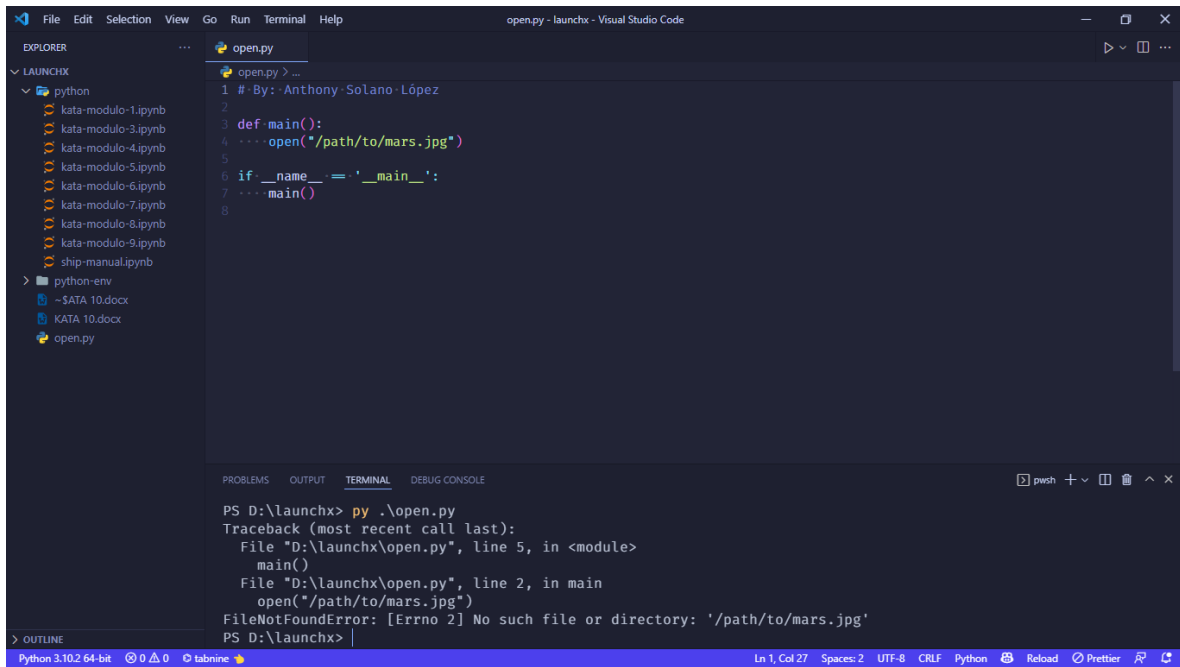


KATA 10, BY: ANTHONY SOLANO LÓPEZ

1. El archivo no existe, pero sin el error controlado



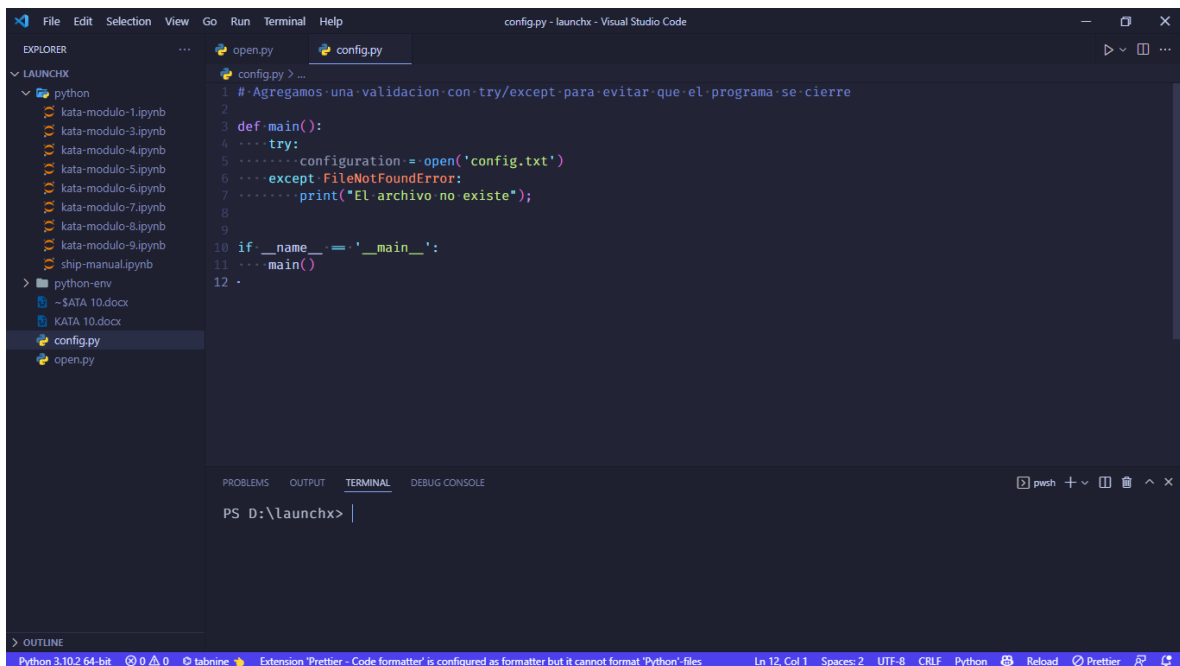
The screenshot shows the Visual Studio Code interface with a file explorer on the left containing a directory named 'LAUNCHX' with a 'python' subdirectory. The main editor displays 'open.py' with the following code:

```
1 # By: Anthony Solano López
2
3 def main():
4     open("/path/to/mars.jpg")
5
6 if __name__ == '__main__':
7     main()
8
```

The terminal at the bottom shows the command `PS D:\launchx> py .\open.py` and the resulting error:

```
Traceback (most recent call last):
  File "D:\launchx\open.py", line 5, in <module>
    main()
  File "D:\launchx\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:\launchx>
```

2. Agregando una excepción para controlar el error en caso de que no exista el archivo

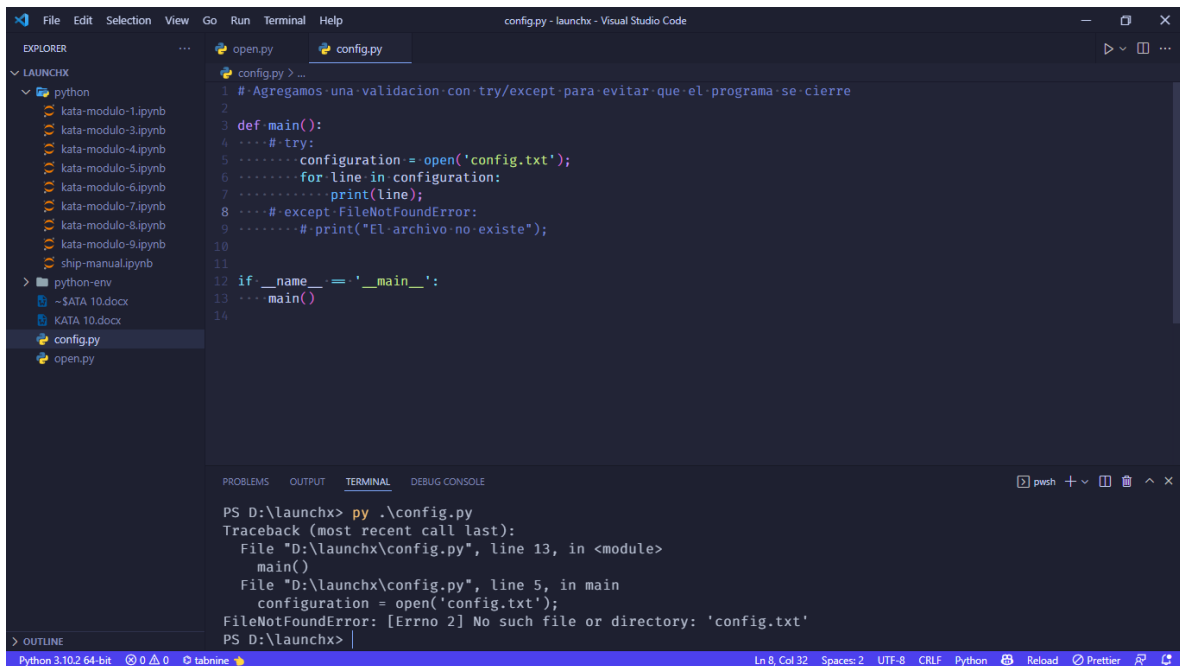


The screenshot shows the Visual Studio Code interface with a file explorer on the left containing a directory named 'LAUNCHX' with a 'python' subdirectory. The main editor displays 'config.py' with the following code:

```
1 # Agregamos una validacion con try/except para evitar que el programa se cierre
2
3 def main():
4     try:
5         configuration = open('config.txt')
6     except FileNotFoundError:
7         print("El archivo no existe");
8
9
10 if __name__ == '__main__':
11     main()
12
```

The terminal at the bottom shows the command `PS D:\launchx>` and the status bar at the bottom indicates that the Prettier extension is configured as a formatter but cannot format Python files.

3. A continuación, quita,ps el archivo config.txt y creamos un directorio denominado config.txt. Intentaremos llamar al archivo config.py para ver el error.



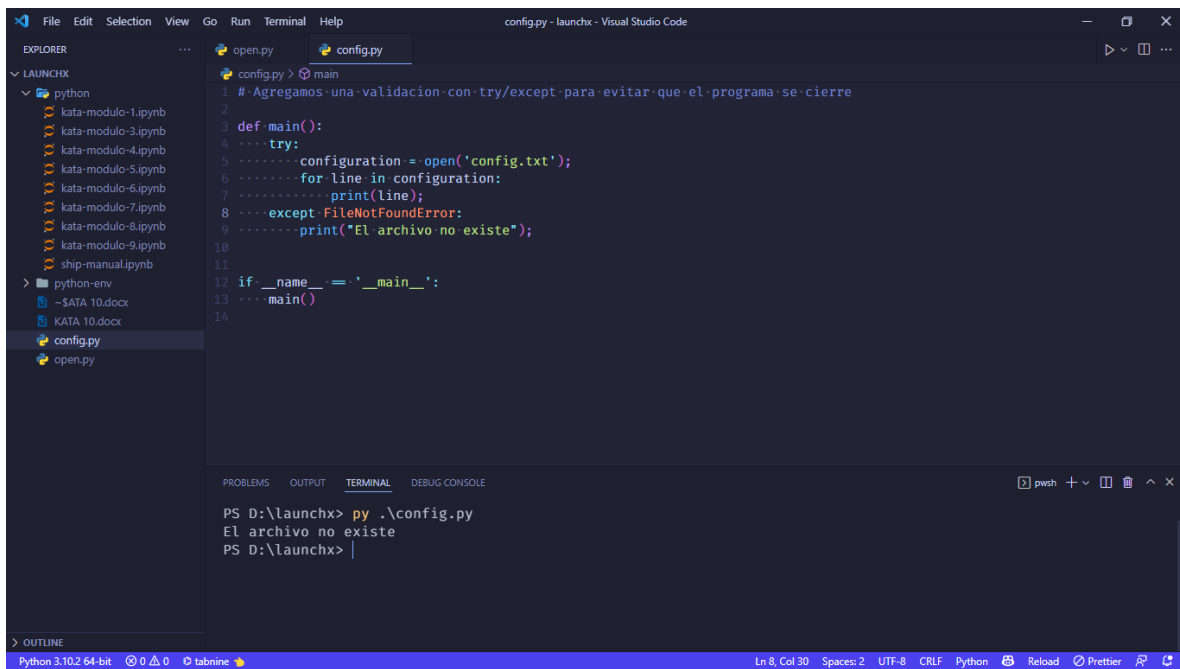
The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project structure under 'LAUNCHX'. The file 'config.py' is selected. The editor shows the following code:

```
1 #Agregamos una validacion con try/except para evitar que el programa se cierre
2
3 def main():
4     try:
5         configuration = open('config.txt');
6         for line in configuration:
7             print(line);
8     except FileNotFoundError:
9         print("El archivo no existe");
10
11
12 if __name__ == '__main__':
13     main()
14
```

The terminal at the bottom shows the command `PS D:\launchx> py .\config.py` and the resulting traceback:

```
Traceback (most recent call last):
  File "D:\launchx\config.py", line 13, in <module>
    main()
  File "D:\launchx\config.py", line 5, in main
    configuration = open('config.txt');
FileNotFoundError: [Errno 2] No such file or directory: 'config.txt'
PS D:\launchx>
```

4. 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():



The screenshot shows the Visual Studio Code interface with the file explorer on the left. The file 'config.py' is selected. The editor shows the following code:

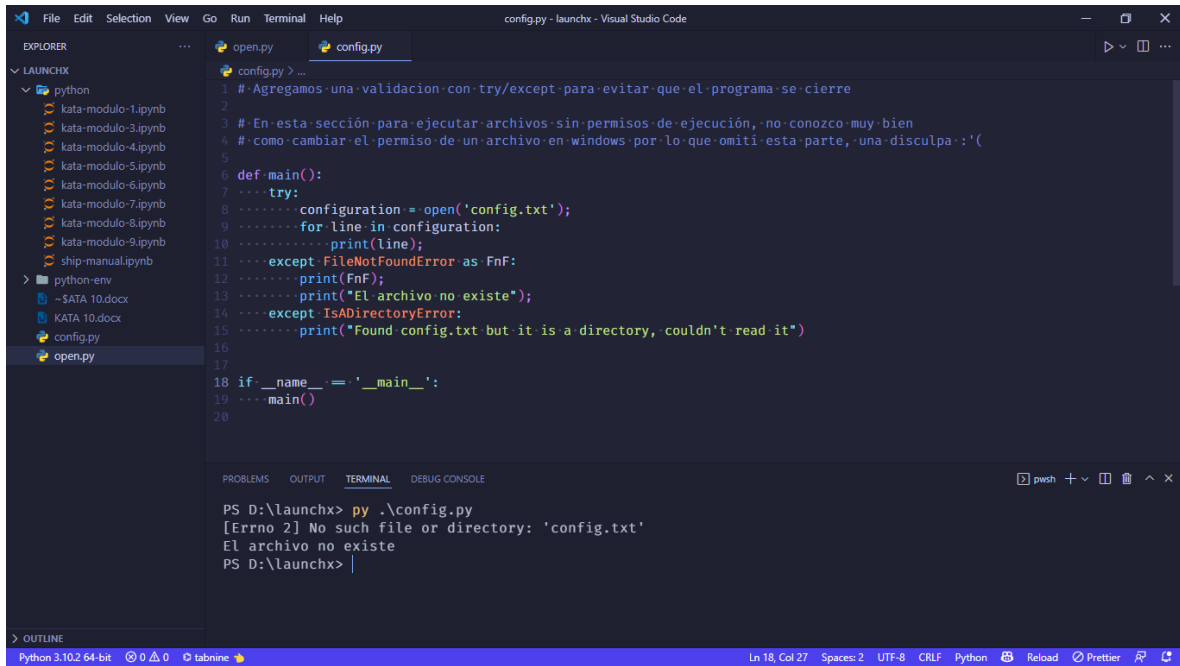
```
1 #Agregamos una validacion con try/except para evitar que el programa se cierre
2
3 def main():
4     try:
5         configuration = open('config.txt');
6         for line in configuration:
7             print(line);
8     except FileNotFoundError:
9         print("El archivo no existe");
10
11
12 if __name__ == '__main__':
13     main()
14
```

The terminal at the bottom shows the command `PS D:\launchx> py .\config.py` and the output:

```
El archivo no existe
PS D:\launchx>
```

```
# En esta sección para ejecutar archivos sin permisos de
ejecución, no conozco muy bien
# como cambiar el permiso de un archivo en windows por lo que
omití esta parte, una disculpa :'(
```

5. Ahora sí, eliminamos el archivo config.txt y ejecutamos nuevamente el código, como podemos ver, nos da el siguiente error, pero ya controlado

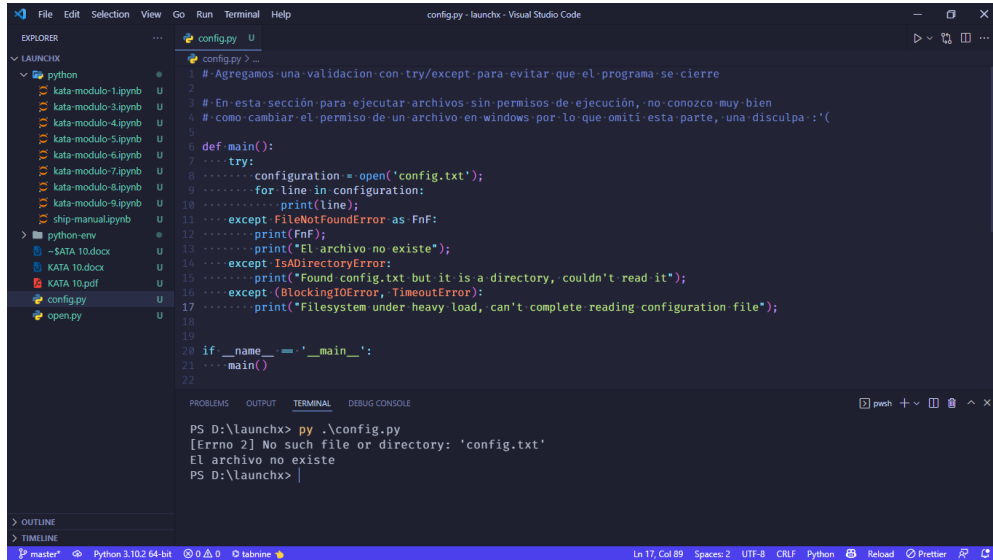


The screenshot shows the Visual Studio Code interface with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'LAUNCHX' with a 'python' folder containing several files. The code editor shows a file named 'config.py' with the following code:

```
1 # Agregamos una validacion con try/except para evitar que el programa se cierre
2
3 # En esta sección para ejecutar archivos sin permisos de ejecución, no conozco muy bien
4 # como cambiar el permiso de un archivo en windows por lo que omití esta parte, una disculpa :'(
5
6 def main():
7     try:
8         configuration = open('config.txt');
9         for line in configuration:
10             print(line);
11     except FileNotFoundError as FnF:
12         print(FnF);
13         print("El archivo no existe");
14     except IsADirectoryError:
15         print("Found config.txt but it is a directory, couldn't read it")
16
17
18 if __name__ == '__main__':
19     main()
20
```

The terminal at the bottom shows the command 'py .\config.py' being executed, resulting in the error message: '[Errno 2] No such file or directory: 'config.txt''. The error message is displayed in the terminal window, which is titled 'TERMINAL'.

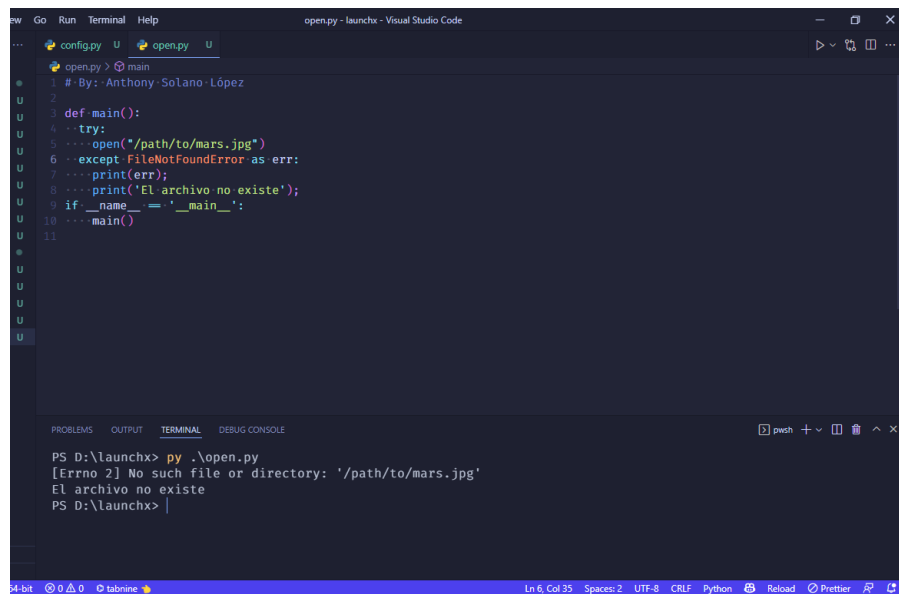
6. Cuando los errores son de una naturaleza similar y no es necesario controlarlos individualmente, puedes agrupar las excepciones como una usando paréntesis en la línea except. Por ejemplo, si el sistema de navegación está bajo cargas pesadas y el sistema de archivos está demasiado ocupado, tiene sentido detectar `BlockingIOError` y `TimeoutError` juntos:



```
config.py 2 ...
1 # Agregamos una validacion con try/except para evitar que el programa se cierre
2
3 # En esta sección para ejecutar archivos sin permisos de ejecución, no conozco muy bien
4 # como cambiar el permiso de un archivo en windows por lo que omiti esta parte, una disculpa :'(
5
6 def main():
7     try:
8         configuration = open('config.txt');
9         for line in configuration:
10             print(line);
11     except FileNotFoundError as FNF:
12         print(FNF);
13         print("El archivo no existe");
14     except IsADirectoryError:
15         print("Found config.txt but it is a directory, couldn't read it");
16     except (BlockingIOError, TimeoutError):
17         print("Filesystem under heavy load, can't complete reading configuration file");
18
19
20 if __name__ == '__main__':
21     main()
22
```

```
PS D:\launchx> py .\config.py
[Errno 2] No such file or directory: 'config.txt'
El archivo no existe
PS D:\launchx> |
```

7. Aunque puedes agrupar excepciones, solo debes hacerlo cuando no sea necesario controlarlas individualmente. Evita agrupar muchas excepciones para proporcionar un mensaje de error generalizado. 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:



```
open.py 1 ...
1 # By: Anthony Solano-López
2
3 def main():
4     try:
5         open("/path/to/mars.jpg")
6     except FileNotFoundError as err:
7         print(err);
8         print("El archivo no existe");
9
10 if __name__ == '__main__':
11     main()

```

```
PS D:\launchx> py .\open.py
[Errno 2] No such file or directory: '/path/to/mars.jpg'
El archivo no existe
PS D:\launchx> |
```

8. En este caso, `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 usar 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 diferenciarlas mediante el atributo `.errno`:



```
File Edit Selection View Go Run Terminal Help
config.py - launchx - Visual Studio Code

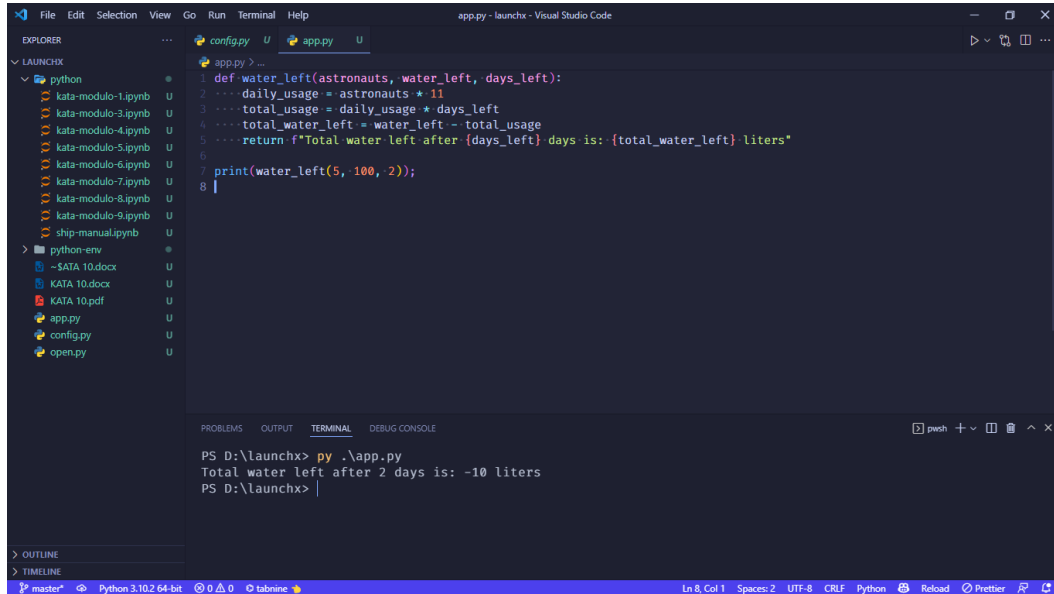
EXPLORER
└─ LAUNCHX
  └─ python
    ├── kata-modulo-1.ipynb
    ├── kata-modulo-3.ipynb
    ├── kata-modulo-4.ipynb
    ├── kata-modulo-5.ipynb
    ├── kata-modulo-6.ipynb
    ├── kata-modulo-7.ipynb
    ├── kata-modulo-8.ipynb
    ├── kata-modulo-9.ipynb
    ├── ship-manual.ipynb
    └─ python-env
      ├── ~$ATA 10.docx
      ├── KATA 10.docx
      ├── KATA 10.pdf
      └─ config.py
      └─ open.py

config.py
1
2
3
4
5
6 def main():
7     try:
8         configuration = open('config.txt');
9         for line in configuration:
10            print(line);
11
12    except OSError as err:
13        if err.errno == 2:
14            print("Couldn't find the config.txt file!")
15        elif err.errno == 13:
16            print("Found config.txt but couldn't read it")
17
18    # except FileNotFoundError as FnF:
19    #     print(FnF);
20    #     print("El archivo no existe");
21    # except IsADirectoryError:
22    #     print("Found config.txt but it is a directory, couldn't read it");
23    # except (BlockingIOError, TimeoutError):
24    #     print("Filesystem under heavy load, can't complete reading configuration file");
25
26 if __name__ == '__main__':
27     main()

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\launchx> py .\config.py
Couldn't find the config.txt file!
PS D:\launchx>
```

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:

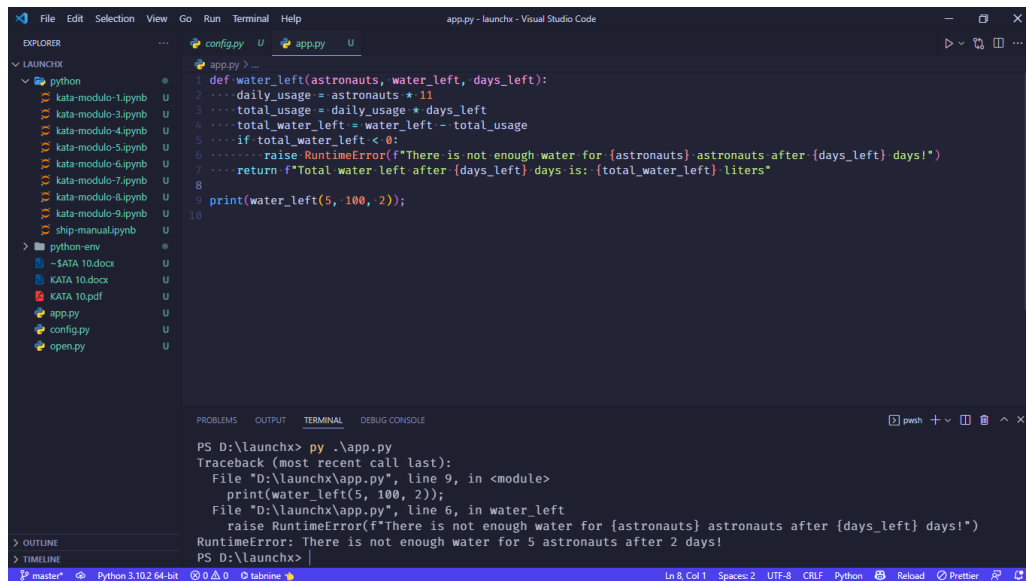


The screenshot shows the Visual Studio Code interface with a file explorer on the left containing a 'python' folder with several '.ipynb' files and a 'python-env' folder. The main editor displays a file named 'app.py' with the following code:

```
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     return f"Total water left after {days_left} days is: {total_water_left} liters"
6
7 print(water_left(5, 100, 2));
8
```

The terminal at the bottom shows the command 'py .\app.py' being executed, resulting in the output: 'Total water left after 2 days is: -10 liters'.

2. 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:



The screenshot shows the Visual Studio Code interface with the same file explorer. The main editor displays a modified 'app.py' file:

```
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:
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"
8
9 print(water_left(5, 100, 2));
10
```

The terminal shows the command 'py .\app.py' being executed, which results in a `RuntimeError` exception being raised. The output in the terminal is:

```
PS D:\launchx> py .\app.py
Traceback (most recent call last):
  File "D:\launchx\app.py", line 9, in <module>
    print(water_left(5, 100, 2));
  File "D:\launchx\app.py", line 6, 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!
PS D:\launchx>
```

Podemos ver un error de en tiempo de ejecución.

```
File Edit Selection View Go Run Terminal Help
app.py - launchx - Visual Studio Code

EXPLORER
  LAUNCHX
    python
      kata-modulo-1.ipynb
      kata-modulo-3.ipynb
      kata-modulo-4.ipynb
      kata-modulo-5.ipynb
      kata-modulo-6.ipynb
      kata-modulo-7.ipynb
      kata-modulo-8.ipynb
      kata-modulo-9.ipynb
      ship-manual.ipynb
    python-env
      ~$ATA 10.docx
      KATA 10.docx
      KATA 10.pdf
      app.py
      config.py
      open.py

app.py
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:
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"
8
9     try:
10    """print('water_left(5, 100, 2)');
11    """except RuntimeError as err:
12    """print(err);
13

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\launchx> py .\app.py
water_left(5, 100, 2)
PS D:\launchx> |
```

3. La función `water_left()` también se puede actualizar para evitar el paso de tipos no admitidos. Intenten pasar argumentos que no sean enteros para comprobar la salida de error:

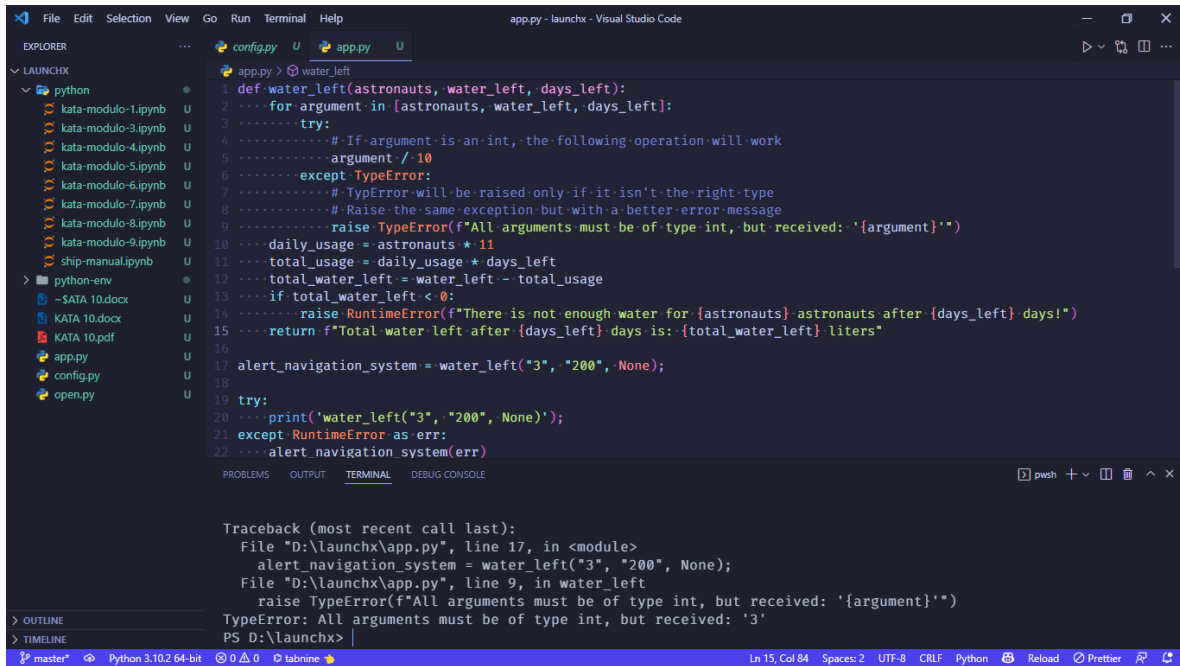
```
File Edit Selection View Go Run Terminal Help
app.py - launchx - Visual Studio Code

EXPLORER
  LAUNCHX
    python
      kata-modulo-1.ipynb
      kata-modulo-3.ipynb
      kata-modulo-4.ipynb
      kata-modulo-5.ipynb
      kata-modulo-6.ipynb
      kata-modulo-7.ipynb
      kata-modulo-8.ipynb
      kata-modulo-9.ipynb
      ship-manual.ipynb
    python-env
      ~$ATA 10.docx
      KATA 10.docx
      KATA 10.pdf
      app.py
      config.py
      open.py

app.py
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:
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"
8
9 alert_navigation_system = water_left("3", "200", None);
10
11 try:
12    """print('water_left("3", "200", None)');
13    """except RuntimeError as err:
14    """alert_navigation_system(err)
15

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\launchx> py .\app.py
Traceback (most recent call last):
  File "D:\launchx\app.py", line 9, in <module>
    alert_navigation_system = water_left("3", "200", None);
  File "D:\launchx\app.py", line 3, in water_left
    total_usage = daily_usage * days_left
TypeError: can't multiply sequence by non-int of type 'NoneType'
PS D:\launchx> |
```

4. 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:



The screenshot shows the Visual Studio Code interface with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The code editor displays a Python script named `app.py` with the following content:

```
1 def water_left(astronauts, water_left, days_left):
2     for argument in [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
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
12    total_water_left = water_left - total_usage
13    if total_water_left < 0:
14        raise RuntimeError(f"There is not enough water for {astronauts} astronauts after {days_left} days!")
15    return f"Total water left after {days_left} days is: {total_water_left} liters"
16
17 alert_navigation_system = water_left("3", "200", None)
18
19 try:
20     print(water_left("3", "200", None))
21 except RuntimeError as err:
22     alert_navigation_system(err)
```

The terminal at the bottom shows the following traceback:

```
Traceback (most recent call last):
  File "D:\launchx\app.py", line 17, in <module>
    alert_navigation_system = water_left("3", "200", None);
  File "D:\launchx\app.py", line 9, 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'
PS D:\launchx>
```