Tema 7:

Manejo de errores e introducción a excepciones (3 h)

7. Manejo de errores e introducción a excepciones

7.1. Errores comunes en Python (sintaxis, tipos, ejecución)

7.2. Lectura e interpretación de tracebacks

7.3. Uso básico de try/except

7.4. Parte práctica (ejemplos y ejercicios)

7.5. Distribución sugerida del tiempo

Profesor: Salvador Martínez Bolinches

Centro: IES Font de Sant Lluís

Año: 2025

Objetivos de aprendizaje

- Comprender la diferencia entre error de sintaxis y excepción en tiempo de ejecución.
- Utilizar bloques `try–except` para manejar errores.
- Emplear `else` y `finally` en la gestión de excepciones.
- Conocer excepciones comunes en Python.
- Aprender a lanzar excepciones con raise.

7.1. Errores comunes en Python

En programación, los errores pueden clasificarse en varias categorías:

1. Errores de sintaxis

Falta el `:` en la instrucción `if`. 👉

```
# Error de sintaxis
if True
    print("Hola")
```

- Ocurren cuando el código no respeta las reglas del lenguaje.
- Son detectados antes de ejecutar el programa.

```
# Error de tipo (TypeError)
numero = "5" + 3
```

2. Errores de tiempo de ejecución (excepciones)

- El programa se ejecuta, pero falla en una línea concreta.

```
python

# Error de índice (IndexError)
lista = [1, 2, 3]
print(lista[5])
```

3. Errores lógicos

- El programa funciona pero el resultado no es el esperado.
- Ejemplo: calcular mal un promedio por error de fórmula.

Veamos un ejemplo de error sintáctico:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

Una excepción o un error de ejecución se produce durante la ejecución del programa. Las excepciones se puede manejar para que no termine el programa. Veamos algunos ejemplos de excepciones:

```
>>> 4/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> a+4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined

>>> "2"+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Hemos obtenido varias excpciones: ZeroDivisionError, NameError y TypeError. Puedes ver la **lista de excepciones** y su significado.

7.2. Lectura e interpretación de *tracebacks*

En el proceso de desarrollo de programas en Python, es habitual que surjan errores en tiempo de ejecución. Cuando esto ocurre, el intérprete muestra en pantalla un mensaje denominado *traceback*. Este elemento constituye una herramienta esencial para la depuración, ya que contiene información detallada sobre el origen del problema. Un *traceback* se compone de tres partes principales:

1. **La línea de código donde ocurrió el error**, indicando el archivo y la posición exacta.
2. **El tipo de error**, que corresponde a la excepción generada (por ejemplo, `IndexError`, `TypeError`, `KeyError`).
3. **Un mensaje explicativo**, que orienta sobre la causa del fallo.

Ejemplo:

```
lista = [1, 2, 3]
print(lista[5])
```

Salida:

```
Traceback (most recent call last):
  File "ejemplo.py", line 2, in <module>
    print(lista[5])
IndexError: list index out of range
```

El análisis de este *traceback* permite identificar que el error se encuentra en la línea 2 del archivo `ejemplo.py`, donde se intenta acceder a un índice inexistente en la lista. El tipo de excepción es `IndexError` y el mensaje “list index out of range” aclara que el acceso excede los límites de la estructura.

La correcta lectura de un *traceback* es fundamental para todo programador, pues posibilita comprender con rapidez la naturaleza del error y aplicar la solución adecuada. Así, se convierte en una competencia clave dentro de la práctica de la programación y en un paso imprescindible en la formación de pensamiento computacional.

👉 En síntesis, saber leer un *traceback* no solo permite localizar errores, sino también fortalecer la capacidad de depuración, optimizando la eficiencia en el desarrollo de software.

7.3. Uso básico de `try/except`

La sintaxis general es:

```
try:
    # Código que puede fallar
    x = int("abc")
except ValueError:
    # Código que se ejecuta si ocurre el error
    print("Conversión inválida")
```

Capturar múltiples excepciones

```
try:
    lista = [1, 2, 3]
    print(lista[5])
except IndexError:
    print("Índice fuera de rango")
except Exception as e:
    print("Error inesperado:", e)
```

Bloques `else` y `finally`

```
try:
    numero = int(input("Introduce un número: "))
except ValueError:
    print("Debes introducir un número válido")
else:
    print("Número correcto:", numero)
finally:
    print("Fin del programa")
```

- `else` → se ejecuta si no hubo error.
- `finally` → se ejecuta siempre, haya o no error.

Veamos un ejemplo simple como podemos tratar una excepción:

```
>>> while True:
...     try:
...         x = int(input("Introduce un número:"))
...         break
...     except ValueError:
...         print ("Debes introducir un número")
```

1. Se ejecuta el bloque de instrucciones de `try`.
2. Si no se produce la excepción, el bloque de `except` no se ejecuta y continúa la ejecución secuencia.
3. Si se produce una excepción, el resto del bloque `try` no se ejecuta, si la excepción que se ha produce corresponde con la indicada en `except` se salta a ejecutar el bloque de instrucciones `except`.
4. Si la excepción producida no se corresponde con las indicadas en `except` se pasa a otra instrucción `try`, si finalmente no hay un manejador nos dará un error y el programa terminará.

Un bloque `except` puede manejar varios tipos de excepciones:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Si quiero controlar varios tipos de excepciones puedo poner varios bloques `except`. Teniendo en cuenta que en el último, si quiero no indico el tipo de excepción (o un ***else***):

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... except:
...   print("Otro error")
```

Se puede utilizar también la clausula `else`:

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... else:
...   print("Otro error")
```

Por último indicar que podemos indicar una clausula `finally` para indicar un bloque de instrucciones que siempre se debe ejecutar, independientemente de la excepción se haya producido o no.

```
>>> try:
...     result = x / y
... except ZeroDivisionError:
...     print("División por cero!")
... else:
...     print("El resultado es", result)
... finally:
...     print("Terminamos el programa")
```

Obteniendo información de las excepciones

Las excepciones son realmente **objetos** de la clase ***except***. Podemos usar los métodos definidos para estos objetos.

```
>>> cad = "a"
>>> try:
...   i = int(cad)
... except ValueError as error:
...   print(type(error))
...   print(error.args)
...   print(error)
...
<class 'ValueError'>
("invalid literal for int() with base 10: 'a'",)
invalid literal for int() with base 10: 'a'
```

Propagando excepciones. raise

Si construimos una función donde se maneje una excepción podemos hacer que la excepción se envía a la función desde la que la hemos llamado. Para ello utilizamos la instrucción `raise`.

Veamos algunos ejemplos:

```
def dividir(x,y):
    try:
        return x/y
    except ZeroDivisionError:
        raise
```

Con `raise` también podemos propagar una excepción en concreto:

```
def nivel(numero):
    if numero<0:
        raise ValueError("El número debe ser positivo:"+str(numero))
    else:
        return numero
```

7.4. Parte práctica (ejemplos y ejercicios)

Ejemplo 1: Conversión segura de entrada

```
try:
    edad = int(input("Introduce tu edad: "))
    print("El año que viene tendrás", edad + 1)
except ValueError:
    print("Por favor introduce un número entero válido.")
```

Ejemplo 2: Manejo de archivos

```
try:
    f = open("archivo.txt", "r")
    contenido = f.read()
    print(contenido)
except FileNotFoundError:
    print("El archivo no existe")
finally:
    print("Ejecución terminada")
```

Ejercicios propuestos

1. Escribe un programa que pida un número al usuario. Si introduce texto, capturar la excepción y mostrar un mensaje.
2. Crea una lista de 5 elementos. Pide al usuario un índice e intenta mostrar el valor correspondiente. Captura `IndexError` si se introduce un índice inválido.
3. Pide al usuario dos números y divídelos. Captura la excepción `ZeroDivisionError` si el divisor es cero.
4. Escribe un programa que:
    - Pida dos números al usuario.
    - Los convierta a enteros.
    - Divida el primero entre el segundo.
    - Maneje `ValueError` y `ZeroDivisionError` adecuadamente.
    - Use `else` para mostrar el resultado cuando todo va bien.
    - Use `finally` para mostrar un mensaje de despedida.
5. Captura y maneja la excepción si intentas acceder a una posición inexistente en una lista.
6. Crea un programa que pida un número entero y, si la conversión es correcta, muestre `"Conversión exitosa"` usando `else`.
7. Escribe un programa que pida la edad y lance un `ValueError` si la edad es negativa.
8. Programa una calculadora que realice suma, resta, multiplicación y división, controlando errores de entrada y divisiones entre cero.
9. Escribe un programa que pida una contraseña y lance un error si tiene menos de 6 caracteres.
10. Escribe un ejemplo de `try` que capture tanto `ValueError` como `ZeroDivisionError.`

7.5. Distribución sugerida del tiempo (3h)

- **Teoría (1 h)** → tipos de errores, tracebacks, try/except, else/finally.
- **Ejemplos guiados ()** → conversión de entradas, archivos, manejo de excepciones comunes.
- **Ejercicios prácticos (2 h)** → resolución individual y en grupos.