Tracebacks

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.

```
  open("/path/to/mars.jpg")
⊗ 0.4s

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 2' in <module>
----> 1 open("/path/to/mars.jpg")

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

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

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:

```
PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Traceback (most recent call last):
  File "D:\Files\LaunchX\CursoIntroPython\Katas\open.py", line 5, in <module>
    main()
  File "D:\Files\LaunchX\CursoIntroPython\Katas\open.py", line 2, in main
    open("/path/to/mars.jpg")
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Try y Except de los bloques

Aunque es común un archivo que no existe, no es el único error que podemos encontrar. Los permisos de archivo no válidos pueden impedir la lectura de un archivo, incluso si este existe. Vamos a crear un archivo de Python denominado config.py. El archivo tiene código que busca y lee el archivo de configuración del sistema de navegación:

```
def main():
    try:
        open('config.txt')
    except FileNotFoundError:
        print("Couldn't find the config.txt file!")

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

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Couldn't find the config.txt file!
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

A continuación, quita,ps el archivo *config.txt* y creamos un directorio denominado *config.txt*. Intentaremos llamar al archivo *config.py* para ver un error nuevo que debería ser similar al siguiente:

```
PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Traceback (most recent call last):
  File "D:\Files\LaunchX\CursoIntroPython\Katas\open.py", line 9, in <module>
    main()
  File "D:\Files\LaunchX\CursoIntroPython\Katas\open.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Una manera poco útil de controlar este error sería detectar todas las excepciones posibles para evitar un traceback. Para comprender por qué detectar todas las excepciones es problemático, probaremos actualizando la función `main()`:

Ahora volvemos a ejecutar el código en el mismo lugar donde existe el archivo *config.txt* con permisos incorrectos:

```
1   def main():
2       try:
3           configuration = open('config.txt')
4       except Exception:
5           print("Couldn't find the config.txt file!")
6
7
8   if __name__ == '__main__':
9       main()
```

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Couldn't find the config.txt file!
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Vamos a corregir este fragmento de código para abordar todas estas frustraciones. Revertiremos la detección de `FileNotFoundError` y luego agregamos otro bloque `except` para detectar `PermissionError`:

```
1   def main():
2       try:
3           configuration = open('config.txt')
4       except FileNotFoundError:
5           print("Couldn't find the config.txt file!")
6       except IsADirectoryError:
7           print("Found config.txt but it is a directory, couldn't read it")
8
9
10  if __name__ == '__main__':
11      main()
```

Ahora volvemos a ejecutarlo, en el mismo lugar donde *config.txt* está con el problema de permisos:

```
PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Found config.txt but it is a directory, couldn't read it
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Eliminamos el archivo config.txt para asegurarnos de que se alcanza el primer bloque `except` en su lugar:

```
PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Couldn't find the config.txt file!
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Cuando los errores son de una naturaleza similar y no es necesario controlarlos individualmente, puedes agrupar las excepciones como una usando paréntesis en la línea `except`. Por ejemplo, si el sistema de navegación está bajo cargas pesadas y el sistema de archivos está demasiado ocupado, tiene sentido detectar `BlockingIOError` y `TimeOutError` juntos:

```
def main():
    try:
        configuration = open('config.txt')
    except FileNotFoundError:
        print("Couldn't find the config.txt file!")
    except IsADirectoryError:
        print("Found config.txt but it is a directory, couldn't read it")
    except (BlockingIOError, TimeoutError):
        print("Filesystem under heavy load, can't complete reading configuration file")

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

Si necesitas acceder al error asociado a la excepción, debes actualizar la línea `except` para incluir la palabra clave `as`. Esta técnica es práctica si una excepción es demasiado genérica y el mensaje de error puede ser útil:

```
1   def main():
2       try:
3           open("mars.jpg")
4       except FileNotFoundError as err:
5           print("got a problem trying to read the file:", err)
6
7   if __name__ == '__main__':
8       main()
```

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

En este caso, `as err` significa que `err` se convierte en una variable con el objeto de excepción como valor. Después, usa este valor para imprimir el mensaje de error asociado a la excepción. Otra razón para usar esta técnica es acceder directamente a los atributos del error. Por ejemplo, si detecta una excepción `OSError` más genérica, que es la excepción primaria de `FilenotFoundError` y `PermissionError`, podemos diferenciarlas mediante el atributo `.errno`:

```
def main():
    try:
        open("config.txt")
    except OSError as err:
        if err.errno == 2:
            print("Couldn't find the config.txt file!")
        elif err.errno == 13:
            print("Found config.txt but couldn't read it")

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

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS D:\Files\LaunchX\CursoIntroPython\Katas> python open.py
Couldn't find the config.txt file!
PS D:\Files\LaunchX\CursoIntroPython\Katas>
```

Generación de excepciones

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:

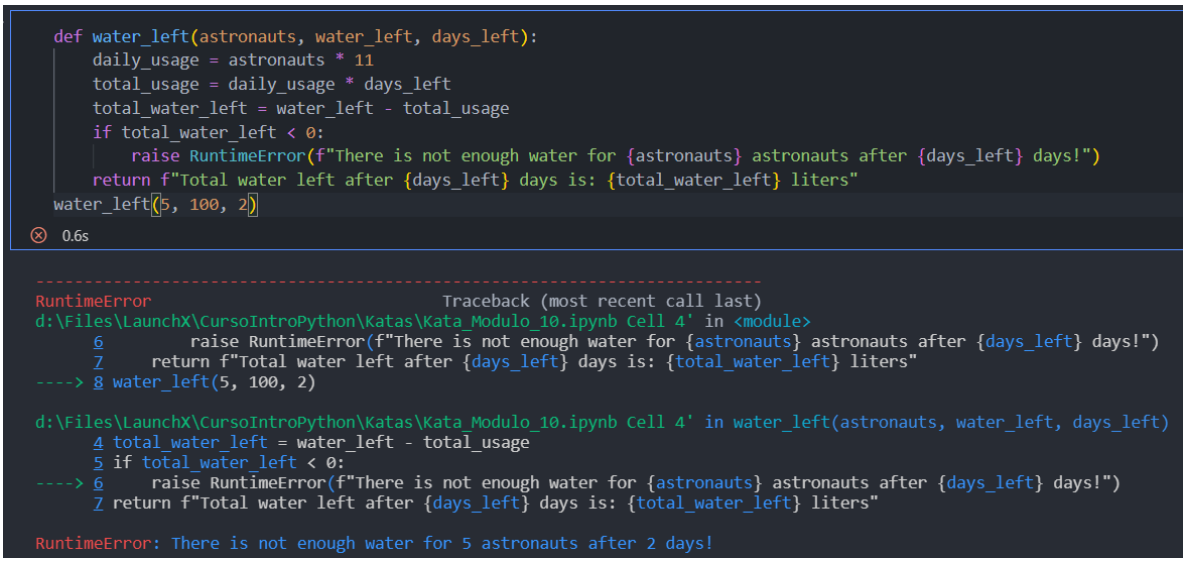
PruProbemos con cinco astronautas, 100 litros de agua sobrante y dos días:

```
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"

water_left(5, 100, 2)
```

```
✓ 0.4s
'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:



```
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
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
    return f"Total water left after {days_left} days is: {total_water_left} liters"
water_left(5, 100, 2)
```

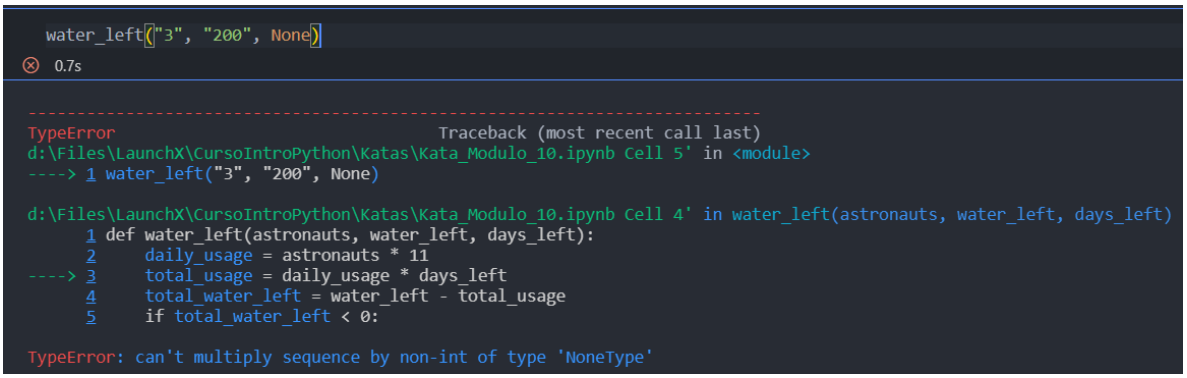
0.6s

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 4' in <module>
      6         raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
      7     return f"Total water left after {days_left} days is: {total_water_left} liters"
----> 8 water_left(5, 100, 2)

d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 4' in water_left(astronauts, water_left, days_left)
      4 total_water_left = water_left - total_usage
      5 if total_water_left < 0:
----> 6     raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
      7 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!
```

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:



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

0.7s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 5' in <module>
----> 1 water_left("3", "200", None)

d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 4' in water_left(astronauts, water_left, days_left)
      1 def water_left(astronauts, water_left, days_left):
      2     daily_usage = astronauts * 11
----> 3     total_usage = daily_usage * days_left
      4     total_water_left = water_left - total_usage
      5     if total_water_left < 0:

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:

```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 {days_left} days!")
    return f"Total water left after {days_left} days is: {total_water_left} liters"
```

✓ 0.6s

Ahora volvemos a intentarlo para obtener un error mejor:

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

⊗ 0.7s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 6' 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)
d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 7' in <module>
----> 1 water_left("3", "200", None)

d:\Files\LaunchX\CursoIntroPython\Katas\Kata_Modulo_10.ipynb Cell 6' 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: '{argument}'")
     10 daily_usage = astronauts * 11
     11 total_usage = daily_usage * days_left

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