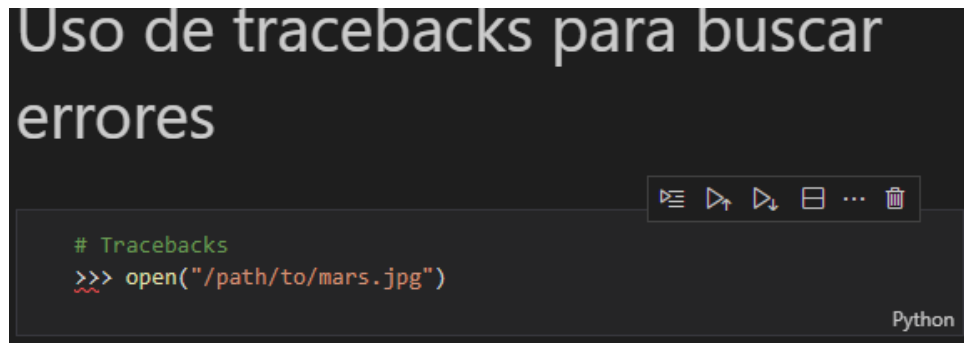


Modulo 10- Control de Excepciones

- Tracebacks

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

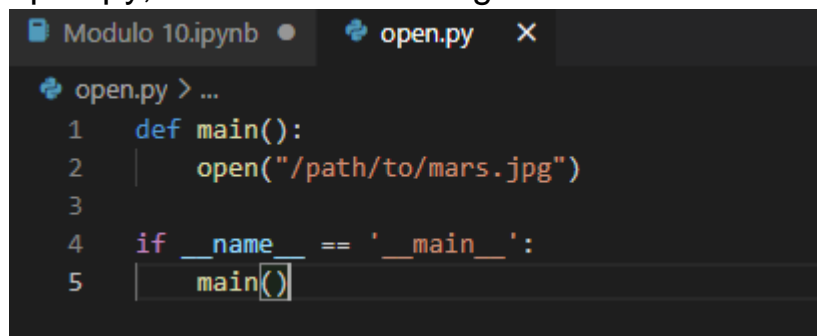


```
# Tracebacks
>>> open("/path/to/mars.jpg")
```

```
-----
FileNotFoundError                                Traceback (most recent call
1 last)
d:\VidualCode\Jupyter\CursoPython\Modulo 10\Modulo 10.ipynb Cell 2'
in <module>
      1 # Tracebacks
----> 2 open("/path/to/mars.jpg")

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

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



```
Modulo 10.ipynb • open.py X

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

```

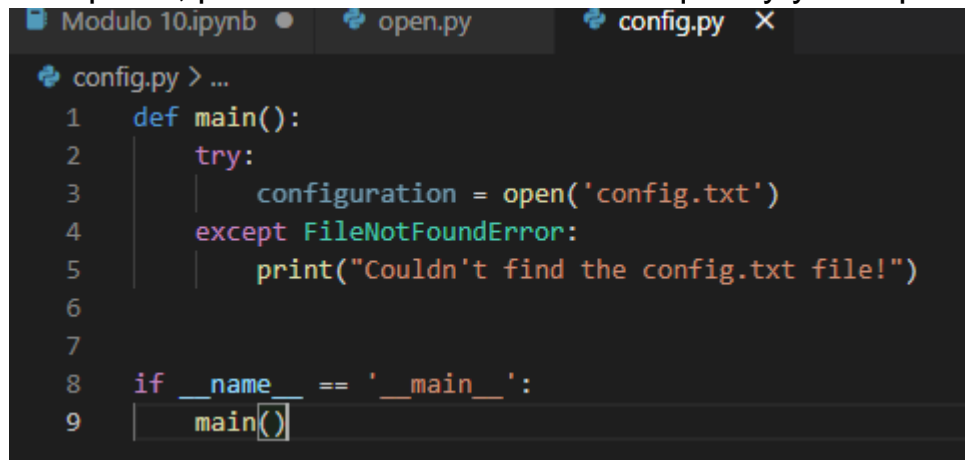
PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/C
ursoPython/open.py
Traceback (most recent call last):
  File "d:\VidualCode\Jupyter\CursoPython\open.py", line 5, in <module>
    main()
  File "d:\VidualCode\Jupyter\CursoPython\open.py", line 2, in main
    open("/path/to/mars.jpg")
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
PS D:\VidualCode\Jupyter\CursoPython>

```

- Controlando las excepciones

Try y Except de los bloques

1. 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:



```

Modulo 10.ipynb  open.py  config.py X
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()

```

2. A continuación, quitamos 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:

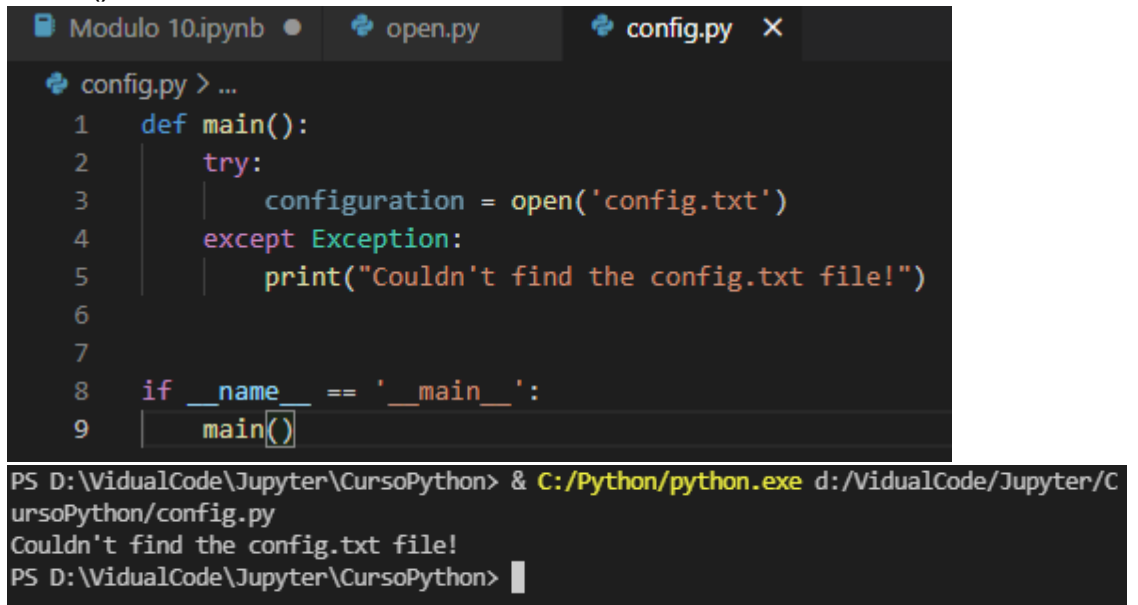
config.txt	16/02/2022 08:12 p. m.	Carpeta de archivos
Modulo 0	09/02/2022 02:14 p. m.	Carpeta de archivos
Modulo 1	09/02/2022 02:14 p. m.	Carpeta de archivos
.....

```

PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/C
ursoPython/config.py
Traceback (most recent call last):
  File "d:\VidualCode\Jupyter\CursoPython\config.py", line 9, in <module>
    main()
  File "d:\VidualCode\Jupyter\CursoPython\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
PS D:\VidualCode\Jupyter\CursoPython>

```

3. 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()`:



```

Modulo 10.ipynb  open.py  config.py X
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()

```

```

PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/C
ursoPython/config.py
Couldn't find the config.txt file!
PS D:\VidualCode\Jupyter\CursoPython>

```

4. 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**:

```
Modulo 10.ipynb • open.py config.py X
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 PermissionError:
7         print("Found config.txt but it is a directory, couldn't read it")
8
9
10 if __name__ == '__main__':
11     main()
```

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

```
PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/CursoPython/config.py
Found config.txt but it is a directory, couldn't read it
PS D:\VidualCode\Jupyter\CursoPython>
```

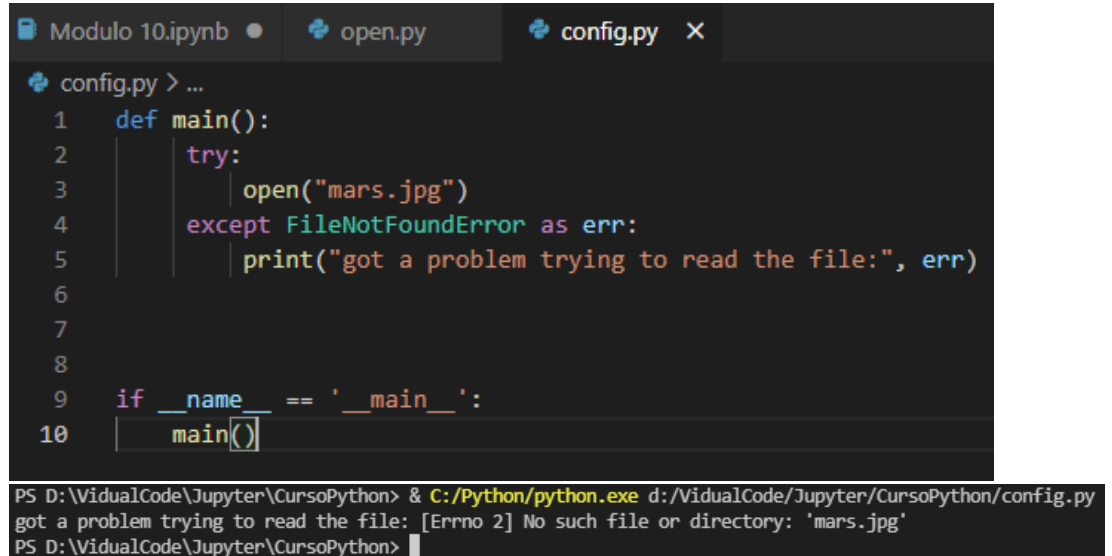
6. Eliminamos el archivo config.txt para asegurarnos de que se alcanza el primer bloque except en su lugar:

```
PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/CursoPython/config.py
Couldn't find the config.txt file!
PS D:\VidualCode\Jupyter\CursoPython>
```

7. 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:

```
Modulo 10.ipynb • open.py config.py X
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 (BlockingIOError, TimeoutError):
9         print("Filesystem under heavy load, can't complete reading configuration file")
10
11
12 if __name__ == '__main__':
13     main()
```

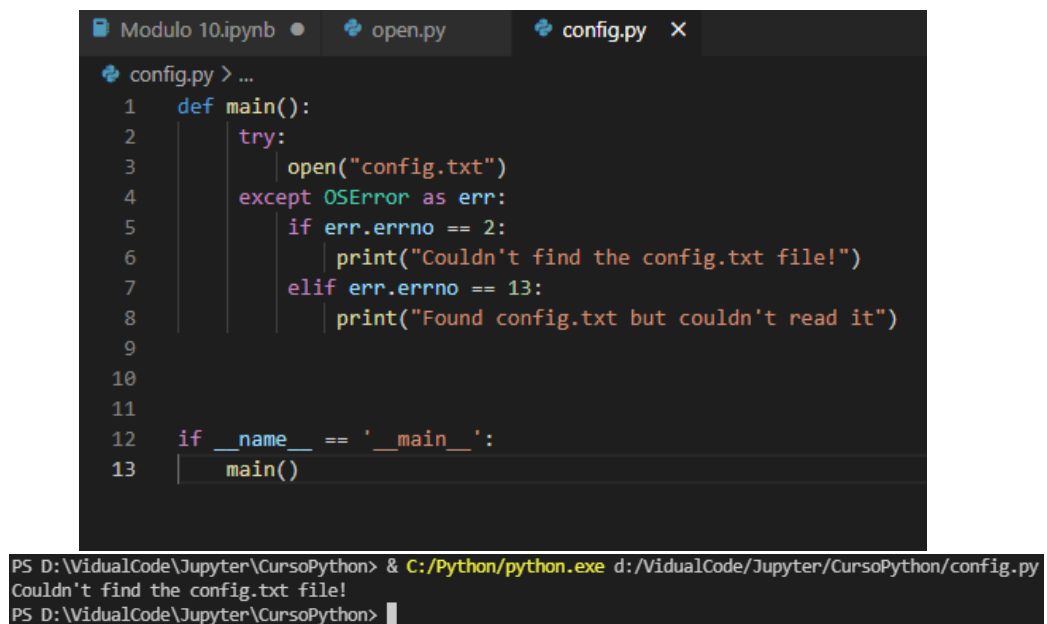
8. Si necesitas acceder al error asociado a la excepción, debes actualizar la línea **except** para incluir la palabra clave **as**. Esta técnica es práctica si una excepción es demasiado genérica y el mensaje de error puede ser útil:



```
Modulo 10.ipynb • open.py config.py X
config.py > ...
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
8
9 if __name__ == '__main__':
10     main()

PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/CursoPython/config.py
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
PS D:\VidualCode\Jupyter\CursoPython>
```

9. Por ejemplo, si detecta una excepción **OSError** más genérica, que es la excepción primaria de **FileNotFoundError** y **PermissionError**, podemos diferenciarlas mediante el atributo **.errno**:



```
Modulo 10.ipynb • open.py config.py X
config.py > ...
1 def main():
2     try:
3         open("config.txt")
4     except OSError as err:
5         if err.errno == 2:
6             print("Couldn't find the config.txt file!")
7         elif err.errno == 13:
8             print("Found config.txt but couldn't read it")
9
10
11
12 if __name__ == '__main__':
13     main()

PS D:\VidualCode\Jupyter\CursoPython> & C:/Python/python.exe d:/VidualCode/Jupyter/CursoPython/config.py
Couldn't find the config.txt file!
PS D:\VidualCode\Jupyter\CursoPython>
```

- Generación de excepciones
 1. 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:

Limitacion de agua en los astronautas

```
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"
```

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

```
# muestra
water_left(5,100,2)

✓ 0.4s

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

3. 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"
```

4. Ahora volvemos a ejecutarlo

```
# muestra
water_left(5,100,2)

⊗ 0.4s Python

-----
RuntimeError                                Traceback (most recent call last)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 3' in <module>
      1 # muestra
----> 2 water_left(5,100,2)

d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 2' 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!
```

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

```
-----
RuntimeError                                Traceback (most recent call last)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 3' in <module>
      2 try:
----> 3     water_left(5, 100, 2)
      4 except RuntimeError as err:

d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 2' in water_left(astronauts, water_left, day
s_left)
      5 if total_water_left < 0:
----> 6     raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} da
ys!")
      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)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 3' in <module>
      3     water_left(5, 100, 2)
      4 except RuntimeError as err:
----> 5     alert_navigation_system(err)

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

6. 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:

```
water_left("3", "200", None)
⊗ 0.4s Python

-----
TypeError                                Traceback (most recent call last)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 4' in <module>
----> 1 water_left("3", "200", None)

d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 2' in water_left(astronauts, water_left, day
s_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'
```

7. 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:

```
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:
            # TypeError 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"
```

8. Ahora volvemos a intentarlo para obtener un error mejor:

```
water_left("3", "200", None)
0.8s Python

-----
TypeError                                Traceback (most recent call last)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 5' 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)
d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 6' in <module>
----> 1 water_left("3", "200", None)

d:\VidualCode\Jupyter\CursoPython\litrosAstronautas.ipynb Cell 5' 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'
```