Manejo de Errores

1. Tracebacks

Un traceback es el cuerpo del texto que puede apuntar al origen (y al final) de un error no controlado.

Si intentamos en un notebook, abrir un archivo inexistente sucede lo siguiente:

```python
#Abriendo un codigo inexistente
#stdin:entrada en el terminal interactivo (Cell 2)
#Error:"FileNotFoundError"

open("/path/to/mars.jpg")
```

Python

Esa salida tiene varias partes clave. En primer lugar, el traceback menciona el orden de la salida. Después, informa de que el archivo es 'stdin' (entrada en el terminal interactivo) en la primera línea de la entrada. El error es 'FileNotFoundError' (el nombre de excepción), lo que significa que el archivo no existe o quizás el directorio correspondiente no existe.

```python
#Abriendo un codigo inexistente
#stdin:entrada en el terminal interactivo (Cell 2)
#Error:"FileNotFoundError"

open("/path/to/mars.jpg")
```

Python

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Untitled-1.ipynb Cell 2' in <module>
----> 1 open("/path/to/mars.jpg")

FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
```

Crea un archivo de Python y asígnale el nombre *open.py*, con el contenido siguiente:

```python
def main():
    open("/path/to/mars.jpg")

if __name__ == '__main__':
    main()
```

Python

Se trata de una sola función 'main()' que abre el archivo inexistente, como antes. Al final, esta función usa un asistente de Python que indica al intérprete que ejecute la función 'main()' cuando se le llama en el terminal.

Ejecútala con Python y podrás comprobar el siguiente mensaje de error:

```python
#Abriendo un codigo inexistente
#main: abre el archivo inexistente como antes
#El error se inicia en la inea 8 que incluye la llamada a main()
#Acontinuacion la salida sigue el error a la linea 5 en la llamada de funcion open()
#FileNotFoundError notifica que el archivo o el directorio no existen.
def main():
    open("/path/to/mars.jpg")

if __name__ == '__main__':
    main()
```

Python

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Untitled-1.ipynb Cell 3' in <module>
      5     open("/path/to/mars.jpg")
      7 if __name__ == '__main__':
----> 8     main()

Untitled-1.ipynb Cell 3' in main()
      4 def main():
----> 5     open("/path/to/mars.jpg")

FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
```

Los tracebacks casi siempre incluyen la información siguiente:

- Todas las rutas de acceso de archivo implicadas, para cada llamada a cada función.
- Los números de línea asociados a cada ruta de acceso de archivo.
- Los nombres de las funciones, métodos o clases implicados en la generación de una excepción.
- El nombre de la excepción que se ha producido.

2. Controlando las excepciones

Try y Except de los bloques

Los archivos de configuración pueden tener todo tipo de problemas, por lo que es fundamental notificarlos con precisión cuando se presenten. Sabemos que, si no existe un archivo o directorio, se genera 'FileNotFoundError'. Si queremos controlar esa excepción, podemos hacerlo con un bloque try y except:

- Try y Except de los bloques

```python
#Controlando una exepcion con "try" and "except"
#El bloque "try" y "except",junto con un mensaje útil, evita un seguimiento y sigue informan
def main():
    try:
        configuration = open('config.txt')
    except FileNotFoundError:
        print("Couldn't find the config.txt file!")

if __name__ == '__main__':
    main()
```

[3] Python

```
Couldn't find the config.txt file!
```

Sugerencia:

Sugerencia

```python
try:
    open("mars.jpg")
except FileNotFoundError as err:
    print("got a problem trying to read the file:", err)
```

[32] ✓ 0.1s Python

```python
try:
    open("mars.jpg")
except FileNotFoundError as err:
    print("got a problem trying to read the file:", err)
```

[32] ✓ 0.1s Python

```
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
```

3. Generación de excepciones

La generación de excepciones también puede ayudar en la toma de decisiones para otro código. Como hemos visto antes, en función del error, el código puede tomar decisiones inteligentes para resolver, solucionar o ignorar un problema.

Los astronautas limitan su uso de agua a unos 11 litros al día. Vamos a crear una función que, con base al número de astronautas, pueda calcular la cantidad de agua quedará después de un día o más:

```python
def water_left(astronauts, water_left, days_left):
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    return f"Total water left after {days_left} days is: {total_water_left} liters"
```

[9] Python

Probemos con cinco astronautas, 100 litros de agua sobrante y dos días:

```python
water_left(5, 100, 2)
```

[10] Python

```
'Total water left after 2 days is: -10 liters'
```

Esto no es muy útil, ya que una carencia en los litros sería un error. Después, el sistema de navegación podría alertar a los astronautas que no habrá suficiente agua para todos en dos días. Si eres un ingeniero(a) que programa el sistema de navegación, podrías generar una excepción en la función water_left() para alertar de la condición de error:

```python
#Generando una excepción en la función water_left() para alertar de la condición de error:
def water_left(astronauts, water_left, days_left):
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
# water_left()
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {da
    return f"Total water left after {days_left} days is: {total_water_left} liters"
```

[12] Python

Volvemos a ejecutarlo:

```python
water_left(5, 100, 2)
```

[13] Python

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 11' in <module>
----> 1 water_left(5, 100, 2)

c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 10' in water_left(astronauts,
water_left, days_left)
      5 total_water_left = water_left - total_usage
      6 if total_water_left < 0:
----> 7     raise RuntimeError(f"There is not enough water for {astronauts} astronauts afte
r {days_left} days!")
      8 return f"Total water left after {days_left} days is: {total_water_left} liters"

RuntimeError: There is not enough water for 5 astronauts after 2 days!
```

En el sistema de navegación, el código para señalar la alerta ahora puede usar ‘RuntimeError’ para generar la alerta:

```
[22]
try:
    water_left("3", "200", None)
except RuntimeError as err:
    alert_navigation_system(err)
```
Python

La función ‘water_left()’ también se puede actualizar para evitar el paso de tipos no admitidos. Intentenis pasar argumentos que no sean enteros para comprobar la salida de error:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 12' in <module>
      1 try:
----> 2     water_left("3", "200", None)
      3 except RuntimeError as err:
      4     alert_navigation_system(err)

c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 12' in water_left(astronauts,
water_left, days_left)
      3 def water_left(astronauts, water_left, days_left):
      4     daily_usage = astronauts * 11
----> 5     total_usage = daily_usage * days_left
      6     total_water_left = water_left - total_usage
      7 # water_left()

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

El error de ‘TypeError’ no es muy descriptivo en el contexto de lo que espera la función. Actualizaremos la función para que use ‘TypeError’, pero con un mensaje mejor:

Actualizaremos la función para que use `TypeError`, pero con un mensaje mejor:

```python
def water_left(astronauts, water_left, days_left):
    for argument in [astronauts, water_left, days_left]:
        try:
            # If argument is an int, the following operation will work
            argument / 10
        except TypeError:
            # TypError will be raised only if it isn't the right type
            # Raise the same exception but with a better error message
            raise TypeError(f"All arguments must be of type int, but received: '{argument}'"
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {da
    return f"Total water left after {days_left} days is: {total_water_left} liters"
```

[23] Python

Ahora volvemos a intentarlo para obtener un error mejor:

```python
water_left("3", "200", None)
```

[22] ⊗ 0.3s Python

al Help • Untitled-1.ipynb - CursoIntroPython-main - Visual Studio Code

Untitled-1.ipynb ● config.ipynb Módulo 10 - Manejo de errores.md

C: > Users > GOBIERNO DEL ESTADO > Downloads > Untitled-1.ipynb > water_left("3", "200", None)

+ Code + Markdown Run All Clear Outputs of All Cells Restart Interrupt ... Python 3.10.2

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 19' in water_left(astronauts,
water_left, days_left)
      3 try:
      4     # If argument is an int, the following operation will work
----> 5     argument / 10
      6 except TypeError:
      7     # TypError will be raised only if it isn't the right type
      8     # Raise the same exception but with a better error message

TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 20' in <module>
----> 1 water_left("3", "200", None)

c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 19' in water_left(astronauts,
water_left, days_left)
      5         argument / 10
      6     except TypeError:
      7         # TypError will be raised only if it isn't the right type
      8         # Raise the same exception but with a better error message
----> 9         raise TypeError(f"All arguments must be of type int, but received: '{argume
nt}'")
     10 daily_usage = astronauts * 11
     11 total_usage = daily_usage * days_left
```

Help • Untitled-1.ipynb - CursoIntroPython-main - Visual Studio Code

Untitled-1.ipynb ● | config.ipynb | Módulo 10 - Manejo de errores.md

> Users > GOBIERNO DEL ESTADO > Downloads > Untitled-1.ipynb > water_left("3", "200", None)

Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Interrupt | ... | Python 3.10.2

```
water_left, days_left)
      3 try:
      4     # If argument is an int, the following operation will work
----> 5     argument / 10
      6 except TypeError:
      7     # TypError will be raised only if it isn't the right type
      8     # Raise the same exception but with a better error message

TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 20' in <module>
----> 1 water_left("3", "200", None)

c:\Users\GOBIERNO DEL ESTADO\Downloads\Untitled-1.ipynb Cell 19' in water_left(astronauts,
water_left, days_left)
      5         argument / 10
      6     except TypeError:
      7         # TypError will be raised only if it isn't the right type
      8         # Raise the same exception but with a better error message
----> 9         raise TypeError(f"All arguments must be of type int, but received: '{argume
nt}'")
     10 daily_usage = astronauts * 11
     11 total_usage = daily_usage * days_left

TypeError: All arguments must be of type int, but received: '3'
```