

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

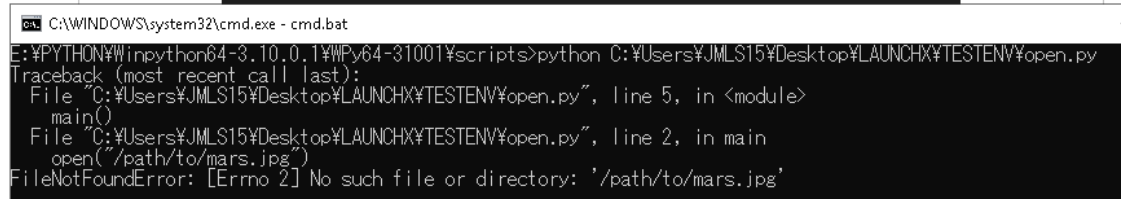
```
...

def main():

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

if __name__ == '__main__':

    main()
```

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe - cmd.bat". The command prompt shows the execution of a Python script: `E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\open.py`. The output is a traceback: `Traceback (most recent call last):  
 File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\open.py", line 5, in <module>  
 main()  
 File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\open.py", line 2, in main  
 open("/path/to/mars.jpg")  
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'`

### ### 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!")

...

Couldn't find the config.txt file!

...
```

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\open.py
Couldn't find the config.txt file!
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>
```

A continuación, quita 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:

```
$ python config.py
```

```
Traceback (most recent call last):
```

```
File "/tmp/config.py", line 9, in <module>
```

```
    main()
```

```
File "/tmp/config.py", line 3, in main
```

```
    configuration = open('config.txt')
```

```
IsADirectoryError: [Errno 21] Is a directory: 'config.txt'
```

```
...
```

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py
Traceback (most recent call last):
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 9, in <module>
    main()
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 3, in main
    configuration = open(r'C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.txt')
PermissionError: [Errno 13] Permission denied: 'C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.txt'
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>_
```

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()``:

```
...
```

```
def main():
```

```
    try:
```

```
        configuration = open('config.txt')
```

```
    except Exception:
```

```
        print("Couldn't find the config.txt file!")
```

'''

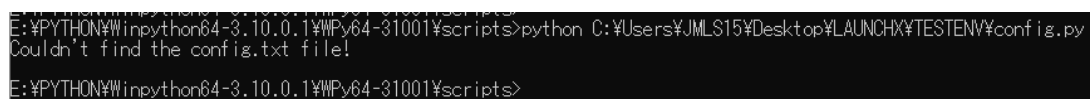
Ahora volvemos a ejecutar el código en el mismo lugar donde existe el archivo `*config.txt*` con permisos incorrectos:

'''

```
$ python config.py
```

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

'''

A screenshot of a Windows command prompt window. The prompt shows the command 'python C:\Users\JMLS15\Desktop\LAUNCH\TESTEN\config.py' being executed. The output is 'Couldn't find the config.txt file!'. The prompt is 'E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>'.

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``:

'''

```
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")
```

'''

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

'''

```
$ python config.py
```

```
Found config.txt but couldn't read it
```

```
...
```

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTEN\config.py
Found config.txt but it is a directory, couldn't read it
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>
```

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

```
...
```

```
$ rm -f config.txt
```

```
$ python config.py
```

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

```
...
```

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTEN\config.py
Couldn't find the config.txt file!
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>
```

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:

```
...
```

```
>>> try:
```

```
...     open("mars.jpg")
```

```
... except FileNotFoundError as err:
```

```
...     print("got a problem trying to read the file:", err)
```

```
...
```

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

```
E:\PYTHON\Winpython64-3.10.0.1\WP64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
E:\PYTHON\Winpython64-3.10.0.1\WP64-31001\scripts>
```

...

>>> 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")
... 
```

...

Couldn't find the config.txt file!

...

```
E:\PYTHON\Winpython64-3.10.0.1\WP64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py
Couldn't find the config.txt file!
E:\PYTHON\Winpython64-3.10.0.1\WP64-31001\scripts>
```

```
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"
...
```

Ahora volvemos a ejecutarlo

...

>>> water\_left(5, 100, 2)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 6, in water\_left

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

...

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCH\TESTENV\config.py
Traceback (most recent call last):
  File "C:\Users\JMLS15\Desktop\LAUNCH\TESTENV\config.py", line 12, in <module>
    main()
  File "C:\Users\JMLS15\Desktop\LAUNCH\TESTENV\config.py", line 10, in main
    water_left(5, 100, 2)
  File "C:\Users\JMLS15\Desktop\LAUNCH\TESTENV\config.py", line 8, in water_left
    raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
RuntimeError: There is not enough water for 5 astronauts after 2 days!
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>
```

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"
```

Ahora volvemos a intentarlo para obtener un error mejor:

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

Traceback (most recent call last):

File "<stdin>", line 5, in water\_left

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

During handling of the preceding exception, another exception occurred:

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 9, in water\_left

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

```
E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>python C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py
Traceback (most recent call last):
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 7, in water_left
    argument / 10
TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 20, in <module>
    main()
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 18, in main
    water_left("3", "200", None)
  File "C:\Users\JMLS15\Desktop\LAUNCHX\TESTENV\config.py", line 11, in water_left
    raise TypeError(f"All arguments must be of type int, but received: '[argument]')
TypeError: All arguments must be of type int, but received: '3'

E:\PYTHON\Winpython64-3.10.0.1\WPY64-31001\scripts>
```