

Manejo de errores

Lanzar una excepción (Raise)

raise se usa para lanzar (o generar) excepciones. Esto permite señalar que ocurrió un error o una situación excepcional durante la ejecución del programa.

Ejemplo. Lanzar una excepción cuando se intente dividir por cero

1. Condición para el Denominador:

- Se verifica si el denominador es cero:
- Si el denominador es igual a cero, se levanta una excepción `ValueError` con el mensaje "El denominador no puede ser 0". Esto impide la división por cero y proporciona un mensaje claro del error.

2. Cálculo de la División:

- Si el denominador no es cero, se realiza la división y se devuelve el resultado:

3. Llamada a la Función dividir:

- Se llama a la función `dividir` con `numerador = 10` y `denominador = 4`:
- Como el denominador no es cero, no se levanta ninguna excepción, y la función devuelve el resultado de la división.

```
def dividir(numerador, denominador):  
    if denominador == 0:  
        raise ValueError("El denominador no puede ser 0")  
    return numerador / denominador  
  
resultado = dividir(10, 4)  
print(f"Resultado: {resultado}")
```

Resultado: 2.5

En la siguiente prueba, como se colocó un denominador igual a cero, dará un error:

```
resultado = dividir(10, 0)
print(f"Resultado: {resultado}")
```

ValueError: El denominador no puede ser 0

```
[1;31m-----[0m
[1;31mValueError[0m                                Traceback (most recent call last)
Cell [1;32mIn[21], line 1[0m
[1;32m----> 1[0m resultado [38;5;241m=[39m dividir([38;5;241m10[39m, [38;5;241m0[39m)
[0;32m      2[0m [38;5;28mprint[39m([38;5;124mf [39m[38;5;124m" [39m[38;5;124mResultado: [39m[

Cell [1;32mIn[19], line 3[0m, in [0;36mdividir[1;34m(numerador, denominador)[0m
[0;32m      1[0m [38;5;28;01mdef[39;00m [38;5;21mdividir[39m(numerador, denominador):
[0;32m      2[0m     [38;5;28;01mif [39;00m denominador [38;5;241m==[39m [38;5;241m0[39m:
[1;32m----> 3[0m         [38;5;28;01mraise[39;00m [38;5;167;01mValueError[39;00m([38;5;124m"
[0;32m      4[0m     [38;5;28;01mreturn[39;00m numerador [38;5;241m/[39m denominador

[1;31mValueError[0m: El denominador no puede ser 0
```

Se obtiene el error ValueError: El denominador no puede ser 0 lanzado con raise.

Realizaremos el manejo de excepciones a continuación.

Se realiza lo siguiente:

1. Condición para el Denominador:

- Se verifica si el denominador es cero:
- Si el denominador es igual a cero, se levanta una excepción **ValueError** con el mensaje "El denominador no puede ser 0". Esto previene la división por cero, que no está permitida en matemáticas.

2. Cálculo de la División:

- Si el denominador no es cero, se realiza la división y se devuelve el resultado:

3. Bloque try-except:

- Se utiliza un bloque **try-except** para manejar posibles excepciones al llamar a la función **dividir**:
- En este caso, se intenta dividir 10 entre 0, lo que levanta una excepción **ValueError** porque el denominador es cero.

4. Manejo de la Excepción:

- El bloque `except` captura cualquier excepción que se levante dentro del bloque `try`:
- Como se lanzó una `ValueError`, el mensaje "Error: El denominador no puede ser 0" se imprime en la consola, indicando que se produjo un error debido a un denominador igual a cero.

```
def dividir(numerador, denominador):
    if denominador == 0:
        raise ValueError("El denominador no puede ser 0")
    return numerador / denominador

try:
    resultado = dividir(10, 0)
    print(f"Resultado: {resultado}")
except Exception as e:
    print(f"Error: {e}")
```

Error: El denominador no puede ser 0

Código para obtener todas las excepciones posibles

1. Importación del Módulo `builtins`:

- El módulo `builtins` en Python contiene funciones, excepciones y otros objetos que están siempre disponibles.

2. Filtrar las Excepciones:

- Se utiliza una lista por comprensión para filtrar todas las excepciones disponibles en el módulo `builtins`.
- Se verifica que cada objeto en `builtins` sea una clase (`type`) y que sea una subclase de `BaseException`:

3. Impresión de la Lista de Excepciones:

- Se imprime la lista de excepciones predefinidas en Python:

```
# Código para obtener todas las excepciones posibles
import builtins

# Filtrar las excepciones en el módulo builtins
excepciones = [exc for exc in dir(builtins) if
                isinstance(getattr(builtins, exc), type)
                and issubclass(getattr(builtins, exc),
```

```
BaseException)]
```

```
# Imprimir la lista de excepciones  
print(excepciones)
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExceptionGroup
```

Otra forma de obtener una lista de excepciones es consultar la documentación oficial en:

<https://docs.python.org/3/library/exceptions.html>

donde se proporciona una lista completa y detallada de todas las excepciones integradas.