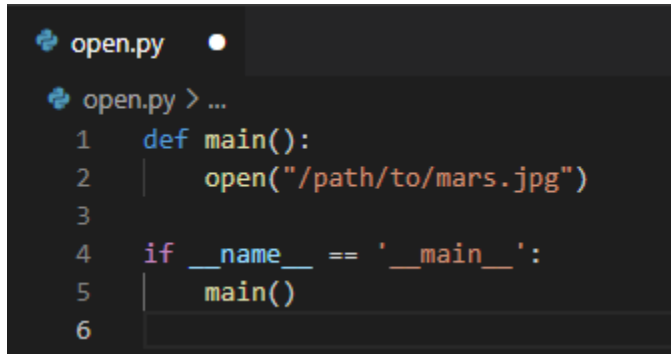


Kata Módulo 10 - Manejo de errores

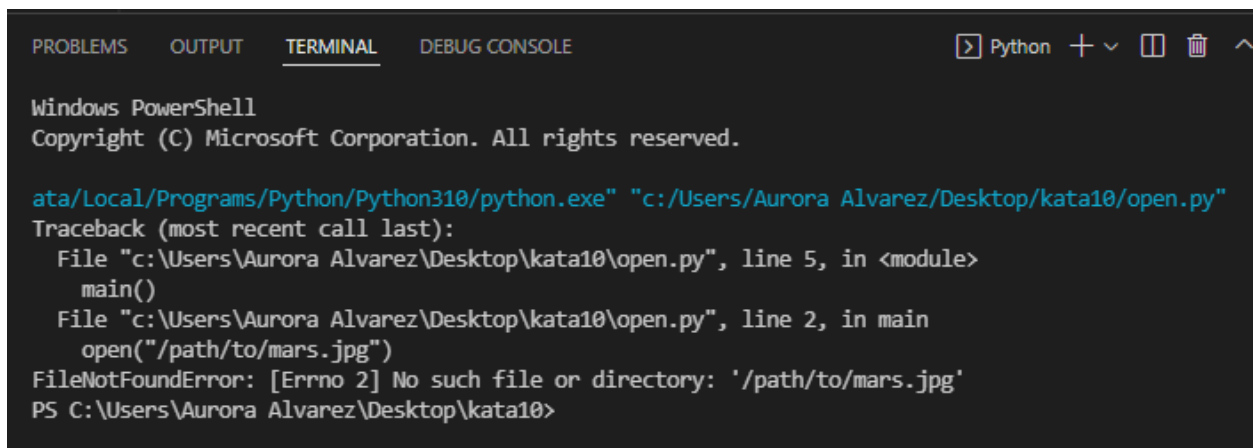
Tracebacks

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



```
open.py
open.py > ...
1 def main():
2     open("/path/to/mars.jpg")
3
4 if __name__ == '__main__':
5     main()
6
```

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:



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
Python + - [] [X] ^

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

ata/Local/Programs/Python/Python310/python.exe" "c:/Users/Aurora Alvarez/Desktop/kata10/open.py"
Traceback (most recent call last):
  File "c:/Users/Aurora Alvarez/Desktop/kata10/open.py", line 5, in <module>
    main()
  File "c:/Users/Aurora Alvarez/Desktop/kata10/open.py", line 2, in main
    open("/path/to/mars.jpg")
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
PS C:\Users\Aurora Alvarez\Desktop\kata10>
```

La salida de error tiene más sentido ahora. Las rutas de acceso apuntan a un único archivo denominado open.py. La salida menciona que el error se inicia en la línea 5, que incluye la llamada a main(). A continuación, la salida sigue el error a la línea 2 en la llamada de función open(). Y, por último, FileNotFoundError notifica de nuevo que el archivo o el directorio no existen.

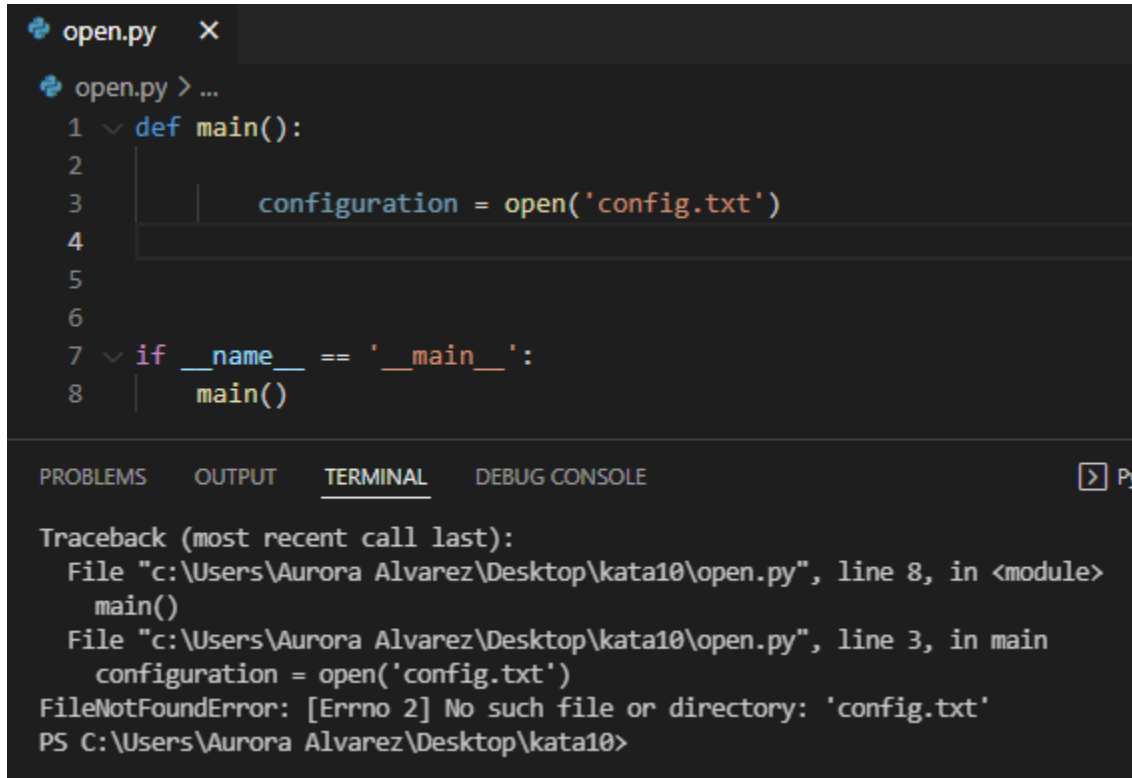
Los tracebacks casi siempre incluyen la información siguiente:

- 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.

Controlando las excepciones

Try y Except de los bloques

Al querer abrir un archivo que no existe, arroja un error en el código lo cual es lo siguiente:



```
open.py X
open.py > ...
1  def main():
2
3      configuration = open('config.txt')
4
5
6
7  if __name__ == '__main__':
8      main()

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Traceback (most recent call last):
  File "c:\Users\Aurora Alvarez\Desktop\kata10\open.py", line 8, in <module>
    main()
  File "c:\Users\Aurora Alvarez\Desktop\kata10\open.py", line 3, in main
    configuration = open('config.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'config.txt'
PS C:\Users\Aurora Alvarez\Desktop\kata10>
```

Después de la palabra clave try, agregamos código que tenga la posibilidad de producir una excepción. A continuación, agregamos la palabra clave except junto con la posible excepción, seguida de cualquier código que deba ejecutarse cuando se produce esa condición. Puesto que config.txt no existe en el sistema, Python imprime que el archivo de configuración no está ahí. El bloque try y except, junto con un mensaje útil, evita un seguimiento y sigue informando al usuario sobre el problema.

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:

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:

```
In [1]: 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     return f"Total water left after {days_left} days is: {total_water_left} liters"
```

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

```
In [2]: 1 water_left(5, 100, 2)

Out[2]: '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:

```
In [3]: 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:
        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"
```

Ahora volvemos a ejecutarlo

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

-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-4-cb536572b4c6> in <module>
----> 1 water_left(5, 100, 2)

<ipython-input-3-23ad9d658ff7> 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!
```

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

```
In [5]: 1 try:
2       water_left(5, 100, 2)
3 except RuntimeError as err:
4       alert_navigation_system(err)

-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-5-5a2433861cc6> in <module>
      1 try:
----> 2     water_left(5, 100, 2)
      3 except RuntimeError as err:

<ipython-input-3-23ad9d658ff7> in water_left(astronauts, water_left, days_left)
      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!

During handling of the above exception, another exception occurred:

NameError                                Traceback (most recent call last)
<ipython-input-5-5a2433861cc6> in <module>
      2     water_left(5, 100, 2)
      3 except RuntimeError as err:
----> 4     alert_navigation_system(err)

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

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:

```
1 def water_left(astronauts, water_left, days_left):
2     for argument in [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             # TypeError 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
12    total_water_left = water_left - total_usage
13    if total_water_left < 0:
14        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
15    return f"Total water left after {days_left} days is: {total_water_left} liters"
```

Ahora volvemos a intentarlo para obtener un error mejor:

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

-----
TypeError                                Traceback (most recent call last)
<ipython-input-6-c9067c6af39d> in water_left(astronauts, water_left, days_left)
      4             # If argument is an int, the following operation will work
----> 5             argument / 10
      6         except TypeError:

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

During handling of the above exception, another exception occurred:

TypeError                                Traceback (most recent call last)
<ipython-input-7-440ddff11c7b> in <module>
----> 1 water_left("3", "200", None)

<ipython-input-6-c9067c6af39d> in water_left(astronauts, water_left, days_left)
      7             # TypeError 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'
```