentencia try/except es la base para gestionar errores en Python. En esencia, se coloca el código que podría generar un error dentro del bloque try. Si ocurre un error (una excepción), el programa automáticamente se dirige al bloque except, donde se especifica cómo debe ser manejado ese error. En resumen, try/except permite "intentar" algo y, si falla, "excepto" que se sepa cómo solucionarlo.

ejemplo:

```
try:
    # Bloque de código que puede causar una excepción
    x = 10 / 0
except ZeroDivisionError:
    # Bloque de código que se ejecuta si ocurre la excepción
    print("No puedes dividir por cero")
```

¿Cómo se pueden capturar múltiples excepciones en un solo bloque de código?

En Python, la gestión de errores es flexible y permite capturar múltiples tipos de excepciones dentro de un mismo bloque try. Esto se logra de dos maneras: utilizando múltiples bloques except, cada uno diseñado para un tipo específico de error, o empleando un único bloque except que captura varios tipos de excepciones a través de una tupla. Esta versatilidad permite a los programadores personalizar el manejo de errores, ya sea de forma individual para cada tipo de error o agrupando errores similares para un tratamiento común.

ejemplo:

```
try:
    x = int(input("Ingresa un número: "))
    y = 10 / x
except ValueError:
    print("Debes ingresar un número válido.")
except ZeroDivisionError:
    print("No puedes dividir por cero.")
except Exception as e:
    print(f"Ha ocurrido un error inesperado: {e}")
```

¿Qué es el throw de excepciones y cómo se usa en una validación de datos?

En Python, la herramienta clave para provocar o "lanzar" una excepción es la palabra reservada raise. Se utiliza cuando se quiere generar un error de forma deliberada, como cuando se detecta una situación incorrecta dentro de una función o clase. Es fundamental usar raise apropiadamente para asegurar que el programa maneje los errores de manera efectiva y el flujo del programa se mantenga controlado.

ejemplo:

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

try:
    resultado = dividir(10, 0)
except ZeroDivisionError as e:
    print(e)
```

¿Cómo se pueden definir excepciones personalizadas y en qué casos sería útil?

Podemos hacerlo creando una nueva clase que herede de la clase base Exception. Es útil cuando se necesita representar errores específicos de nuestra aplicación.

ejemplo:

```
class MiErrorPersonalizado(Exception):
    def __init__(self, mensaje):
```

```
        self.mensaje = mensaje
        super().__init__(self.mensaje)

try:
    raise MiErrorPersonalizado("Este es un error personalizado")
except MiErrorPersonalizado as e:
    print(f"Se ha capturado un error: {e}")
```

¿Cuál es la función de finally en el manejo de excepciones?

El bloque finally garantiza que un conjunto de instrucciones se ejecute siempre, sin importar si ocurre una excepción en el bloque try o no. Se usa principalmente para tareas de limpieza, como cerrar archivos o liberar recursos, asegurando que estas acciones se completen, independientemente de si hubo errores durante la ejecución.

ejemplo:

```
try:
    archivo = open('datos.txt', 'r')
    # Código para leer del archivo
except FileNotFoundError:
    print("El archivo no existe.")
finally:
    archivo.close()  # Siempre se ejecuta para asegurar que el archivo se cierre
```

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

Python tiene mecanismos automáticos para limpiar recursos, como cerrar archivos, que se activan al manejar excepciones. Esto, como el uso de with al trabajar con archivos, ayuda a prevenir que el programa entre en un estado inestable, incluso si ocurren errores durante la ejecución.