

Kata del módulo 10

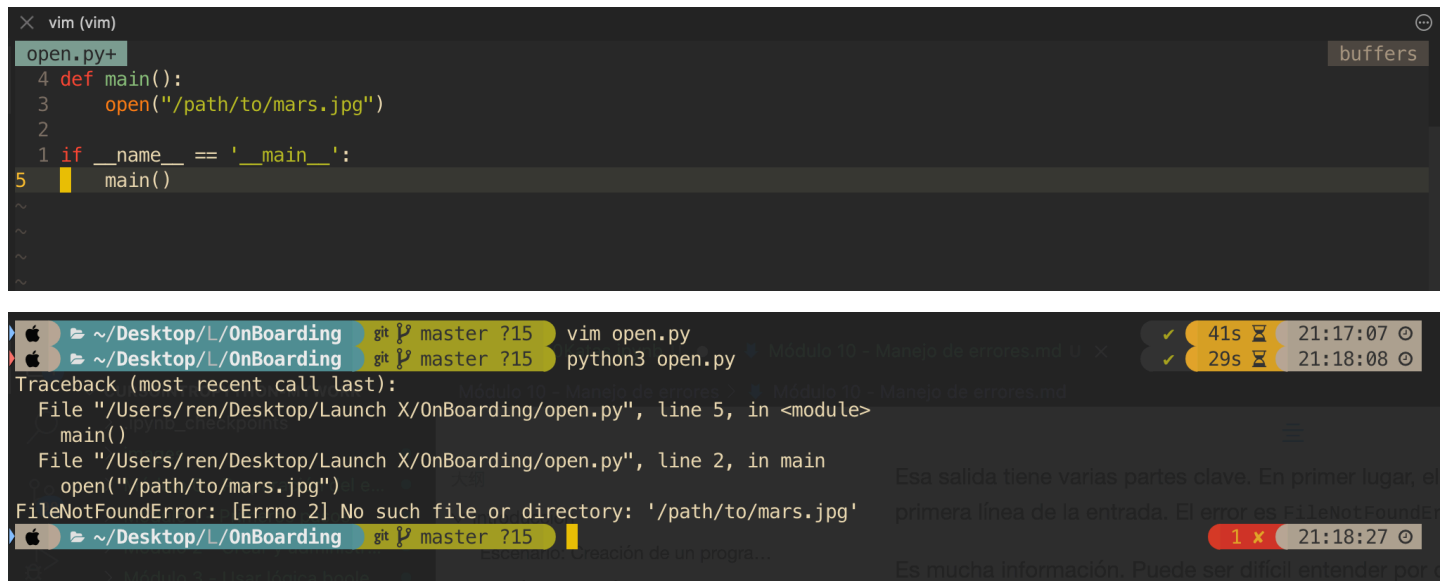
Uso de tracebacks para buscar errores

Tracebacks

#

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

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



The screenshot shows a terminal window with a dark background. At the top, a terminal window titled 'vim (vim)' shows the contents of 'open.py' with line numbers 1 through 5. Below this, a terminal window shows the command 'python3 open.py' being executed. The output is a traceback error: 'FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg''. The traceback shows the call stack: 'File "/Users/ren/Desktop/Launch X/OnBoarding/open.py", line 5, in <module>: main()', 'File "/Users/ren/Desktop/Launch X/OnBoarding/open.py", line 2, in main: open("/path/to/mars.jpg")', and the final error message. The terminal window also shows a list of recent commands and their execution times.

Controlando las excepciones

Try y Except de los bloques

#

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:

The image shows a terminal window at the top with the command `vim config.py` and a file explorer on the left. Below the terminal is a code editor showing the content of `config.py`:

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

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.

The image shows a terminal window where the command `python3 config.py` has been executed. The output is a traceback error:

```
Traceback (most recent call last):
  File "/Users/ren/Desktop/Launch X/OnBoarding/Kata 10/config.py", line 9, in <module>
    main()
  File "/Users/ren/Desktop/Launch X/OnBoarding/Kata 10/config.py", line 3, in main
    configuration = open('config.txt')
IsADirectoryError: [Errno 21] Is a directory: 'config.txt'
```

Una manera poco útil de controlar este error sería detectando todas las excepciones para evitar un Traceback. Para comprenderlo mejor probaremos actualizando la función `main()`

The image shows a code editor with the updated `main()` function in `config.py`:

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

Que, ejecutando en consola, nos aparece lo siguiente:

The image shows a terminal window where the command `python3 config.py` has been executed. The output is:

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

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`:

```

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")

```

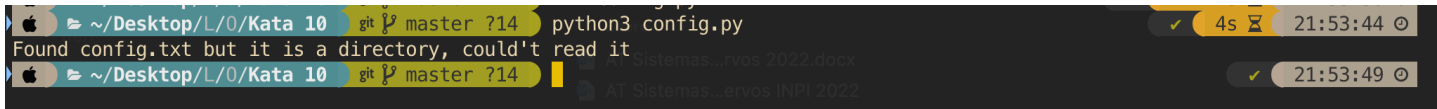


```

vim (vim)
config.py
9 def main():
8     try:
7         configuration = open('config.txt')
6     except FileNotFoundError:
5         print("Couldn't find the config.txt file!")
4     except IsADirectoryError:
3         print("Found config.txt but it is a directory, couldn't read it")
2
1 if __name__ == '__main__':
10     main()

```

Y al ejecutarlo, nos produce el siguiente resultado en consola:



```

~/Desktop/L/0/Kata 10 git master ?14 python3 config.py
Found config.txt but it is a directory, couldn't read it

```

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



```

~/Desktop/L/0/Kata 10 git master ?14 rmdir config.txt
~/Desktop/L/0/Kata 10 git master ?14 python3 config.py
Couldn't find the config.txt file!

```

Podemos agrupar las excepciones como si fuera una, usando paréntesis en la línea `except`, por ejemplo, si el sistema está bajo cargas pesadas y el sistema de archivos está demasiado ocupado, tiene sentido detectar `BlockingIOError` y `TimeoutError` juntos:



```

vim (vim)
config.py+
11 def main():
10     try:
9         configuration = open('config.txt')
8     except FileNotFoundError:
7         print("Couldn't find the config.txt file!")
6     except IsADirectoryError:
5         print("Found config.txt but it is a directory, couldn't read it")
4     except (BlockingIOError, TimeoutError):
3         print("Filesystem under heavy load, can't complete reading configuration file")
2
1 if __name__ == '__main__':
12     main()

```

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:

```
vim (vim)
mars.py+
7 def main():
6     try:
5         open("mars.jpg")
4     except FileNotFoundError as err:
3         print("got a problem trying to read the file:", err)
2
1 if __name__ == '__main__':
8     main()
```

Y en consola nos produce lo siguiente:

```
~/Desktop/L/0/Kata 10 git master ?17 python3 mars.py
got a problem trying to read the file: [Errno 2] No such file or directory: 'mars.jpg'
~/Desktop/L/0/Kata 10 git master ?17
```

En este caso, `as 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 utilizar 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 diferenciarla mediante el atributo `.errno`:

```
vim (vim)
config.py
10 def main():
9     try:
8         configuration = open('config.txt')
7     except OSError as err:
6         if err.errno == 2:
5             print("Couldn't find the config.txt")
4         elif err.errno == 13:
3             print("Found config.txt but it is a directory, couldn't read it")
2
1 if __name__ == '__main__':
11     main()
```

Y en consola nos produce lo siguiente:

```
~/Desktop/L/0/Kata 10 git master ?19 python3 config.py
Couldn't find the config.txt
~/Desktop/L/0/Kata 10 git master ?19
```

Generación de excepciones

Si conocemos una situación que podría provocar una condición de error al escribir código, resulta útil generar excepciones que permitan que otro código comprenda cuál es el problema.

En este ejemplo, tenemos que los astronautas limitan el uso de agua a 11 litros diarios; crearemos una función que, con base al número de astronautas, pueda calcular la cantidad de agua que quedará después de un día o más:

```
x vim (vim)
astronauts.py
5 def water_left(astronauts, water_left, days_left):
4     daily_usage = astronauts * 11
3     total_usage = daily_usage * days_left
2     total_water_left = water_left - total_usage
1     return f"Total water left after {days_left} days is: {total_water_left} liters"
6
```

Probemos con 5 astronautas, 100 litros de agua y 2 días:

```
▶ v 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"
[1] ✓ 0.4s Python

▶ v water_left(5, 100, 2)
[2] ✓ 0.3s 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; para ello generaremos una excepción en la función `water_left()` para alertar de la condición del 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!")
✓ 0.2s Python
```

Y al volverlo a ejecutar, tenemos:

```
▶ v water_left(5, 100, 2)
[4] ✗ 0.6s Python
...
RuntimeError                                Traceback (most recent call last)
/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.ipynb Cell 4' in <module>
----> 1 water_left(5, 100, 2)

/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.ipynb Cell 3' 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!")

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

Actualizamos la función `water_left()` para evitar el paso de tipos no admitidos

```
def water_left(astronauts, water_left, days_left):
    try:
        water_left(5, 100, 2)
    except RuntimeError as err:
        alert_navigation_system(err)
    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"
```

✓ 0.3s

Y al pasar los argumentos, comprobamos el error `TypeError` :

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

✗ 0.5s

Python

```
-----
TypeError                                 Traceback (most recent call last)
/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.ipynb Cell 4' in <module>
----> 1 water_left("3", "200", None)

/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.ipynb Cell 3' in water_left(astronauts, water_
left, days_left)
     1 def water_left(astronauts, water_left, days_left):
     2     try:
----> 3         water_left(5, 100, 2)
     4     except RuntimeError as err:
     5         alert_navigation_system(err)

TypeError: 'str' object is not callable
```

Este error no es muy descriptivo en el contexto de la función, por lo que actualizaremos la función para que use `TypeError` pero con un mensaje:

```
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"
```

✓ 0.3s

Python

Y al volver a intentarlo, obtenemos:

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

[9] 0.6s Python

-----
TypeError                                Traceback (most recent call last)
/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.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)
/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.ipynb Cell 6' in <module>
----> 1 water_left("3", "200", None)

/Users/ren/Desktop/Launch X/CursoIntroPython-MyWork/Módulo 10 - Manejo de errores/Módulo10Katas.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'
```

Fin del ejercicio

- Github: [Rene-Bedolla](#)