

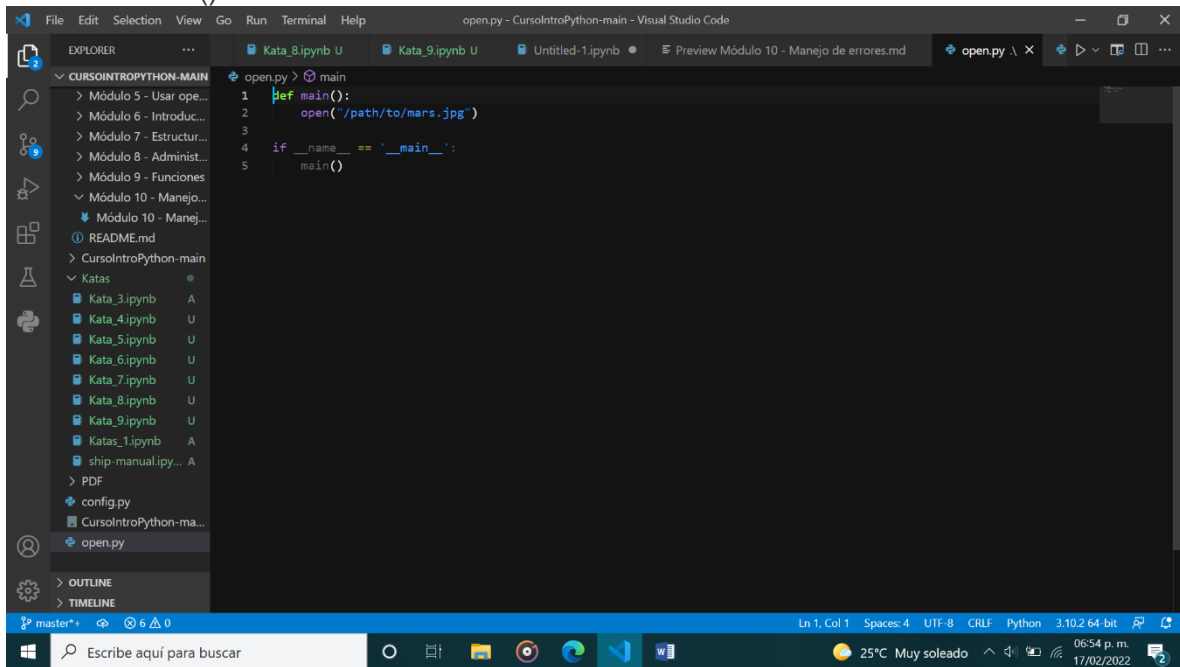
# Kata 10

## Manejo de errores

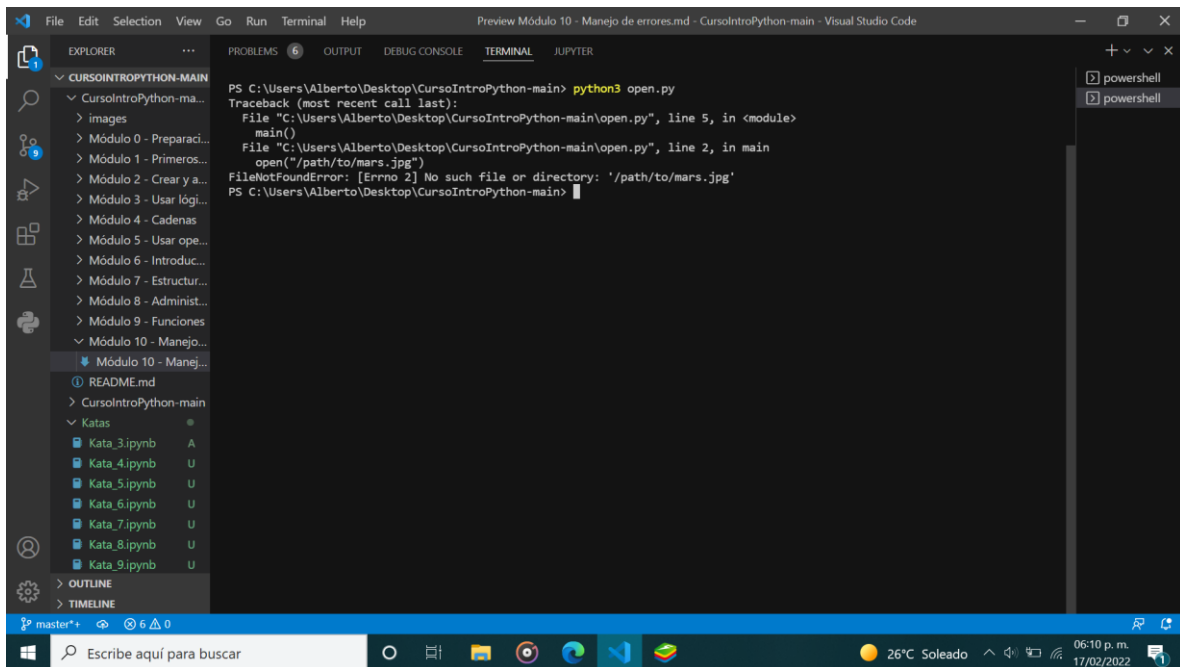
### Tracebacks

1. Crear un archivo de Python y asignar el nombre open.py, con el siguiente código:

```
def main():  
    open("/path/to/mars.jpg")  
  
if __name__ == '__main__':  
    main()
```

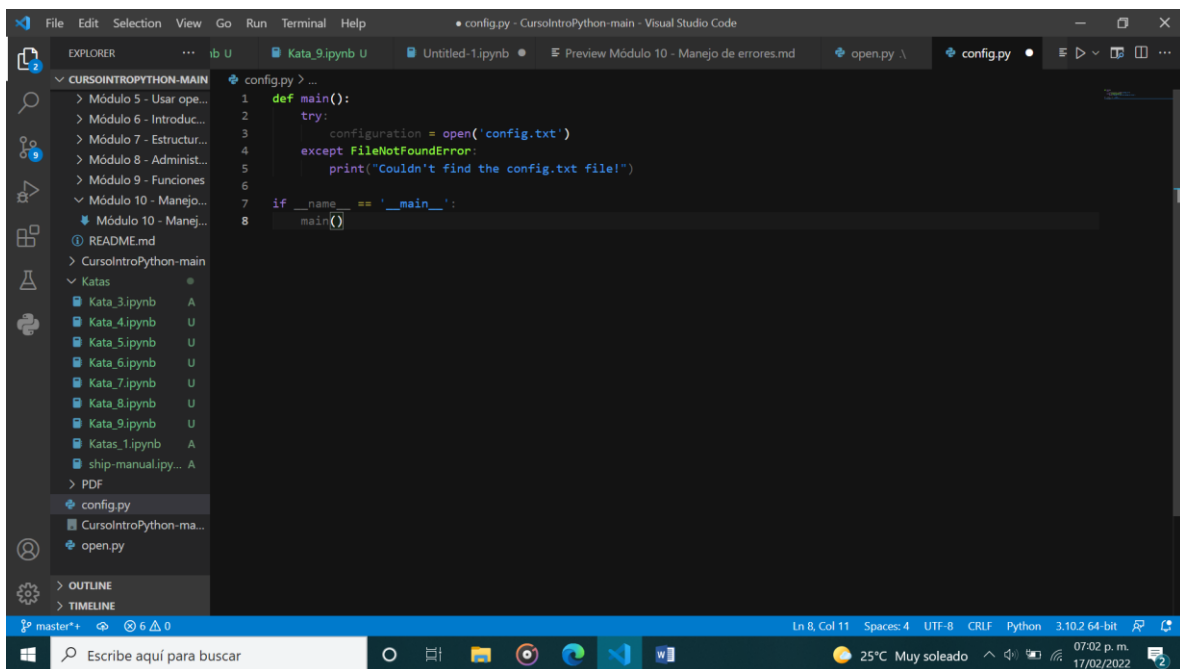


## 2. Ejecutar en el terminal.

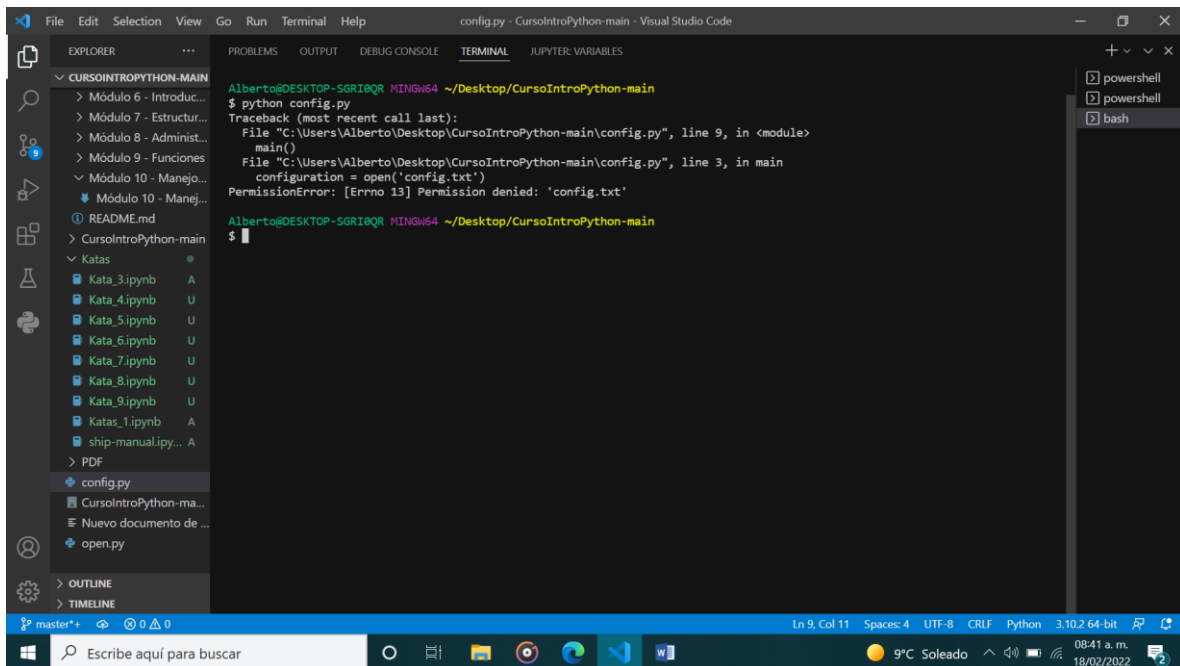


# Controlando las excepciones

## 1. Crear un archivo de Python con el nombre config.py



2. A continuación, se quita, el archivo *config.txt* y crear un directorio denominado *config.txt*. Se llama al archivo *config.py* para ver un error nuevo que debería ser similar al siguiente:

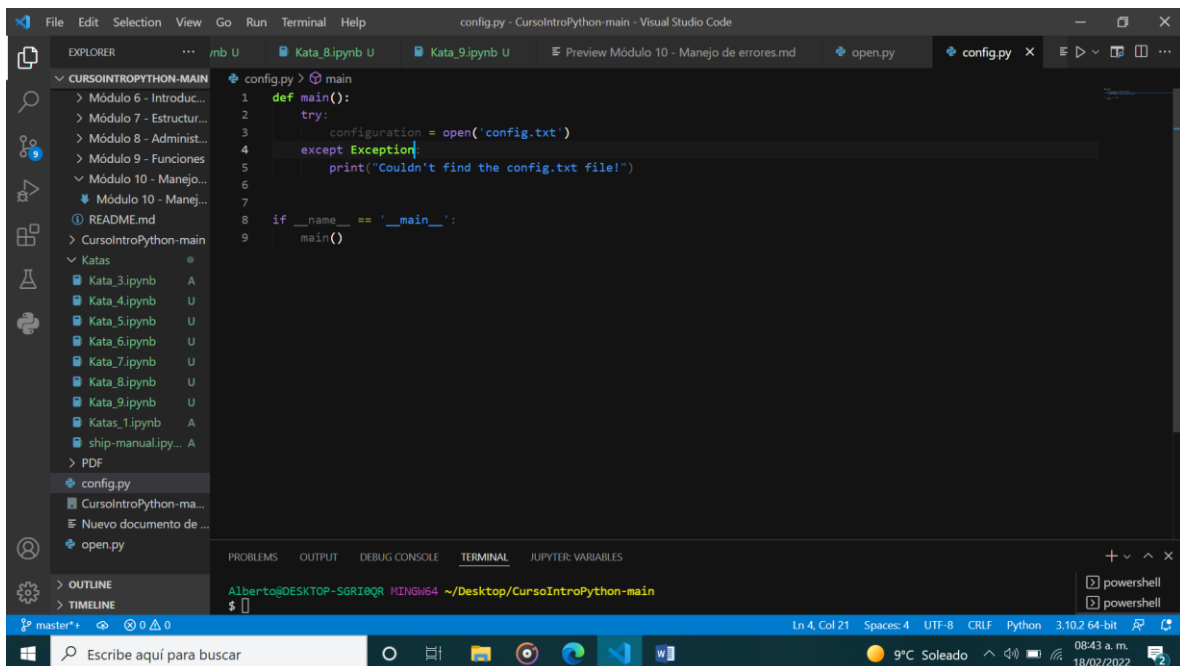


The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the project structure. The Terminal panel at the bottom shows the following output:

```
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$ python config.py
Traceback (most recent call last):
  File "C:\Users\Alberto\Desktop\CursoIntroPython-main\config.py", line 9, in <module>
    main()
  File "C:\Users\Alberto\Desktop\CursoIntroPython-main\config.py", line 3, in main
    configuration = open('config.txt')
PermissionError: [Errno 13] Permission denied: 'config.txt'

Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$
```

3. Actualizar función *main()*:



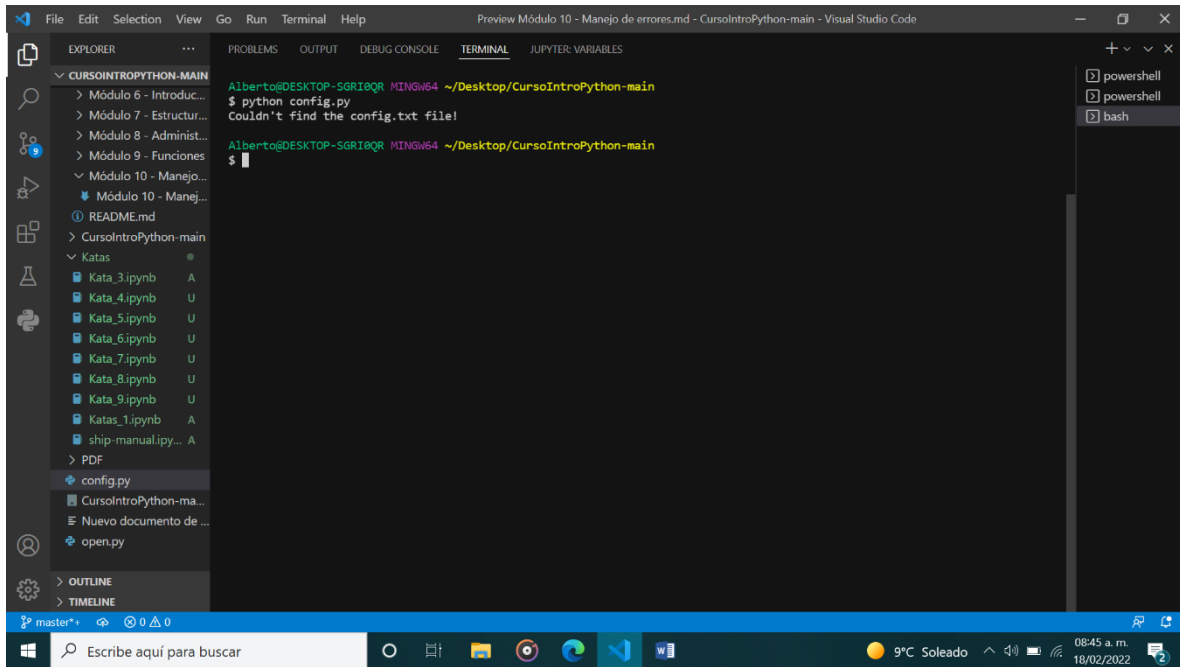
The screenshot shows the Visual Studio Code interface with the *config.py* file open. The code is as follows:

```
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()
```

The terminal panel at the bottom shows the command prompt ready for input:

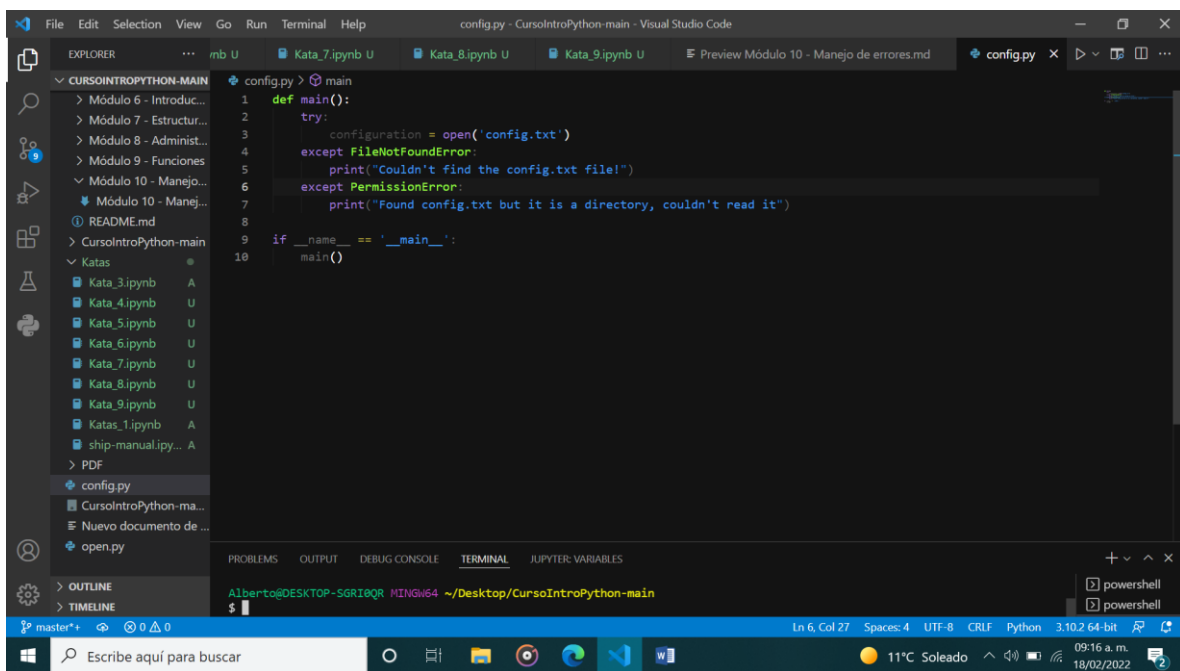
```
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$
```

4. Ahora se vuelve a ejecutar el código en el mismo lugar donde existe el archivo *config.txt* con permisos incorrectos:



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the file structure of 'CURSORINTROPYTHON-MAIN'. The file 'config.py' is highlighted. The Terminal panel at the bottom shows the command prompt 'Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main' and the command '\$ python config.py'. The output of the command is 'Couldn't find the config.txt file!'. The status bar at the bottom indicates the file is on the 'master' branch.

5. Se corrige este fragmento de código para abordar todas estas frustraciones. Revierte la detección de *FileNotFoundError* y luego se agrega otro bloque *except* para detectar *PermissionError*:

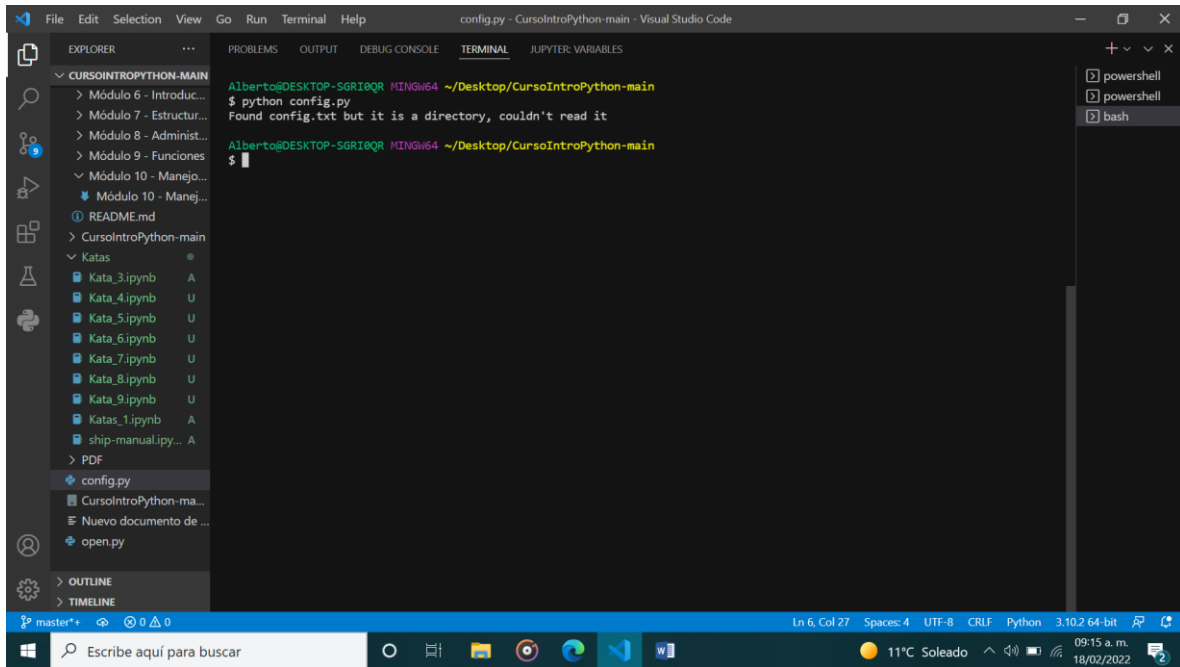


The screenshot shows the Visual Studio Code editor with the file 'config.py' open. The code is as follows:

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

The Explorer panel on the left shows the file structure, and the Terminal panel at the bottom shows the command prompt 'Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main'.

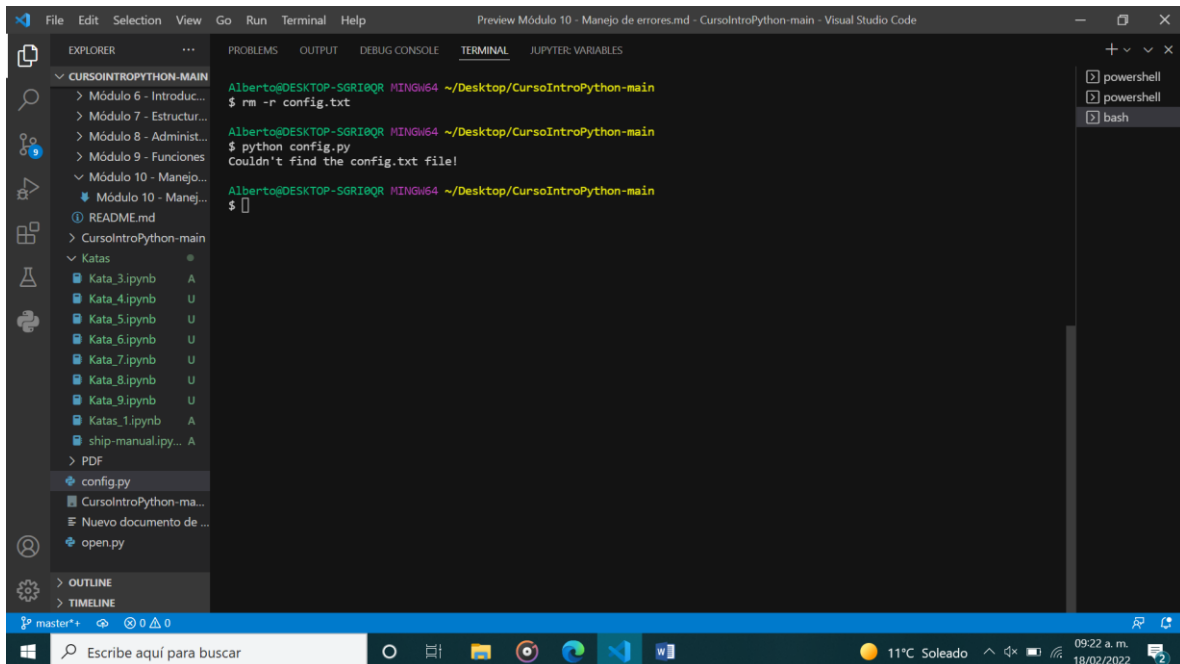
6. Se vuelve a ejecutar el código en el mismo lugar donde existe el archivo *config.txt* con permisos incorrectos:



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the file structure of 'CURSORINTROPYTHON-MAIN'. The file 'config.py' is selected. The Terminal panel on the right shows the following output:

```
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$ python config.py
Found config.txt but it is a directory, couldn't read it
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$
```

7. Eliminar el archivo *config.txt* para asegurar de que se alcanza el primer bloque except:



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the file structure of 'CURSORINTROPYTHON-MAIN'. The file 'config.py' is selected. The Terminal panel on the right shows the following output:

```
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$ rm -r config.txt
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$ python config.py
Couldn't find the config.txt file!
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$
```

## Generación de excepciones

1. Los astronautas limitan su uso de agua a unos 11 litros al día. Crear una función que, con base al número de astronautas, pueda calcular la cantidad de agua quedará después de un día o más:

The screenshot shows the JupyterLab interface. The left sidebar contains a file explorer with a tree view of the project structure, including folders like 'CURSORINTROPYTHON-MAIN' and 'Katas', and files like 'README.md' and 'config.py'. The main area displays a Python notebook with a code cell containing a function definition:

```
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"
```

Below the code cell, the output shows a green checkmark and the text '0.1s'. The bottom panel contains a terminal window with the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER VARIABLES
Couldn't find the config.txt file!
Alberto@DESKTOP-SGRI0QR MINGW64 ~/Desktop/CursoIntroPython-main
$
```

The terminal window also shows a list of available shells: powershell, powershell, and bash.

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

The screenshot displays the Visual Studio Code editor with a Jupyter Notebook open. The notebook is titled 'Untitled-1.ipynb' and is part of a workspace named 'CursoIntroPython-main'. The left sidebar shows the file explorer with a directory structure including 'CursoIntroPython-main', 'Katas', and 'PDF'. The main editor area shows the notebook content, which includes a function definition for 'water\_left' and its execution. The function calculates the total water left after a certain number of days based on daily usage and initial water left. The execution output shows the function being called with arguments (5, 100, 2) and returning the string 'Total water left after 2 days is: -10 liters'. The bottom panel shows the terminal with an error message: 'Couldn't find the config.txt file!'. The status bar at the bottom indicates the Jupyter Server is local and the current cell is 2 of 2.

Visual Studio Code interface showing a Jupyter Notebook for 'CursoIntroPython-main'.

The notebook contains the following code:

```
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"
```

The code is executed, and the output is displayed:

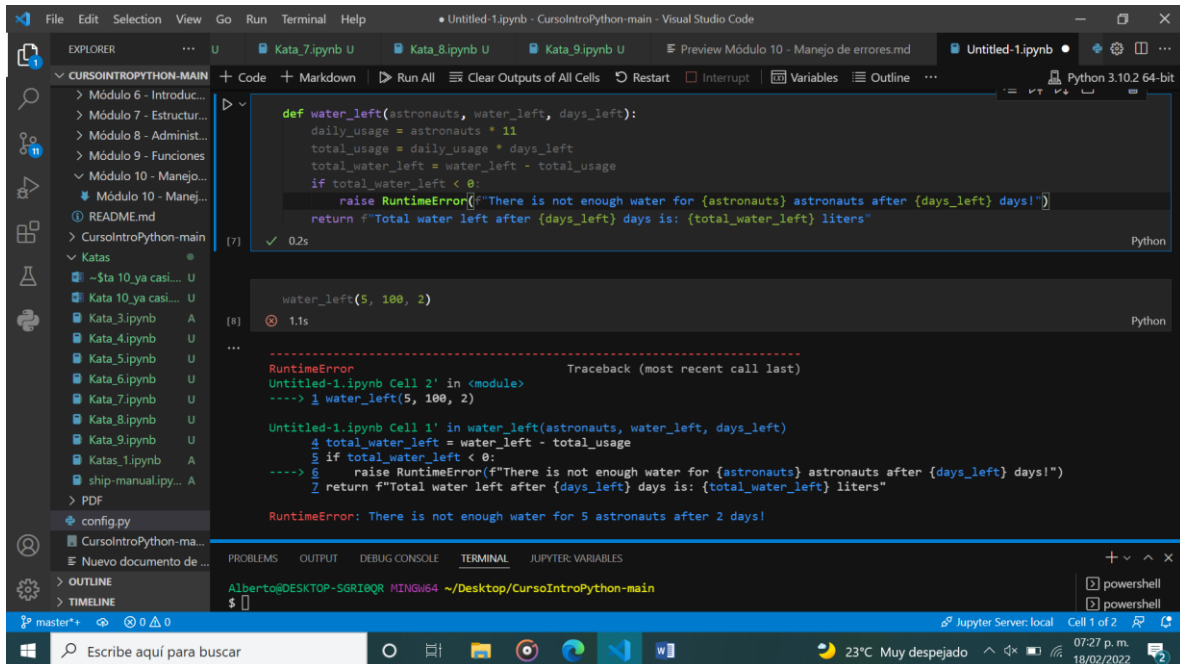
```
water_left(5, 100, 2)
... 'Total water left after 2 days is: -10 liters'
```

The terminal shows an error message:

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

The status bar at the bottom indicates the Jupyter Server is local and the current cell is 2 of 2.

3. Una carencia en los litros sería un error, se puede generar una excepción en la función `water_left()` para alertar de la condición de error:

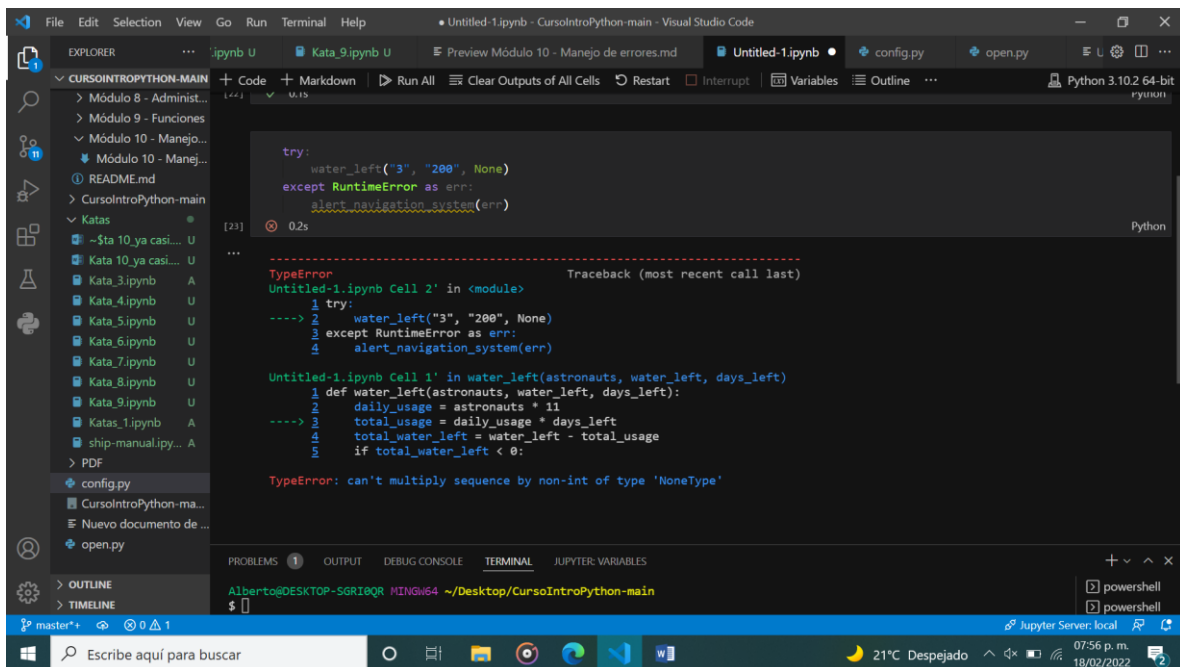


```
def water_left(astronauts, water_left, days_left):
    daily_usage = astronauts * 11
    total_usage = daily_usage * days_left
    total_water_left = water_left - total_usage
    if total_water_left < 0:
        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
    return f"Total water left after {days_left} days is: {total_water_left} liters"

water_left(5, 100, 2)
```

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

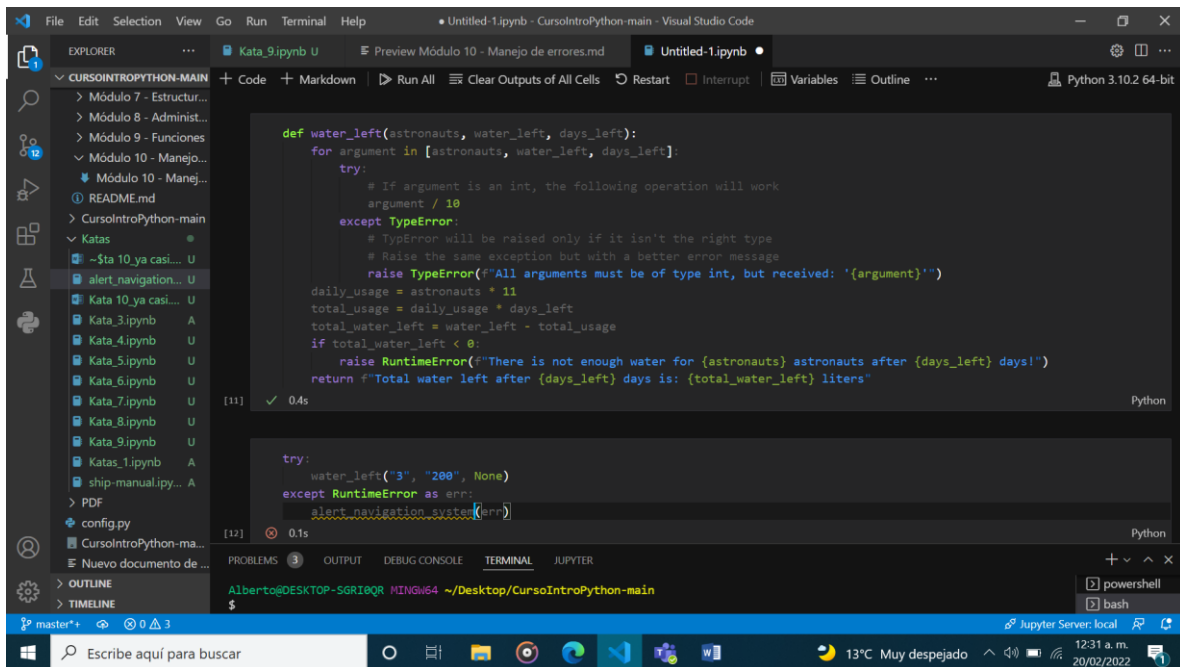
4. En el sistema de navegación, el código para señalar la alerta ahora puede usar `RuntimeError` para generar la alerta:



```
try:
    water_left("3", "200", None)
except RuntimeError as err:
    alert_navigation_system(err)
```

```
TypeError: can't multiply sequence by non-int of type 'NoneType'
```

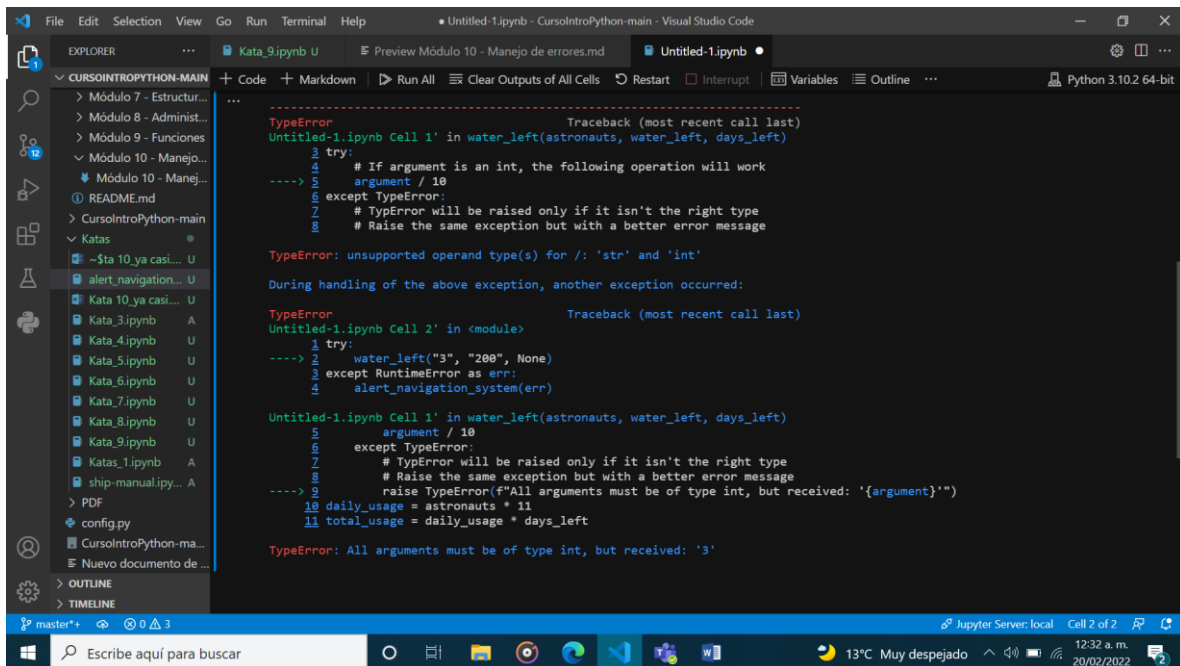
## 5. Actualizar la función para que use TypeError, pero con un mejor mensaje:



The screenshot shows the Visual Studio Code editor with a file named 'Untitled-1.ipynb' open. The code defines a function `water_left` that takes three arguments: `astronauts`, `water_left`, and `days_left`. It uses a `try` block to attempt a division `argument / 10`. If a `TypeError` occurs, it raises a custom `TypeError` with the message: `raise TypeError(f'All arguments must be of type int, but received: '{argument}')`. The function calculates `daily_usage = astronauts * 11`, `total_usage = daily_usage * days_left`, and `total_water_left = water_left - total_usage`. It then checks if `total_water_left < 0`. If true, it raises a `RuntimeError` with the message: `raise RuntimeError(f'There is not enough water for {astronauts} astronauts after {days_left} days!')`. Finally, it returns a string: `return f'Total water left after {days_left} days is: {total_water_left} liters'`. Below the code, there are two execution cells. Cell [11] shows the function definition being executed successfully. Cell [12] shows the function being called with `water_left("3", "200", None)`, which raises a `TypeError` that is caught by `alert_navigation_system(err)`. The terminal at the bottom shows the command prompt with the file path `~/Desktop/CursoIntroPython-main`.

```
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}')
```

## 6. Se ejecuta para obtener un mejor error:



The screenshot shows the Visual Studio Code editor with the same 'Untitled-1.ipynb' file. The code is the same as in the previous screenshot. Below the code, there are two execution cells. Cell [1] shows the function definition being executed successfully. Cell [2] shows the function being called with `water_left("3", "200", None)`, which raises a `TypeError` that is caught by `alert_navigation_system(err)`. The terminal at the bottom shows the command prompt with the file path `~/Desktop/CursoIntroPython-main`. The error messages are displayed in the output area, showing the traceback for the `TypeError` and the `RuntimeError`.

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
During handling of the above exception, another exception occurred:

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