

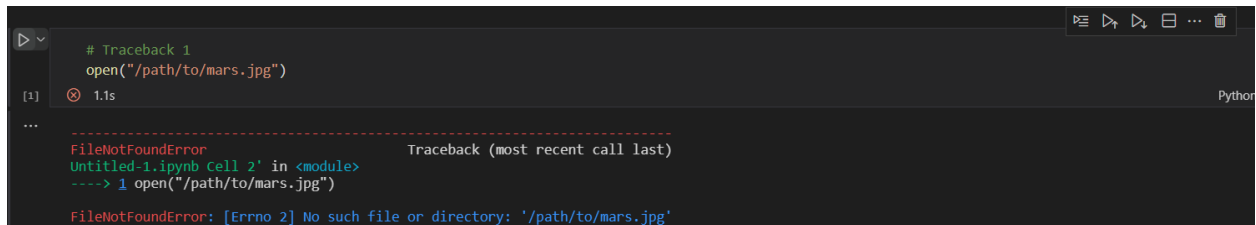
Kata 10 Manejo de errores

Uso de tracebacks para buscar errores

Las excepciones en Python son una característica principal del lenguaje. Es posible que te sorprenda leer que algo que genera errores se resalta como una característica. Esta sorpresa puede deberse a que las herramientas de software sólidas no parecen bloquearse con un traceback seguimiento (varias líneas de texto que indican cómo se inició y finalizó el error).

Sin embargo, las excepciones son útiles porque ayudan en la toma de decisiones generando mensajes de error descriptivos. Pueden ayudarte a controlar los problemas esperados e inesperados.

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

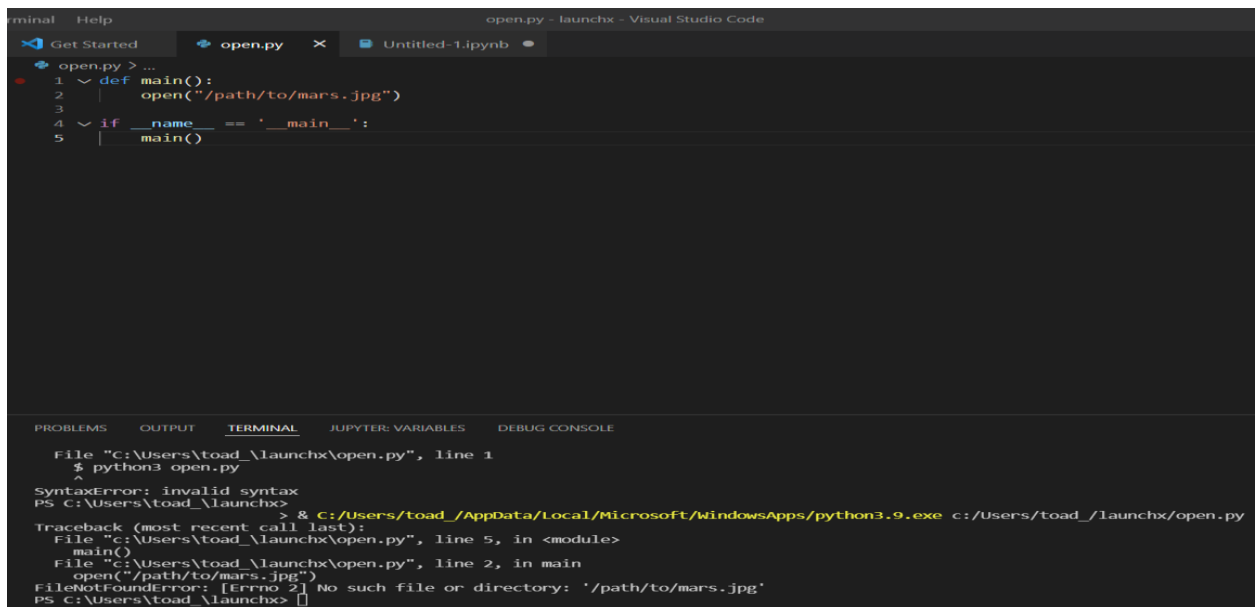


```
# Traceback 1
open("/path/to/mars.jpg")
(1) 1.1s
...
-----
FileNotFoundError                                Traceback (most recent call last)
Untitled-1.ipynb cell 2 in <module>
----> 1 open("/path/to/mars.jpg")

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

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:



```
File "C:\Users\toad_\launchx\open.py", line 1
$ python3 open.py
^
SyntaxError: invalid syntax
PS C:\Users\toad_\launchx>
> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/open.py
Traceback (most recent call last):
  File "C:\Users\toad_\launchx\open.py", line 5, in <module>
    main()
  File "C:\Users\toad_\launchx\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\toad_\launchx>
```

Try y Except de los bloques

Vamos a usar el ejemplo de navegador a fin de crear código que abra archivos de configuración para la misión de Marte. 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:
    open('config.txt')
except FileNotFoundError:
    print("Couldn't find the config.txt file!")
```

[3] ✓ 0.1s Python

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

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:

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

PROBLEMS OUTPUT **TERMINAL** JUPYTER: VARIABLES DEBUG CONSOLE

```
File "c:\Users\toad\launchx\config.py", line 9, in <module>
    main()
File "c:\Users\toad\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
Traceback (most recent call last):
  File "c:\Users\toad\launchx\config.py", line 9, in <module>
    main()
  File "c:\Users\toad\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> []
```

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():

```
config.py > ...
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()
```

PROBLEMS OUTPUT TERMINAL JUPYTER: VARIABLES DEBUG CONSOLE

```
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/windowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/windowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/windowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/windowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
Traceback (most recent call last):
  File "c:\Users\toad_\launchx\config.py", line 9, in <module>
    main()
  File "c:\Users\toad_\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/windowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
Couldn't find the config.txt file!
PS C:\Users\toad_\launchx> █
```

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`:

```
config.py > main
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     except PermissionError:
9         print("No tengo permisos para leer el archivo.txt")
10
11 if __name__ == '__main__':
12     main()
```

PROBLEMS OUTPUT **TERMINAL** JUPYTER: VARIABLES DEBUG CONSOLE

```
File "c:\Users\toad\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
Traceback (most recent call last):
  File "c:\Users\toad\launchx\config.py", line 11, in <module>
    main()
  File "c:\Users\toad\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
No tengo permisos para leer el archivo.txt
PS C:\Users\toad\launchx>
```

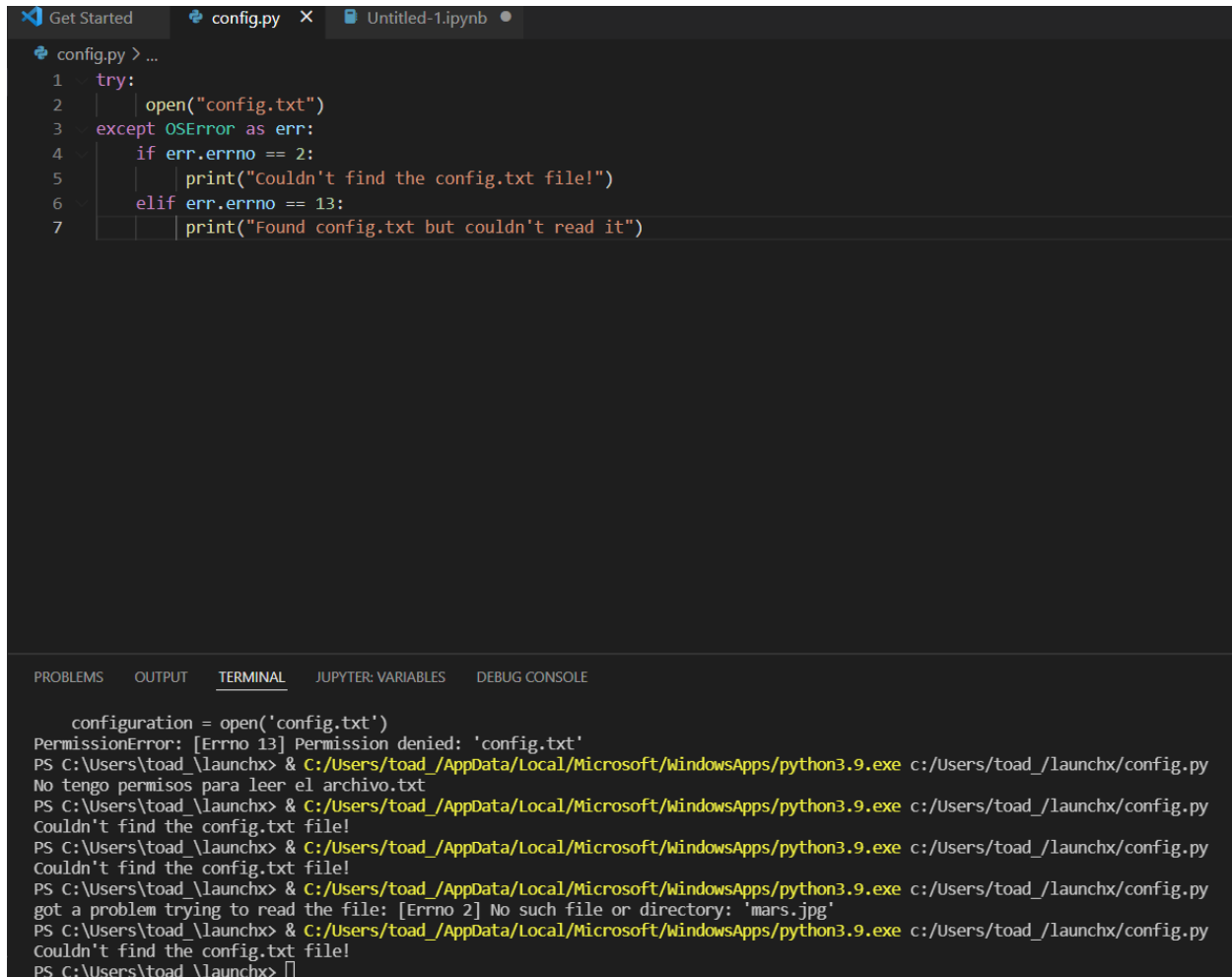
Eliminamos el archivo `config.txt` para asegurarnos de que se alcanza el primer bloque `except` en su lugar:

```
config.py > ...
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     except PermissionError:
9         print("No tengo permisos para leer el archivo.txt")
10
11 if __name__ == '__main__':
12     main()
```

PROBLEMS OUTPUT **TERMINAL** JUPYTER: VARIABLES DEBUG CONSOLE

```
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
Traceback (most recent call last):
  File "c:\Users\toad\launchx\config.py", line 11, in <module>
    main()
  File "c:\Users\toad\launchx\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
No tengo permisos para leer el archivo.txt
PS C:\Users\toad\launchx> & C:/Users/toad/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad/launchx/config.py
Couldn't find the config.txt file!
PS C:\Users\toad\launchx>
```

En este caso, `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`:

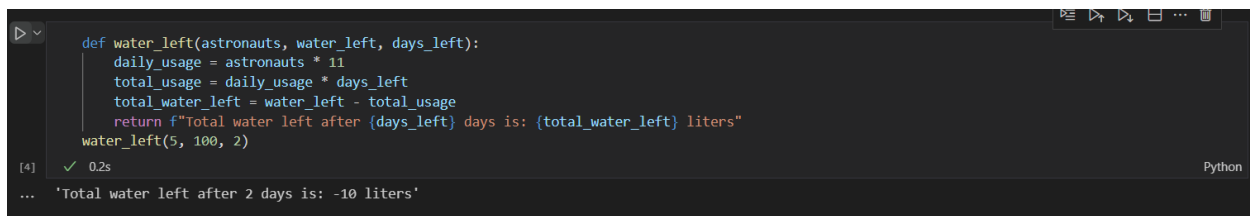


```
config.py > ...
1  try:
2      | open("config.txt")
3  except OSError as err:
4      | if err.errno == 2:
5      |     | print("Couldn't find the config.txt file!")
6      | elif err.errno == 13:
7      |     | print("Found config.txt but couldn't read it")

configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
No tengo permisos para leer el archivo.txt
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
Couldn't find the config.txt file!
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
Couldn't find the config.txt file!
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
PS C:\Users\toad_\launchx> & C:/Users/toad_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/toad_/launchx/config.py
Couldn't find the config.txt file!
PS C:\Users\toad_\launchx> 
```

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:

Probemos 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)

[4]: ✓ 0.2s 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:

Ahora volvemos a ejecutarlo.

```
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)
```

⊗ 0.2s

```
-----
RuntimeError                                Traceback (most recent call last)
Untitled-1.ipynb Cell 5' 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)

Untitled-1.ipynb Cell 5' 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. Intentemos pasar argumentos que no sean enteros para comprobar la salida 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)
```

⊗ 0.2s

```
-----
RuntimeError                                Traceback (most recent call last)
Untitled-1.ipynb Cell 5' 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)

Untitled-1.ipynb Cell 5' 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!
```

Ahora volvemos a intentarlo para obtener un error mejor:

```
Get Started  config.py  Untitled-1.ipynb
+ Code  + Markdown  | ▶ Run All  | Clear Outputs of All Cells  | Restart  | Interrupt  | Variables  | Outline  | ...
[9] 0.2s
    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"
water_left("3", "200", None)

-----
TypeError                                Traceback (most recent call last)
Untitled-1.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     # TypeError 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)
Untitled-1.ipynb Cell 6' in <module>
     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"
----> 16 water_left("3", "200", None)

Untitled-1.ipynb Cell 6' in water_left(astronauts, water_left, days_left)
      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

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