Carballo Ramírez Hanny

Kata Módulo 10

Tracebacks

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

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

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
c:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10Katas.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:

Módulo10katas > open.py > ...

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

Ejecútala con Python y podrás comprobar el siguiente mensaje de error:

```
C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>python ope
n.py
Traceback (most recent call last):
  File "C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas\op
en.py", line 5, in <module>
    main()
  File "C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas\op
en.py", line 2, in main
    open("/path/to/mars.jpg")
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
```

Controlando las excepciones

Aunque en este módulo se explica cómo controlar las excepciones detectándolas, no es necesario detectar las excepciones todo el tiempo. A veces resulta útil permitir que se puedan generar excepciones para que otros autores de llamadas puedan tratar los errores.

Try y Except de los bloques

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!")
...
[1] ✓ 0.3s                                                    Python
Couldn't find the config.txt file!
```

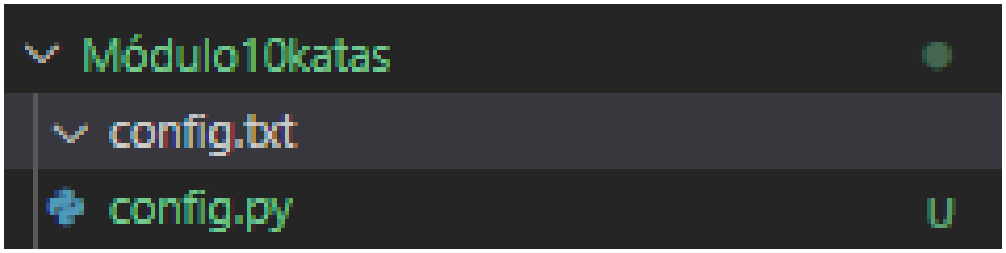
Aunque es común un archivo que no existe, no es el único error que podemos encontrar. Los permisos de archivo no válidos pueden impedir la lectura de un archivo, incluso si este existe. Vamos a crear un archivo de Python denominado config.py. El archivo tiene código que busca y lee el archivo de configuración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

Módulo10katas > config.py > ...

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


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

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:

```
C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>python con
fig.py
Traceback (most recent call last):
  File "C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas\co
nfig.py", line 9, in <module>
    main()
  File "C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas\co
nfig.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'
```

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():

```
Módulo10katas > config.py > main
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()
```

```
C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>python con
fig.py
Couldn't find the config.txt file!

C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>
```

El problema ahora es que el mensaje de error es incorrecto. El archivo existe, pero tiene permisos diferentes y Python no puede leerlo.

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:

```
Módulo10katas > 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
9   if __name__ == '__main__':
10      main()
```

```
C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>python con
fig.py
Found config.txt but it is a directory, couldn't read it

C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>
```

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

```
C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>python con
fig.py
Couldn't find the config.txt file!

C:\Users\Portable\Desktop\launchX\CursoPython- KatasPropias\Módulo10katas>
```

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:

```
Módulo10katas > config.py > main
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")
```

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:

```
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"
✓ 0.5s
```

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"
✓ 0.1s
```

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:
            # 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"
✓ 0.1s
```