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

Una excepción es un evento inesperado que ocurre durante la ejecución de un programa y que interrumpe su flujo normal, por ejemplo: intentar dividir por cero, acceder a un archivo inexistente o ingresar datos inválidos.

Manejar excepciones correctamente es fundamental porque:

- Evita que el programa se detenga abruptamente.
- Permite guiar al usuario con mensajes claros.
- Facilita la identificación, registro y solución de errores.
- Permite garantizar que ciertos procesos críticos (como cierre de archivos o conexiones) siempre se completen.

2. ¿Cuáles son los tipos de excepciones más comunes?

Algunas excepciones comunes en Python son:

- ValueError: Se genera cuando un valor tiene un tipo correcto pero un contenido inválido.
- TypeError: Se produce cuando una operación se realiza sobre tipos incompatibles.
- KeyError: Ocurre al intentar acceder a una clave inexistente en un diccionario.
- IndexError: Se lanza al acceder a un índice inexistente en listas o tuplas.
- ZeroDivisionError: Cuando se intenta dividir entre cero.
- FileNotFoundError: Archivo inexistente o ruta incorrecta.
- AttributeError: Cuando un objeto no tiene el atributo solicitado.

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

La sentencia try/except permite “probar” un bloque de código que puede generar un error y “capturarlo” para manejarlo de forma controlada.

Ejemplo básico:

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("No se puede dividir por cero.")
```

Se debe utilizar cuando:

- Existe riesgo de valores inválidos del usuario.
- Se trabaja con archivos, redes o conexiones externas.
- Hay operaciones aritméticas o lógicas susceptibles de error.
- Deseamos capturar y registrar errores sin detener el sistema.

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

Python permite capturar varias excepciones en una sola sentencia except, utilizando una tupla:

```
try:
    proceso()
except (ValueError, TypeError) as e:
    print("Error de tipo o valor:", e)
```

Esto es útil cuando diferentes errores deben recibir una misma respuesta del sistema.

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

La palabra clave raise permite lanzar manualmente una excepción.
Es común utilizarla para validar reglas de negocio.

Ejemplo:

```
if tiempo_reserva <= 0:
    raise ValueError("La duración debe ser mayor a cero.")
```

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

Las excepciones personalizadas se crean definiendo nuevas clases que heredan de `Exception`:

```
class ReservaInvalidaError(Exception):
    pass
```

Son útiles cuando:

- Existe una regla de negocio específica (ej: bicicleta ya reservada).
- Se quiere distinguir claramente errores propios del sistema.
- Se requiere más claridad en los logs y depuración.

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

El bloque finally se ejecuta siempre, ocurra o no una excepción.

Se utiliza para:

- Liberar recursos.
- Cerrar conexiones.
- Registrar logs.

- Limpiar buffers o archivos temporales.

Ejemplo:

```
try:
    archivo = open("registro.txt")
except FileNotFoundError:
    print("Archivo no encontrado.")
finally:
    print("Ejecución finalizada.")
```

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

Ejemplos:

- Cerrar archivos o conexiones a bases de datos.
- Guardar logs del error.
- Restablecer estados internos del sistema.
- Actualizar inventarios y registros.
- Enviar notificaciones de error si corresponde.