Manejo de Errores en Python

Los errores son una parte inevitable de la programación, independientemente de tu nivel de experiencia. Tanto si eres un principiante como un desarrollador experimentado, te encontrarás con situaciones en las que tu código no se ejecutará como esperas. Para estos casos, Python ofrece un mecanismo robusto llamado **manejo de errores** o **manejo de excepciones**, que permite detectar y gestionar situaciones problemáticas de forma controlada.

En este cuaderno, exploraremos en profundidad el mundo del manejo de errores en Python. Abordaremos los diferentes tipos de errores que pueden surgir, cómo detectarlos y gestionarlos de manera elegante, y las mejores prácticas para escribir código confiable, robusto y fácil de mantener.

Al final de este cuaderno, habrás adquirido un entendimiento sólido de:

- **Tipos comunes de errores en Python.**
- **Cómo utilizar bloques `try-except` para capturar y manejar excepciones de manera efectiva.**
- **El papel de las cláusulas `else` y `finally` en el manejo de excepciones.**
- **Cómo personalizar mensajes de error y crear excepciones personalizadas.**
- **Mejores prácticas para escribir código resistente a errores.**

¿Por qué es importante el manejo de errores?

El manejo adecuado de errores es crucial para asegurar que nuestras aplicaciones puedan lidiar con situaciones inesperadas sin fallar abruptamente. Un programa que no maneja excepciones de forma adecuada puede terminar en errores fatales que afecten la experiencia del usuario o, peor aún, puedan comprometer la seguridad y la estabilidad del sistema.

El manejo de errores también es clave para la **mantenibilidad** y **escalabilidad** del código. Con buenas prácticas de manejo de excepciones, tu código será más fácil de depurar y más resistente a futuros cambios o mejoras.

Tipos de errores comunes en Python

En Python, existen dos categorías principales de errores:

1. **Errores de sintaxis**: Estos ocurren cuando hay algún problema con la estructura del código. Python no podrá interpretar el código y lo notificará antes de intentar ejecutarlo.
   - Ejemplo: Un paréntesis no cerrado o un `if` mal formulado.

```
if True
    print("Hola")
# Error de sintaxis: falta el ":" en la línea del if.
```

2. **Excepciones**: Estos son errores que ocurren durante la ejecución del programa. Aunque el código tenga una sintaxis correcta, puede fallar por circunstancias inesperadas como intentar dividir entre cero o acceder a un índice fuera de rango.
   - Ejemplo: Dividir un número por cero.

```
x = 10 / 0
# Excepción: ZeroDivisionError
```

Uso del bloque `try-except` para manejar excepciones

El bloque `try-except` es el mecanismo principal de Python para manejar excepciones. Este permite "intentar" ejecutar un bloque de código, y si ocurre una excepción, "capturarla" y gestionarla de manera adecuada.

Estructura básica:

```python try: # Código que podría generar una excepción resultado = 10 / 0 except ZeroDivisionError: # Código que se ejecuta si ocurre una excepción del tipo ZeroDivisionError print("Error: No se puede dividir por cero.")

# Introducción

En el mundo de la programación, los errores son una ocurrencia común. Incluso un pequeño error, como un punto y coma faltante o un error tipográfico, puede llevar a problemas inesperados en tu código. En Python, estos errores se dividen en dos categorías amplias: **errores de sintaxis** y **errores en tiempo de ejecución**.
```

Errores de Sintaxis

Los **errores de sintaxis** ocurren cuando tu código viola las reglas del lenguaje Python. Estos son fáciles de identificar porque el intérprete de Python te dirige directamente a la línea problemática. Un ejemplo típico sería olvidarse de incluir un símbolo crucial, como los dos puntos (`:`) en una sentencia `while`.

```
# Ejemplo de error de sintaxis: falta el ":" en el bloque while.
while True print("Hola mundo!")  #Esto dará un SyntaxError.
```

```
  Cell In[1], line 2
    while True print("Hola mundo!")  #Esto dará un SyntaxError.
               ^
SyntaxError: invalid syntax
```

Cuando Python detecta un error de sintaxis, muestra un mensaje indicando la línea donde ocurrió el problema. Este tipo de error debe corregirse antes de que el programa pueda ejecutarse.

Errores en Tiempo de Ejecución

Los **errores en tiempo de ejecución** son más difíciles de prever. Ocurren cuando el programa ya está en ejecución y suelen estar relacionados con problemas lógicos o entradas no previstas. Estos errores no se detectan hasta que el código se ejecuta.

```
# Ejemplo de error en tiempo de ejecución: división por cero.
10 / 0  # Esto provocará un ZeroDivisionError.
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[2], line 2
      1 # Ejemplo de error en tiempo de ejecución: división por cero.
----> 2 10 / 0  # Esto provocará un ZeroDivisionError.

ZeroDivisionError: division by zero
```

```
# Ejemplo de error en tiempo de ejecución: acceso a una variable no definida.
print(x)  # Si 'x' no ha sido definida, causará un NameError.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 # Ejemplo de error en tiempo de ejecución: acceso a una variable no definida.
```

```
----> 2 print(x)  # Si 'x' no ha sido definida, causará un NameError.

NameError: name 'x' is not defined
```

Estos errores requieren una estrategia para ser manejados de manera adecuada, evitando que el programa se bloquee abruptamente.

Manejo de Errores en Python

Python permite manejar estos errores mediante bloques `try-except`. El bloque `try` contiene el código que puede generar un error, mientras que el bloque `except` captura y maneja el error si ocurre.

```
### Uso básico de `try-except`

# Ejemplo de manejo de errores con try-except.
try:
     resultado = 10 / 0
except ZeroDivisionError:
    print("Error: No se puede dividir por cero.")  # Captura el error
de división por cero.

Error: No se puede dividir por cero.
```

En este ejemplo, cuando ocurre una división por cero, el bloque `except` captura el error y evita que el programa termine abruptamente.

Múltiples excepciones

Es posible manejar diferentes tipos de errores utilizando múltiples bloques `except`, de modo que puedas personalizar la respuesta según el tipo de excepción.

```
# Ejemplo con múltiples excepciones.
try:
    x = int(input("Introduce un número: "))
    resultado = 10 / x
except ValueError:
    print("Error: Debes introducir un número válido.")
except ZeroDivisionError:
    print("Error: No se puede dividir por cero.")

Introduce un número:  0

Error: No se puede dividir por cero.
```

Este código captura tanto errores de tipo `ValueError` (si el usuario no ingresa un número) como errores de `ZeroDivisionError` (si el usuario ingresa un 0).

Cláusulas `else` y `finally`

Además de `try` y `except`, Python incluye dos cláusulas opcionales que pueden utilizarse para manejar errores de forma más precisa: `else` y `finally`.

Cláusula `else`

El bloque `else` se ejecuta solo si no ocurre ninguna excepción. Es útil cuando quieres ejecutar código solo si todo salió bien.

```
# Ejemplo de `else`.
try:
    resultado = 10 / 2
except ZeroDivisionError:
    print("Error: No se puede dividir por cero.")
else:
    print("El resultado es:", resultado)  # Se ejecuta si no hay
error.
```

```
El resultado es: 5.0
```

Cláusula `finally`

El bloque `finally` se ejecuta siempre, ocurra o no una excepción. Es útil para liberar recursos o realizar acciones de limpieza, como cerrar archivos abiertos.

```
archivo = open('datos.txt', 'r')
```

```
---------------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[10], line 1
----> 1 archivo = open('datos.txt', 'r')

File
~/.local/lib/python3.10/site-packages/IPython/core/interactiveshell.py
:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what
you are doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'datos.txt'
```

```python
# Ejemplo de `finally`.
try:
    archivo = open('datos.txt', 'r')
    contenido = archivo.read()
except FileNotFoundError:
    print("Error: El archivo no fue encontrado.")
finally:
    try:
        archivo.close()  # Esto se ejecuta siempre, independientemente de si hubo un error o no.
        print("Archivo cerrado.")
    except NameError:
        print("Error: El archivo no fue definido")
```

```
Error: El archivo no fue encontrado.
Error: El archivo no fue definido
```

Creación de Excepciones Personalizadas

A veces, los errores predefinidos de Python no son suficientes para manejar casos específicos en tu programa. En esos casos, puedes crear excepciones personalizadas utilizando la palabra clave `raise`.

```python
# Ejemplo de excepción personalizada.
def verificar_edad(edad):
    if edad < 18:
        raise ValueError("La edad debe ser al menos 18.")
    return True

try:
    verificar_edad(15)
except ValueError as e:
    print("Error:", e)
```

En este ejemplo, si la edad es menor de 18, se genera un `ValueError` personalizado.

El manejo de errores es una habilidad clave en la programación. Conocer cómo manejar errores correctamente no solo hace que tu código sea más robusto y confiable, sino que también mejora la experiencia del usuario al proporcionar mensajes de error claros y evitar que el programa falle inesperadamente. Utiliza los bloques `try-except`, `else`, `finally` y excepciones personalizadas para asegurar que tu código sea resistente a errores.

Fixing Syntax Errors

Los errores de sintaxis son fáciles de detectar porque Python te indica la línea exacta donde se encuentran y describe lo que salió mal. Un ejemplo típico es cuando olvidas colocar los dos puntos después de un `True` en una estructura `while`.

```
#Ejemplo de error de sintaxis:
while True print("Hola mundo!")  # SyntaxError: falta ':' en la línea
del while.
```

Ruta del error: Las líneas anteriores al mensaje final muestran la secuencia de ejecución antes del error. Descripción del error: La última línea muestra el tipo de error (en este caso, `SyntaxError`).

A veces, el problema puede estar en una línea anterior a la señalada. Si no encuentras el error en la línea indicada, revisa las instrucciones anteriores para identificar la causa.

```
#Ejemplo adicional de error de sintaxis:
for i in range(5)  # SyntaxError: falta ':' al final del bucle.
    print(i)
```

```
  Cell In[14], line 2
    for i in range(5)  # SyntaxError: falta ':' al final del bucle.
                       ^
SyntaxError: expected ':'
```

```
if x == 10  # SyntaxError: falta ':' al final de la condición.
    print("x es 10")
```

```
  Cell In[15], line 1
    if x == 10  # SyntaxError: falta ':' al final de la condición.
                ^
SyntaxError: expected ':'
```

Excepciones / Errores en Tiempo de Ejecución

Una excepción es un error que ocurre en tiempo de ejecución. Aunque el código es sintácticamente correcto, algo falla mientras el programa se ejecuta. Este tipo de errores se llaman "errores en tiempo de ejecución".

Python tiene varios tipos de excepciones, y el mensaje de error suele incluir el tipo de excepción junto con detalles sobre lo que la causó, lo que te ayuda a identificar el problema rápidamente.

Ejemplos de excepciones comunes:

ZeroDivisionError: se lanza cuando intentas dividir un número por cero.

```
# Ejemplo de ZeroDivisionError:
10 / 0  # Esto provocará una excepción: ZeroDivisionError.
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call
last)
```

```
Cell In[16], line 2
      1 # Ejemplo de ZeroDivisionError:
----> 2 10 / 0  # Esto provocará una excepción: ZeroDivisionError.

ZeroDivisionError: division by zero
```

```
# Ejemplo adicional de ZeroDivisionError:
def divide(a, b):
    return a / b
divide(10, 0)  # Esto provocará una excepción: ZeroDivisionError.
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call
last)
Cell In[17], line 4
      2 def divide(a, b):
      3     return a / b
----> 4 divide(10, 0)  # Esto provocará una excepción:
ZeroDivisionError.

Cell In[17], line 3, in divide(a, b)
      2 def divide(a, b):
----> 3     return a / b

ZeroDivisionError: division by zero
```

NameError: ocurre cuando intentas acceder a una variable que no ha sido definida.

```
# Ejemplo de NameError:
print(xx)  # Esto provocará una excepción: NameError, porque 'xx' no
ha sido definido.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
Cell In[18], line 2
      1 # Ejemplo de NameError:
----> 2 print(xx)  # Esto provocará una excepción: NameError, porque
'xx' no ha sido definido.

NameError: name 'xx' is not defined
```

```
# Ejemplo adicional de NameError:
def print_name():
    print(name)  # Provocará una excepción: NameError, porque 'name'
no ha sido definido.
print_name()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 4
      2 def print_name():
      3     print(name)  # Provocará una excepción: NameError, porque 'name' no ha sido definido.
----> 4 print_name()

Cell In[19], line 3, in print_name()
      2 def print_name():
----> 3     print(name)

NameError: name 'name' is not defined
```

TypeError: ocurre cuando intentas realizar una operación con tipos de datos incompatibles.

```
# Ejemplo de TypeError:
"Hello" + 2  # Esto provocará una excepción: TypeError, no se puede concatenar una cadena con un número.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 2
      1 # Ejemplo de TypeError:
----> 2 "Hello" + 2  # Esto provocará una excepción: TypeError, no se puede concatenar una cadena con un número.

TypeError: can only concatenate str (not "int") to str
```

```
# Ejemplo adicional de TypeError:
sum_value = "Total: " + 100  # Provocará una excepción: TypeError, no se puede concatenar una cadena con un entero.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 2
      1 # Ejemplo adicional de TypeError:
----> 2 sum_value = "Total: " + 100  # Provocará una excepción: TypeError, no se puede concatenar una cadena con un entero.

TypeError: can only concatenate str (not "int") to str
```

Excepciones Base y Concretas

Python organiza las excepciones en una jerarquía, dividida en excepciones base (más generales) y excepciones más específicas. Las excepciones base te permiten manejar varios tipos de errores en conjunto, mientras que las excepciones específicas proporcionan un manejo más detallado.

⚠ Puedes consultar la documentación oficial de excepciones de Python aquí: https://docs.python.org/3/library/exceptions.html ⚠

A continuación veremos algunos tipos comunes de excepciones:

LookupError

Esta es la clase base para excepciones que ocurren cuando una clave o índice utilizado no es válido. Incluye errores como IndexError y KeyError.

```
#Ejemplo de KeyError:
dict_ = {"name": "Albert", "age": 30}
dict_["city"]  # Provocará una excepción: KeyError, porque la clave
'city' no existe.
```

```
---------------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
Cell In[22], line 3
      1 #Ejemplo de KeyError:
      2 dict_ = {"name": "Albert", "age": 30}
----> 3 dict_["city"]  # Provocará una excepción: KeyError, porque la
clave 'city' no existe.

KeyError: 'city'
```

```
#Ejemplo adicional de KeyError:
dict_ = {"apple": 1, "banana": 2}
print(dict_["orange"])  # Provocará una excepción: KeyError, porque la
clave 'orange' no existe.
```

```
---------------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
Cell In[23], line 3
      1 #Ejemplo adicional de KeyError:
      2 dict_ = {"apple": 1, "banana": 2}
----> 3 print(dict_["orange"])  # Provocará una excepción: KeyError,
porque la clave 'orange' no existe.

KeyError: 'orange'
```

```
#Ejemplo de IndexError:
lst_ = [1, 2, 3, 54]
lst_[5]  # Provocará una excepción: IndexError, porque el índice 5
está fuera del rango de la lista.
```

```
---------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
Cell In[24], line 3
      1 #Ejemplo de IndexError:
      2 lst_ = [1, 2, 3, 54]
----> 3 lst_[5]  # Provocará una excepción: IndexError, porque el
índice 5 está fuera del rango de la lista.

IndexError: list index out of range
```

```
#Ejemplo adicional de IndexError:
lst_ = [10, 20, 30]
print(lst_[3])  # Provocará una excepción: IndexError, porque el
índice 3 está fuera del rango de la lista.
```

```
---------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
Cell In[25], line 3
      1 #Ejemplo adicional de IndexError:
      2 lst_ = [10, 20, 30]
----> 3 print(lst_[3])  # Provocará una excepción: IndexError, porque
el índice 3 está fuera del rango de la lista.

IndexError: list index out of range
```

ImportError

Ocurre cuando Python no puede cargar un módulo o encuentra problemas al intentar hacerlo.

```
# Ejemplo de ImportError:
import maz  # Provocará una excepción: ModuleNotFoundError, porque el
módulo 'maz' no existe.
```

```
---------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[26], line 2
      1 # Ejemplo de ImportError:
----> 2 import maz  # Provocará una excepción: ModuleNotFoundError,
porque el módulo 'maz' no existe.
```

```
ModuleNotFoundError: No module named 'maz'

# Ejemplo adicional de ImportError:
import non_existent_module  # Provocará una excepción:
ModuleNotFoundError, porque el módulo no existe.

---------------------------------------------------------------------
-----
ModuleNotFoundError                         Traceback (most recent call
last)
Cell In[27], line 2
      1 # Ejemplo adicional de ImportError:
----> 2 import non_existent_module  # Provocará una excepción:
ModuleNotFoundError, porque el módulo no existe.

ModuleNotFoundError: No module named 'non_existent_module'
```

AttributeError

Ocurre cuando un objeto no tiene el atributo o método que intentas usar.

```
#Ejemplo de AttributeError:
num = 20
num.upper()  # Provocará una excepción: AttributeError, porque los
enteros no tienen el método 'upper'.

---------------------------------------------------------------------
-----
AttributeError                              Traceback (most recent call
last)
Cell In[28], line 3
      1 #Ejemplo de AttributeError:
      2 num = 20
----> 3 num.upper()  # Provocará una excepción: AttributeError, porque
los enteros no tienen el método 'upper'.

AttributeError: 'int' object has no attribute 'upper'

#Ejemplo adicional de AttributeError:
class MyClass:
    def __init__(self):
        self.value = 10
my_obj = MyClass()
my_obj.non_existent_method()  # Provocará una excepción:
AttributeError, porque 'non_existent_method' no existe.

---------------------------------------------------------------------
-----
AttributeError                              Traceback (most recent call
```

```
last)
Cell In[29], line 6
      4         self.value = 10
      5 my_obj = MyClass()
----> 6 my_obj.non_existent_method()  # Provocará una excepción:
AttributeError, porque 'non_existent_method' no existe.

AttributeError: 'MyClass' object has no attribute
'non_existent_method'
```

ValueError

Ocurre cuando una función recibe un argumento con el tipo correcto pero un valor inválido.

```
#Ejemplo de ValueError:
lst_ = [1, 2, 3, 4, 5, 6]
lst_.remove(10)  # Provocará una excepción: ValueError, porque el
valor 10 no está en la lista.

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[30], line 3
      1 #Ejemplo de ValueError:
      2 lst_ = [1, 2, 3, 4, 5, 6]
----> 3 lst_.remove(10)  # Provocará una excepción: ValueError, porque
el valor 10 no está en la lista.

ValueError: list.remove(x): x not in list

#Ejemplo adicional de ValueError:
int("Hello")  # Provocará una excepción: ValueError, porque no se
puede convertir la cadena "Hello" a entero.

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[31], line 2
      1 #Ejemplo adicional de ValueError:
----> 2 int("Hello")  # Provocará una excepción: ValueError, porque no
se puede convertir la cadena "Hello" a entero.

ValueError: invalid literal for int() with base 10: 'Hello'
```

TypeError

Ocurre cuando intentas realizar una operación entre tipos de datos no compatibles.

```
# Ejemplo de TypeError:
"Hello" + 2  # Provocará una excepción: TypeError, porque no se puede concatenar una cadena con un número.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[32], line 2
      1 # Ejemplo de TypeError:
----> 2 "Hello" + 2  # Provocará una excepción: TypeError, porque no se puede concatenar una cadena con un número.

TypeError: can only concatenate str (not "int") to str
```

```
# Ejemplo adicional de TypeError:
"5" * "2"  # Provocará una excepción: TypeError, porque no se puede multiplicar una cadena por otra cadena.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 2
      1 # Ejemplo adicional de TypeError:
----> 2 "5" * "2"  # Provocará una excepción: TypeError, porque no se puede multiplicar una cadena por otra cadena.

TypeError: can't multiply sequence by non-int of type 'str'
```

```
import os
```

```
!pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /home/ubuntu/.local/lib/python3.10/site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/ubuntu/.local/lib/python3.10/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: numpy>=1.21.0 in /home/ubuntu/.local/lib/python3.10/site-packages (from pandas) (1.26.4)
Requirement already satisfied: tzdata>=2022.1 in /home/ubuntu/.local/lib/python3.10/site-packages (from pandas) (2024.1)
Requirement already satisfied: pytz>=2020.1 in /home/ubuntu/.local/lib/python3.10/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in
```

```
/home/ubuntu/.local/lib/python3.10/site-packages (from python-
dateutil>=2.8.2->pandas) (1.16.0)
```

OSError

Ocurre cuando hay problemas relacionados con el sistema operativo, como archivos no encontrados o fallos en la operación de entrada/salida.

```
!pip install ipywidgets
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Input In [25], in <cell line: 1>()
----> 1 get_ipython().system('pip install ipywidgets')

File /lib/python3.9/site-packages/IPython/core/interactiveshell.py:2451, in InteractiveShell.system_piped(self, cmd)
   2446     raise OSError("Background processes not supported.")
   2448 # we explicitly do NOT return the subprocess status code, because
   2449 # a non-None value would trigger :func:`sys.displayhook` calls.
   2450 # Instead, we store the exit_code in user_ns.
-> 2451 self.user_ns['_exit_code'] = system(self.var_expand(cmd, depth=1))

File /lib/python3.9/site-packages/IPython/utils/_process_posix.py:148, in ProcessHandler.system(self, cmd)
    146     child = pexpect.spawnb(self.sh, args=['-c', cmd]) # Pexpect-U
    147 else:
--> 148     child = pexpect.spawn(self.sh, args=['-c', cmd])  # Vanilla Pexpect
    149 flush = sys.stdout.flush
    150 while True:
    151     # res is the index of the pattern that caused the match, so we
    152     # know whether we've finished (if we matched EOF) or not

File /lib/python3.9/site-packages/IPython/utils/_process_posix.py:57, in ProcessHandler.sh(self)
     55     self._sh = pexpect.which(shell_name)
     56     if self._sh is None:
---> 57         raise OSError('"{}" shell not found'.format(shell_name))
     59 return self._sh

OSError: "sh" shell not found
```

```
# Ejemplo de OSError:
import pandas as pd
df = pd.read_csv("archivo_inexistente.csv")  # Provocará una
excepción: FileNotFoundError, porque el archivo no existe.
```

```
---------------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[36], line 3
      1 # Ejemplo de OSError:
      2 import pandas as pd
----> 3 df = pd.read_csv("archivo_inexistente.csv")  # Provocará una
excepción: FileNotFoundError, porque el archivo no existe.

File
~/.local/lib/python3.10/site-packages/pandas/io/parsers/readers.py:912
```

```
, in read_csv(filepath_or_buffer, sep, delimiter, header, names,
index_col, usecols, dtype, engine, converters, true_values,
false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
parse_dates, infer_datetime_format, keep_date_col, date_parser,
date_format, dayfirst, cache_dates, iterator, chunksize, compression,
thousands, decimal, lineterminator, quotechar, quoting, doublequote,
escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision,
storage_options, dtype_backend)
    899 kwds_defaults = _refine_defaults_read(
    900     dialect,
    901     delimiter,
   (...)
    908     dtype_backend=dtype_backend,
    909 )
    910 kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File
~/.local/lib/python3.10/site-packages/pandas/io/parsers/readers.py:577
, in _read(filepath_or_buffer, kwds)
    574 _validate_names(kwds.get("names", None))
    576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
    579 if chunksize or iterator:
    580     return parser

File
~/.local/lib/python3.10/site-packages/pandas/io/parsers/readers.py:140
7, in TextFileReader.__init__(self, f, engine, **kwds)
   1404     self.options["has_index_names"] = kwds["has_index_names"]
   1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File
~/.local/lib/python3.10/site-packages/pandas/io/parsers/readers.py:166
1, in TextFileReader._make_engine(self, f, engine)
   1659     if "b" not in mode:
   1660         mode += "b"
-> 1661 self.handles = get_handle(
   1662     f,
   1663     mode,
   1664     encoding=self.options.get("encoding", None),
   1665     compression=self.options.get("compression", None),
   1666     memory_map=self.options.get("memory_map", False),
   1667     is_text=is_text,
   1668     errors=self.options.get("encoding_errors", "strict"),
   1669     storage_options=self.options.get("storage_options", None),
   1670 )
```

```
   1671 assert self.handles is not None
   1672 f = self.handles.handle

File ~/.local/lib/python3.10/site-packages/pandas/io/common.py:859, in
get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    854 elif isinstance(handle, str):
    855     # Check whether the filename is to be opened in binary
mode.
    856     # Binary mode does not support 'encoding' and 'newline'.
    857     if ioargs.encoding and "b" not in ioargs.mode:
    858         # Encoding
--> 859         handle = open(
    860             handle,
    861             ioargs.mode,
    862             encoding=ioargs.encoding,
    863             errors=errors,
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
    868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory:
'archivo_inexistente.csv'
```

```
# Ejemplo adicional de OSError:
with open("archivo_inexistente.txt") as f:  # Provocará una excepción:
FileNotFoundError, porque el archivo no existe.
    content = f.read()
```

```
---------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[37], line 2
      1 # Ejemplo adicional de OSError:
----> 2 with open("archivo_inexistente.txt") as f:  # Provocará una
excepción: FileNotFoundError, porque el archivo no existe.
      3     content = f.read()

File
~/.local/lib/python3.10/site-packages/IPython/core/interactiveshell.py
:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what
you are doing, "
    321         "you can use builtins' open."
```

```
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory:
'archivo_inexistente.txt'
```

TimeOutError

Ocurre cuando se supera un tiempo de espera. Es común verlo en operaciones que involucran redes o servidores. El `TimeOutError` generalmente se encuentra en contextos de red o de llamadas a APIs donde el tiempo de respuesta excede un límite preestablecido. En un entorno práctico, este tipo de error puede ser manejado utilizando bibliotecas que permiten establecer tiempos de espera, como `requests` en Python

```python
# Ejemplo de TimeOutError:

# Importamos la biblioteca requests para realizar solicitudes HTTP.
import requests

# Definimos una función para hacer una solicitud a una URL con un
tiempo de espera.
def fetch_data(url):
    # Hacemos una solicitud GET con un tiempo de espera de 2 segundos.
    response = requests.get(url, timeout=2)
    return response.json()  # Retornamos la respuesta en formato JSON.

# Llamamos a la función con una URL que sabemos que no responde
rápidamente.
fetch_data("https://httpbin.org/delay/5")  # Esta URL espera 5
segundos antes de responder.
```

```
---------------------------------------------------------------------
-----
TimeoutError                              Traceback (most recent call
last)
File
~/.local/lib/python3.10/site-packages/urllib3/connectionpool.py:537,
in HTTPConnectionPool._make_request(self, conn, method, url, body,
headers, retries, timeout, chunked, response_conn, preload_content,
decode_content, enforce_content_length)
    536 try:
--> 537     response = conn.getresponse()
    538 except (BaseSSLError, OSError) as e:

File ~/.local/lib/python3.10/site-packages/urllib3/connection.py:466,
in HTTPConnection.getresponse(self)
    465 # Get the response from http.client.HTTPConnection
--> 466 httplib_response = super().getresponse()
    468 try:
```

```
File /usr/lib/python3.10/http/client.py:1375, in
HTTPConnection.getresponse(self)
   1374 try:
-> 1375     response.begin()
   1376 except ConnectionError:

File /usr/lib/python3.10/http/client.py:318, in
HTTPResponse.begin(self)
    317 while True:
--> 318     version, status, reason = self._read_status()
    319     if status != CONTINUE:

File /usr/lib/python3.10/http/client.py:279, in
HTTPResponse._read_status(self)
    278 def _read_status(self):
--> 279     line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
    280     if len(line) > _MAXLINE:

File /usr/lib/python3.10/socket.py:705, in SocketIO.readinto(self, b)
    704 try:
--> 705     return self._sock.recv_into(b)
    706 except timeout:

File /usr/lib/python3.10/ssl.py:1303, in SSLSocket.recv_into(self,
buffer, nbytes, flags)
   1300         raise ValueError(
   1301           "non-zero flags not allowed in calls to recv_into()
on %s" %
   1302           self.__class__)
-> 1303     return self.read(nbytes, buffer)
   1304 else:

File /usr/lib/python3.10/ssl.py:1159, in SSLSocket.read(self, len,
buffer)
   1158 if buffer is not None:
-> 1159     return self._sslobj.read(len, buffer)
   1160 else:

TimeoutError: The read operation timed out

The above exception was the direct cause of the following exception:

ReadTimeoutError                            Traceback (most recent call
last)
File ~/.local/lib/python3.10/site-packages/requests/adapters.py:486,
in HTTPAdapter.send(self, request, stream, timeout, verify, cert,
proxies)
    485 try:
--> 486     resp = conn.urlopen(
```

```
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:

File
~/.local/lib/python3.10/site-packages/urllib3/connectionpool.py:847,
in HTTPConnectionPool.urlopen(self, method, url, body, headers,
retries, redirect, assert_same_host, timeout, pool_timeout,
release_conn, chunked, body_pos, preload_content, decode_content,
**response_kw)
    845     new_e = ProtocolError("Connection aborted.", new_e)
--> 847 retries = retries.increment(
    848     method, url, error=new_e, _pool=self,
_stacktrace=sys.exc_info()[2]
    849 )
    850 retries.sleep()

File ~/.local/lib/python3.10/site-packages/urllib3/util/retry.py:470,
in Retry.increment(self, method, url, response, error, _pool,
_stacktrace)
    469 if read is False or method is None or not
self._is_method_retryable(method):
--> 470     raise reraise(type(error), error, _stacktrace)
    471 elif read is not None:

File ~/.local/lib/python3.10/site-packages/urllib3/util/util.py:39, in
reraise(tp, value, tb)
     38         raise value.with_traceback(tb)
---> 39     raise value
     40 finally:

File
~/.local/lib/python3.10/site-packages/urllib3/connectionpool.py:793,
in HTTPConnectionPool.urlopen(self, method, url, body, headers,
retries, redirect, assert_same_host, timeout, pool_timeout,
release_conn, chunked, body_pos, preload_content, decode_content,
**response_kw)
    792 # Make the request on the HTTPConnection object
--> 793 response = self._make_request(
    794     conn,
```

```
    795     method,
    796     url,
    797     timeout=timeout_obj,
    798     body=body,
    799     headers=headers,
    800     chunked=chunked,
    801     retries=retries,
    802     response_conn=response_conn,
    803     preload_content=preload_content,
    804     decode_content=decode_content,
    805     **response_kw,
    806 )
    808 # Everything went great!

File
~/.local/lib/python3.10/site-packages/urllib3/connectionpool.py:539,
in HTTPConnectionPool._make_request(self, conn, method, url, body,
headers, retries, timeout, chunked, response_conn, preload_content,
decode_content, enforce_content_length)
    538 except (BaseSSLError, OSError) as e:
--> 539     self._raise_timeout(err=e, url=url,
timeout_value=read_timeout)
    540     raise

File
~/.local/lib/python3.10/site-packages/urllib3/connectionpool.py:370,
in HTTPConnectionPool._raise_timeout(self, err, url, timeout_value)
    369 if isinstance(err, SocketTimeout):
--> 370     raise ReadTimeoutError(
    371         self, url, f"Read timed out. (read
timeout={timeout_value})"
    372     ) from err
    374 # See the above comment about EAGAIN in Python 3.

ReadTimeoutError: HTTPSConnectionPool(host='httpbin.org', port=443):
Read timed out. (read timeout=2)

During handling of the above exception, another exception occurred:

ReadTimeout                               Traceback (most recent call
last)
Cell In[38], line 13
     10     return response.json()  # Retornamos la respuesta en
formato JSON.
     12 # Llamamos a la función con una URL que sabemos que no
responde rápidamente.
---> 13 fetch_data("https://httpbin.org/delay/5")  # Esta URL espera 5
segundos antes de responder.

Cell In[38], line 9, in fetch_data(url)
```

```
      7 def fetch_data(url):
      8     # Hacemos una solicitud GET con un tiempo de espera de 2
segundos.
----> 9     response = requests.get(url, timeout=2)
     10     return response.json()

File ~/.local/lib/python3.10/site-packages/requests/api.py:73, in
get(url, params, **kwargs)
     62 def get(url, params=None, **kwargs):
     63     r"""Sends a GET request.
     64
     65     :param url: URL for the new :class:`Request` object.
   (...)
     70     :rtype: requests.Response
     71     """
---> 73     return request("get", url, params=params, **kwargs)

File ~/.local/lib/python3.10/site-packages/requests/api.py:59, in
request(method, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is
closed, thus we
     56 # avoid leaving sockets open which can trigger a
ResourceWarning in some
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)

File ~/.local/lib/python3.10/site-packages/requests/sessions.py:589,
in Session.request(self, method, url, params, data, headers, cookies,
files, auth, timeout, allow_redirects, proxies, hooks, stream, verify,
cert, json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File ~/.local/lib/python3.10/site-packages/requests/sessions.py:703,
in Session.send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File ~/.local/lib/python3.10/site-packages/requests/adapters.py:532,
in HTTPAdapter.send(self, request, stream, timeout, verify, cert,
proxies)
```

```
    530     raise SSLError(e, request=request)
    531 elif isinstance(e, ReadTimeoutError):
--> 532     raise ReadTimeout(e, request=request)
    533 elif isinstance(e, _InvalidHeader):
    534     raise InvalidHeader(e, request=request)

ReadTimeout: HTTPSConnectionPool(host='httpbin.org', port=443): Read
timed out. (read timeout=2)
```

```python
# Ejemplo adicional de TimeOutError:

import socket

# Establece un tiempo de espera de 1 segundo
socket.setdefaulttimeout(1)

# Intenta conectar a un puerto que no responderá
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.0.2.1", 80))  # Esta IP es un ejemplo y no debería
responder
```

```
---------------------------------------------------------------------
-----
TimeoutError                              Traceback (most recent call
last)
Cell In[39], line 10
      8 # Intenta conectar a un puerto que no responderá
      9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
---> 10 s.connect(("192.0.2.1", 80))  # Esta IP es un ejemplo y no
debería responder

TimeoutError: timed out
```

Manejo de Excepciones

El manejo de excepciones es una técnica de programación para controlar errores que ocurren durante la ejecución de una aplicación. Se manejan de manera similar a una declaración condicional. Si no ocurre una excepción (general o específica), que sería el caso normal, la aplicación continúa con las siguientes instrucciones y, si ocurre una, se ejecutarán las instrucciones indicadas por el desarrollador para su tratamiento, lo que puede continuar la aplicación o detenerla, dependiendo del caso.

El manejo de excepciones en Python comienza con una estructura de palabras clave, así:

```python
try:
    # instrucciones que podrían generar una excepción
    instructions
except:
    # instrucciones a ejecutar si ocurre una excepción
    instructions if that exception occurs
```

- Los errores nos proporcionan información.
- Los errores detienen el código.

Esto puede suceder porque no anticipamos el problema o porque anticipamos el problema y queremos mantener el código en funcionamiento.

En este manejo de excepciones, el tipo de la excepción puede ser genérico o específico. La mejor manera de controlar y manejar lo sucedido es hacerlo de la forma más específica posible, escribiendo múltiples cláusulas except para el mismo try, cada una manejando un tipo diferente de excepción.

```
try:
    # instrucciones que podrían generar una excepción
    instructions
except ExceptionType1:
    # instrucciones a ejecutar si ocurre ExceptionType1
    instructions if that exception occurs 1
except ExceptionType2:
    # instrucciones a ejecutar si ocurre ExceptionType2
    instructions if that exception occurs 2
```

```
  Cell In[40], line 6
    instructions if that exception occurs 1
    ^
SyntaxError: expected 'else' after 'if' expression
```

```
# Ejemplo de lista con un elemento no numérico:
lst_ = [1, 2, 3, 4, 5, "6", 7, 8, 9]

# Intentamos elevar al cuadrado cada elemento de la lista.
squared_list = []

# Intentamos agregar el cuadrado de cada elemento a la lista.
for i in lst_:
    try:
        squared_list.append(i**2)  # Intenta elevar al cuadrado
    except TypeError:
        # Manejo del error si el elemento no es un número
        print(f"El elemento {i} no se puede elevar al cuadrado, es de
tipo {type(i)}")

print("Lista de cuadrados:", squared_list)  # Muestra la lista con los
cuadrados.
```

```
El elemento 6 no se puede elevar al cuadrado, es de tipo <class 'str'>
Lista de cuadrados: [1, 4, 9, 16, 25, 49, 64, 81]
```

```
# Ejemplo mejorado: lista para almacenar excepciones.
lst_ = [1, 2, 3, 4, 5, "6", 7, 8, 9]  # Lista original con un elemento
no numérico.
```

```python
# Inicializa listas vacías para almacenar resultados y excepciones.
squared_list = []  # Lista para almacenar los cuadrados de los elementos.
exceptions_list = []  # Lista para almacenar elementos que causan errores.

# Iteramos sobre cada elemento de la lista original.
for i in lst_:
    try:
        squared_list.append(i**2)  # Intenta elevar al cuadrado el elemento.
    except TypeError:
        # Si ocurre un TypeError, se almacena el elemento que causó el error.
        exceptions_list.append(i)
        print(f"El elemento {i} no se puede elevar al cuadrado, es de tipo {type(i)}")  # Mensaje de error.

# Muestra la lista con los cuadrados y la lista de excepciones.
print("Lista de cuadrados:", squared_list)  # Muestra la lista con los cuadrados.
print("Lista de excepciones:", exceptions_list)  # Muestra la lista de excepciones.
```

```
El elemento 6 no se puede elevar al cuadrado, es de tipo <class 'str'>
Lista de cuadrados: [1, 4, 9, 16, 25, 49, 64, 81]
Lista de excepciones: ['6']
```

```python
# Definición de la función para elevar al cuadrado y manejar excepciones.
def elements_powered(lst_):
    # Inicializa listas vacías para almacenar resultados y excepciones.
    squared_list = []  # Lista para almacenar los cuadrados de los elementos.
    exceptions_list = []  # Lista para almacenar elementos que causan errores.

    # Iteramos sobre cada elemento de la lista proporcionada.
    for i in lst_:
        try:
            squared_list.append(i**2)  # Intenta elevar al cuadrado el elemento.
        except TypeError:
            # Si ocurre un TypeError, se almacena el elemento que causó el error.
            exceptions_list.append(i)
            print(f"El elemento {i} no se puede elevar al cuadrado, es de tipo {type(i)}")  # Mensaje de error.
```

```
    return squared_list, exceptions_list  # Devuelve ambas listas.

# Uso de la función
powered, exceptions = elements_powered([1, 2, 3, 4, 5, "6", 7, 8, 9,
"10", [0]])

# Muestra los resultados
print("Lista de cuadrados:", powered)  # Muestra la lista con los
cuadrados.
print("Lista de excepciones:", exceptions)  # Muestra la lista de
excepciones.
```

```
El elemento 6 no se puede elevar al cuadrado, es de tipo <class 'str'>
El elemento 10 no se puede elevar al cuadrado, es de tipo <class
'str'>
El elemento [0] no se puede elevar al cuadrado, es de tipo <class
'list'>
Lista de cuadrados: [1, 4, 9, 16, 25, 49, 64, 81]
Lista de excepciones: ['6', '10', [0]]
```

Lo que no se debe hacer al manejar excepciones

⚠: PARA EVITAR

```
# Manejo ineficiente de excepciones:
try:
    # Código que podría generar una excepción.
    pass
except:
    # Manejo muy general sin especificar el tipo de excepción.
    pass

# Ejemplo de lista con un elemento no numérico:
lst_ = [1, 2, 3, 4, 5, "a string"]  # Lista que incluye un string.

# Intentamos elevar al cuadrado cada elemento de la lista.
new_list = []  # Inicializamos una lista vacía para almacenar los
cuadrados.

# Iteramos sobre un rango de 20 para intentar acceder a cada elemento
de lst_.
for i in range(20):
    try:
        new_list.append(lst_[i]**2)  # Intenta elevar al cuadrado el
elemento en la posición i.
    except IndexError:
        # Manejo del error si se intenta acceder a un índice fuera del
rango de la lista.
        print(f"La longitud de la lista es menor que {i}")  # Mensaje
```

```
que indica el problema con el índice.
    except TypeError:
        # Manejo del error si el tipo de dato no permite la operación
(por ejemplo, elevar al cuadrado un string).
        print(f"El tipo de {lst_[i]} es {type(lst_[i])}, no se puede
elevar al cuadrado.")  # Mensaje que indica el tipo de dato
problemático.

# Muestra la lista con los cuadrados obtenidos.
print("Lista de cuadrados:", new_list)  # Imprime la lista con los
resultados.
```

```
El tipo de a string es <class 'str'>, no se puede elevar al cuadrado.
La longitud de la lista es menor que 6
La longitud de la lista es menor que 7
La longitud de la lista es menor que 8
La longitud de la lista es menor que 9
La longitud de la lista es menor que 10
La longitud de la lista es menor que 11
La longitud de la lista es menor que 12
La longitud de la lista es menor que 13
La longitud de la lista es menor que 14
La longitud de la lista es menor que 15
La longitud de la lista es menor que 16
La longitud de la lista es menor que 17
La longitud de la lista es menor que 18
La longitud de la lista es menor que 19
Lista de cuadrados: [1, 4, 9, 16, 25]
```

```
# Definición de función para elevar al cuadrado con manejo de
excepciones.
def squaring_numbers(lst_):
    new_list = []  # Inicializamos una lista vacía para almacenar los
cuadrados.

    # Iteramos hasta 20 para intentar acceder a cada elemento de lst_.
    for i in range(20):
        try:
            # Intenta elevar al cuadrado el elemento en la posición i.
            new_list.append(lst_[i]**2)
        except IndexError:
            # Manejo del error si se intenta acceder a un índice fuera
del rango de la lista.
            print(f"La longitud de la lista es menor que {i}")
        except TypeError as e:
            # Manejo del error si el tipo de dato no permite la
operación.
            print(f"El tipo de {lst_[i]} es {type(lst_[i])}: error
{e}")
```

```
    return new_list  # Devuelve la lista de cuadrados

# Lista de ejemplo que se va a utilizar.
lst_ = [1, 2, 3, 4, 5, "a string"]  # Lista con un elemento no
numérico.

# Uso de la función
resultados = squaring_numbers(lst_)  # Llamada a la función con la
lista de ejemplo.

# Muestra los resultados obtenidos.
print("Lista de cuadrados:", resultados)  # Imprime la lista de
cuadrados.
```

```
El tipo de a string es <class 'str'>: error unsupported operand
type(s) for ** or pow(): 'str' and 'int'
La longitud de la lista es menor que 6
La longitud de la lista es menor que 7
La longitud de la lista es menor que 8
La longitud de la lista es menor que 9
La longitud de la lista es menor que 10
La longitud de la lista es menor que 11
La longitud de la lista es menor que 12
La longitud de la lista es menor que 13
La longitud de la lista es menor que 14
La longitud de la lista es menor que 15
La longitud de la lista es menor que 16
La longitud de la lista es menor que 17
La longitud de la lista es menor que 18
La longitud de la lista es menor que 19
Lista de cuadrados: [1, 4, 9, 16, 25]
```

Desarrollo Guiado por Pruebas (Test Driven Development)

1. **Primero, escribe la prueba.**
2. **Escribe el código.**
3. **Asegúrate de que el código pase la prueba que escribiste previamente.**

Python proporciona una manera de establecer condiciones exigibles, es decir, condiciones que un objeto debe cumplir; de lo contrario, se lanzará una excepción. Esto actúa como una "red de seguridad" contra posibles errores del programador.

La declaración `assert` se utiliza en Python para verificar que una condición específica se cumpla en un punto determinado del código. Si la condición es `True`, el programa continúa su ejecución normalmente. Si es `False`, se lanza una excepción llamada `AssertionError`, que puede ser capturada y manejada. Esto permite que los desarrolladores incluyan una forma de validación que ayude a identificar errores o comportamientos no deseados en el código.

La forma de llamar a esta expresión es la siguiente:

```python assert boolean_condition # Si la condición es falsa, se lanza un AssertionError.
```

```
# Ejemplo de aserción:
assert 10 == 10  # Esto no detendrá el código porque la afirmación es
verdadera.

# Ejemplo de aserciones que fallan:
assert 10 == 11, "No son iguales"  # Esto lanzará un AssertionError.
assert 10 == 11, "Harold no murió :)"  # Esto lanzará un
AssertionError.
```

```
---------------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
Cell In[48], line 2
      1 # Ejemplo de aserciones que fallan:
----> 2 assert 10 == 11, "No son iguales"  # Esto lanzará un
AssertionError.
      3 assert 10 == 11, "Harold no murió :)"  # Esto lanzará un
AssertionError.

AssertionError: No son iguales
```

Ejemplo 1: Validación de Entradas en una Función

```
def calcular_media(numeros):
    assert len(numeros) > 0, "La lista no puede estar vacía"  #
Verifica que la lista no esté vacía
    assert all(isinstance(num, (int, float)) for num in numeros),
"Todos los elementos deben ser números"  # Verifica que todos los
elementos sean números

    return sum(numeros) / len(numeros)  # Devuelve la media

# Prueba de la función
try:
    print(calcular_media([10, 20, 30]))  # Esto debería funcionar
    print(calcular_media([]))  # Esto provocará un AssertionError
except AssertionError as e:
    print("Error:", e)  # Manejo del error
```

```
20.0
Error: La lista no puede estar vacía
```

Ejemplo 2: Comprobación de Invariantes en una Clase

```
class Pila:
    def __init__(self):
        self.items = []

    def apilar(self, item):
```

```
        self.items.append(item)  # Añade un elemento a la pila

    def desapilar(self):
        assert not self.esta_vacia(), "No se puede desapilar de una
pila vacía"  # Verifica que la pila no esté vacía
        return self.items.pop()  # Devuelve el último elemento
agregado

    def esta_vacia(self):
        return len(self.items) == 0  # Verifica si la pila está vacía

# Uso de la clase Pila
p = Pila()
p.apilar(1)
p.apilar(2)
print(p.desapilar())  # Esto debería funcionar
print(p.desapilar())  # Esto debería funcionar
try:
    print(p.desapilar())  # Esto provocará un AssertionError
except AssertionError as e:
    print("Error:", e)  # Manejo del error
```

```
2
1
Error: No se puede desapilar de una pila vacía
```

Ejemplo 3: Verificación de Condiciones en Funciones Matemáticas

```
import math

def calcular_raiz_cuadrada(x):
    assert x >= 0, "El número no puede ser negativo"  # Verifica que x
sea no negativo

    return math.sqrt(x)  # Devuelve la raíz cuadrada de x

# Prueba de la función
try:
    print(calcular_raiz_cuadrada(25))  # Esto debería funcionar
    print(calcular_raiz_cuadrada(-4))  # Esto provocará un
AssertionError
except AssertionError as e:
    print("Error:", e)  # Manejo del error
```

```
5.0
Error: El número no puede ser negativo
```

Lanzar Errores

El lanzamiento de errores (o excepciones) es una técnica que permite a los programadores gestionar situaciones inesperadas en su código. Cuando una función encuentra un valor o una situación que no puede manejar, se puede lanzar una excepción para indicar que algo ha salido mal. Esto permite que el programa se detenga de manera controlada o que el error sea gestionado por otra parte del código.

```
# Definición de función que multiplica un número por 2.
def multiply_by_2(x):
    # Verifica si el tipo de x es un entero.
    if type(x) == int:
        return x * 2  # Retorna el valor multiplicado por 2.
    else:
        # Lanza una excepción si el tipo no es correcto.
        raise ValueError("Input must be an integer")

# Uso de la función con un valor válido.
resultado = multiply_by_2(10)  # Esto funcionará.
print("Resultado:", resultado)  # Imprime el resultado: 20

# Intento de uso con un tipo incorrecto.
try:
    multiply_by_2("10")  # Esto lanzará una excepción.
except ValueError as e:
    # Manejo del error, imprime el mensaje de la excepción.
    print("Error:", e)  # Imprime: Error: Input must be an integer
```

```
Resultado: 20
Error: Input must be an integer
```

```
# Ejemplo mejorado que permite convertir cadenas

# Definición de función que multiplica un número por 2, permitiendo
cadenas numéricas.
def multiply_by_2(x):
    # Verifica si x es un entero.
    if type(x) == int:
        return x * 2  # Retorna el valor multiplicado por 2.

    # Verifica si x es una cadena.
    elif type(x) == str:
        try:
            # Intenta convertir la cadena a entero y multiplicar por
2.
            return int(x) * 2
        except ValueError:
            # Lanza excepción si la conversión falla.
            raise ValueError("La cadena no puede ser convertida a un
entero")
```

```
    else:
        # Lanza excepción si x no es un entero ni una cadena que
represente un entero.
        raise ValueError("El argumento debe ser un entero o una cadena
que represente un entero")

# Intento de uso con un tipo incorrecto
try:
    resultado = multiply_by_2([10])  # Esto lanzará una excepción.
except ValueError as e:
    print("Error:", e)  # Manejo del error para ValueError.
```

```
Error: El argumento debe ser un entero o una cadena que represente un
entero
```

```
# Llamadas adicionales para demostrar el manejo de excepciones
try:
    print(multiply_by_2(10))      # Esto funcionará.
    print(multiply_by_2("10"))    # Esto también funcionará.
    print(multiply_by_2("abc"))    # Esto lanzará una excepción.
except ValueError as e:
    print("Error:", e)  # Manejo del error para ValueError.
```

```
20
20
Error: La cadena no puede ser convertida a un entero
```

```
# Definición de una excepción personalizada.
class MyCoolException(Exception):
    def __init__(self, string):
        self.value = string

    def __str__(self):
        return f"Se produjo un error aquí: {self.value}"

# Función que usa la excepción personalizada.
def multiply_by_2(x):
    if type(x) == int:
        return x * 2  # Retorna el valor multiplicado por 2.
    else:
        raise MyCoolException("La entrada debe ser un entero")  #
Lanza una excepción personalizada si el tipo no es correcto.

# Intento de uso con un tipo incorrecto
try:
    result = multiply_by_2("567")  # Esto lanzará la excepción
personalizada.
except MyCoolException as e:
    print(e)  # Manejo del error personalizado.
```

```
Se produjo un error aquí: La entrada debe ser un entero
```

```python
# Llamada con un tipo correcto
try:
    print(multiply_by_2(10))  # Esto funcionará y mostrará 20.
except MyCoolException as e:
    print(e)  # No se ejecutará porque no hay error.
```

```
20
```

Ejercicios para practicar

1. **División con Manejo de Excepciones:** Escribe una función que tome dos números y realice la división. Utiliza `try` y `except` para manejar la excepción `ZeroDivisionError` si el segundo número es cero. Si hay un error, imprime un mensaje de error y devuelve `None`.

2. **Validación de Entrada:** Crea una función que solicite al usuario que ingrese un número entero. Utiliza `try` y `except` para manejar la excepción `ValueError` si la entrada no es un número entero. Asegúrate de que el programa siga pidiendo un número hasta que se reciba uno válido.

3. **Uso de `finally`:** Escribe una función que abra un archivo para lectura. Utiliza `try`, `except`, y `finally` para asegurar que el archivo se cierre correctamente, independientemente de si ocurre un error o no.

4. **Uso de `else` en Manejo de Excepciones:** Crea una función que calcule la raíz cuadrada de un número. Utiliza `try`, `except`, y `else` para manejar posibles errores de tipo (debería ser un número positivo).

5. **Afirmaciones (Assertions):** Escribe una función que reciba una lista de números y devuelva el número mayor. Usa `assert` para asegurarte de que la lista no esté vacía y maneja el error si se lanza una excepción.

6. **Uso de `raise` para Excepciones Personalizadas:** Crea una excepción personalizada llamada `NegativeNumberError`. Escribe una función que verifique si un número es negativo y, si lo es, lanza esta excepción.

7. **Conversión de Cadenas a Números:** Escribe una función que convierta una cadena a un entero. Utiliza `try` y `except` para manejar `ValueError` si la conversión falla. Si la conversión es exitosa, imprime el número convertido.

8. **Lectura de Datos de un Archivo CSV:** Crea una función que lea un archivo CSV y maneje errores de archivo usando `try`, `except` y `finally`.

9. **Validación de Correo Electrónico:** Escribe una función que valide un correo electrónico. Usa `assert` para verificar que contenga un `@` y un `.`. Lanza una excepción si no es válido.

10. **Calculo de Promedio con Manejo de Excepciones:** Crea una función que calcule el promedio de una lista de números. Usa `try` y `except` para manejar el caso en que la lista esté vacía. Si la lista no tiene elementos, lanza una excepción y maneja el error.

Soluciones

```
# 1. División con Manejo de Excepciones:
def division(numerator, denominator):
    try:
        result = numerator / denominator  # Intenta realizar la
división.
    except ZeroDivisionError:
        print("Error: No se puede dividir por cero.")  # Maneja la
excepción si el denominador es cero.
        return None
    return result  # Retorna el resultado si no hay excepción.

# Uso de la función
print(division(10, 2))
print(division(10, 0))
```

```
5.0
Error: No se puede dividir por cero.
None
```

```
# 2. Validación de Entrada:
def obtener_entero():
    while True:
        try:
            numero = int(input("Introduce un número entero: "))  #
Intenta convertir la entrada a entero.
            return numero  # Retorna el número si la conversión es
exitosa.
        except ValueError:
            print("Error: Debes introducir un número entero válido.")
# Manejo del error si no es un entero.

# Uso de la función
numero = obtener_entero()
```

```
Introduce un número entero:  x

Error: Debes introducir un número entero válido.

Introduce un número entero:  20
```

```
# 3. Uso de `finally`:
def leer_archivo(filename):
    try:
        file = open(filename, 'r')  # Intenta abrir el archivo.
```

```
        content = file.read()  # Lee el contenido del archivo.
        return content
    except FileNotFoundError:
        print("Error: Archivo no encontrado.")  # Maneja la excepción
si el archivo no existe.
        return ""  # Retorna una cadena vacía si ocurre un error.
    finally:
        try:
            file.close()  # Siempre cierra el archivo, incluso si hay
un error.
        except:
            print("Error: No existe archivo para cerrar")

# Uso de la función
print(leer_archivo("archivo.txt"))
```

```
Error: Archivo no encontrado.
Error: No existe archivo para cerrar
```

```
# 4. Uso de `else` en Manejo de Excepciones:
import math

def raiz_cuadrada(numero):
    try:
        if numero < 0:
            raise ValueError("El número no puede ser negativo.")  #
Lanza una excepción si el número es negativo.
        resultado = math.sqrt(numero)  # Calcula la raíz cuadrada.
    except ValueError as e:
        print(e)  # Maneja el error si se lanza una excepción.
        return ""
    else:
        return resultado  # Retorna el resultado si no hay excepción.

# Uso de la función
print(raiz_cuadrada(16))
print(raiz_cuadrada(-4))
```

```
4.0
El número no puede ser negativo.
```

```
# 6. Uso de `raise` para Excepciones Personalizadas:
class NegativeNumberError(Exception):
    pass

def verificar_numero(num):
    if num < 0:
        raise NegativeNumberError("El número no puede ser negativo.")
# Lanza una excepción personalizada.
```

```
    return num  # Retorna el número si no es negativo.

# Uso de la función
try:
    verificar_numero(-5)  # Esto lanzará la excepción personalizada.
except NegativeNumberError as e:
    print(e)  # Manejo del error personalizado.
```

```
El número no puede ser negativo.
```

```
# 7. Conversión de Cadenas a Números:
def convertir_a_entero(cadena):
    try:
        numero = int(cadena)  # Intenta convertir la cadena a entero.
        print(f"Número convertido: {numero}")  # Muestra el número
convertido.
    except ValueError:
        print("Error: La cadena no se puede convertir a un entero.")
# Manejo del error de conversión.

# Uso de la función
convertir_a_entero("123")
convertir_a_entero("abc")
```

```
Número convertido: 123
Error: La cadena no se puede convertir a un entero.
```

```
# 8. Lectura de Datos de un Archivo CSV:
import csv

def leer_csv(filename):
    try:
        with open(filename, 'r') as file:  # Intenta abrir el archivo
CSV.
            reader = csv.reader(file)  # Crea un lector CSV.
            for row in reader:
                print(row)  # Imprime cada fila del archivo.
    except FileNotFoundError:
        print("Error: Archivo CSV no encontrado.")  # Maneja el error
si el archivo no existe.
    finally:
        print("Intento de lectura del archivo completada.")  # Mensaje
final al completar la ejecución

# Uso de la función
leer_csv("datos.csv")
```

```
Error: Archivo CSV no encontrado.
Intento de lectura del archivo completada.
```

```python
# 9. Validación de Correo Electrónico:
def validar_correo(correo):
    # Asegura que el correo tenga formato correcto.
    assert ('@' in correo and '.' in correo), "El correo electrónico
no es válido."
    print("El correo electrónico es válido.")  # Mensaje de éxito si
no hay excepción.

# Uso de la función con un correo válido
try:
    validar_correo("ejemplo@dominio.com")
except AssertionError as e:
    print(e)

try:
    validar_correo("ejemplo")
except AssertionError as e:
    print(e)
```

```
El correo electrónico es válido.
El correo electrónico no es válido.
```

```python
# 10. Cálculo de Promedio con Manejo de Excepciones:
def calcular_promedio(lista):
    try:
        if len(lista) == 0:
            raise ValueError("La lista no puede estar vacía para
calcular el promedio.")  # Lanza excepción si la lista está vacía.
        promedio = sum(lista) / len(lista)  # Calcula el promedio.
        return promedio  # Retorna el promedio.
    except ValueError as e:
        print(e)  # Manejo del error si se lanza una excepción.
        return ""

# Uso de la función
print(calcular_promedio([1, 2, 3, 4]))
print(calcular_promedio([]))
```

```
2.5
La lista no puede estar vacía para calcular el promedio.
```