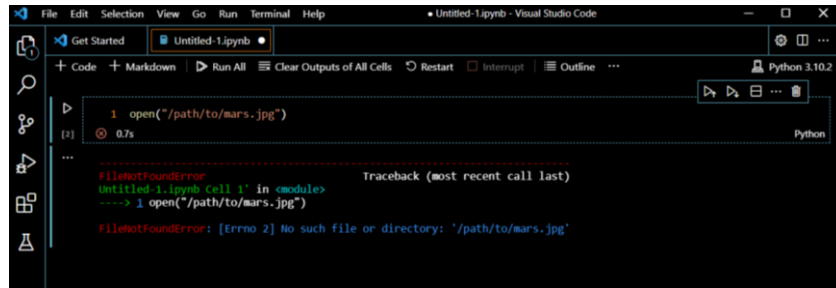


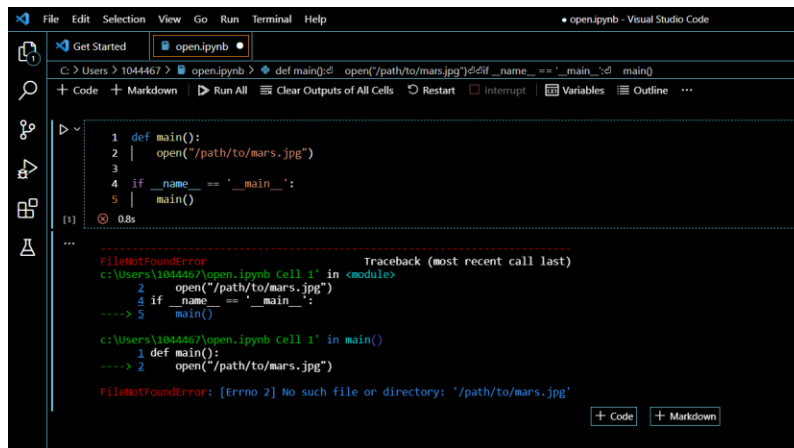
Tracebacks

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



```
1 open("/path/to/mars.jpg")
FileNotFoundError                                Traceback (most recent call last)
untitled-1.ipynb Cell 1: 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:



```
1 def main():
2     open("/path/to/mars.jpg")
3
4 if __name__ == '__main__':
5     main()
FileNotFoundError                                Traceback (most recent call last)
c:\Users\1044467\open.ipynb Cell 1: in <module>
2     open("/path/to/mars.jpg")
4 if __name__ == '__main__':
----> 5     main()
c:\Users\1044467\open.ipynb Cell 1: in main()
1 def main():
----> 2     open("/path/to/mars.jpg")
FileNotFoundError: [Errno 2] No such file or directory: '/path/to/mars.jpg'
```

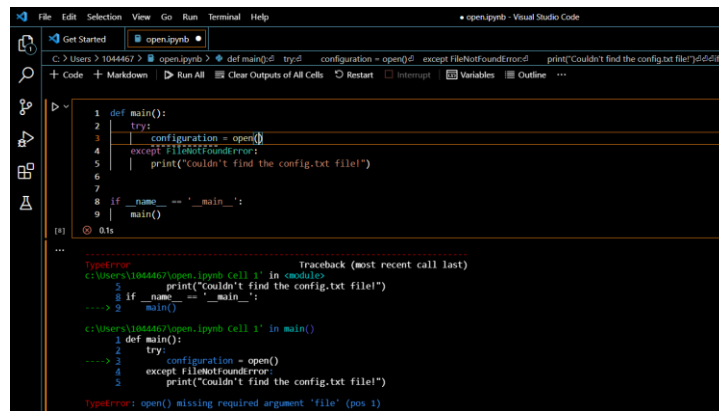
Try y Except de los bloques

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 presentan. 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`



```
1 try:
2     open('config.txt')
3 except FileNotFoundError:
4     print("couldn't find the config.txt file!")
5
... couldn't find the config.txt file!
```

Después de la palabra clave try, agregamos código que tenga la posibilidad de producir una excepción. A continuación, agregamos la palabra clave except junto con la posible excepción, seguida de cualquier código que deba ejecutarse cuando se produce esa condición. Puesto que config.txt no existe en el sistema, Python imprime que el archivo de configuración no está ahí. El bloque try y except, junto con un mensaje útil, evita un seguimiento y sigue informando al usuario sobre el problema.

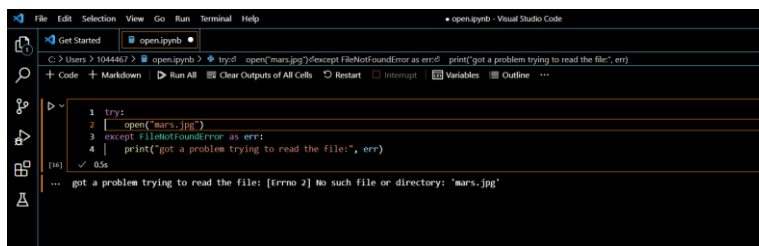


```
File Edit Selection View Go Run Terminal Help
open.py - Visual Studio Code
C:\Users\1044467> open.py
def main():
    try:
        configuration = open()
    except FileNotFoundError:
        print("couldn't find the config.txt file")
if __name__ == '__main__':
    main()

Traceback (most recent call last)
c:\Users\1044467\open.py:2: in <module>
    print("couldn't find the config.txt file")
2 if __name__ == '__main__':
----> 2     main()
c:\Users\1044467\open.py:1: in main()
    1 def main():
    2     try:
    3         configuration = open()
    4     except FileNotFoundError:
    5         print("couldn't find the config.txt file")
TypeError: open() missing required argument 'file' (pos 1)
```

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:



```
File Edit Selection View Go Run Terminal Help
open.py - Visual Studio Code
C:\Users\1044467> open.py
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'
```

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 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:



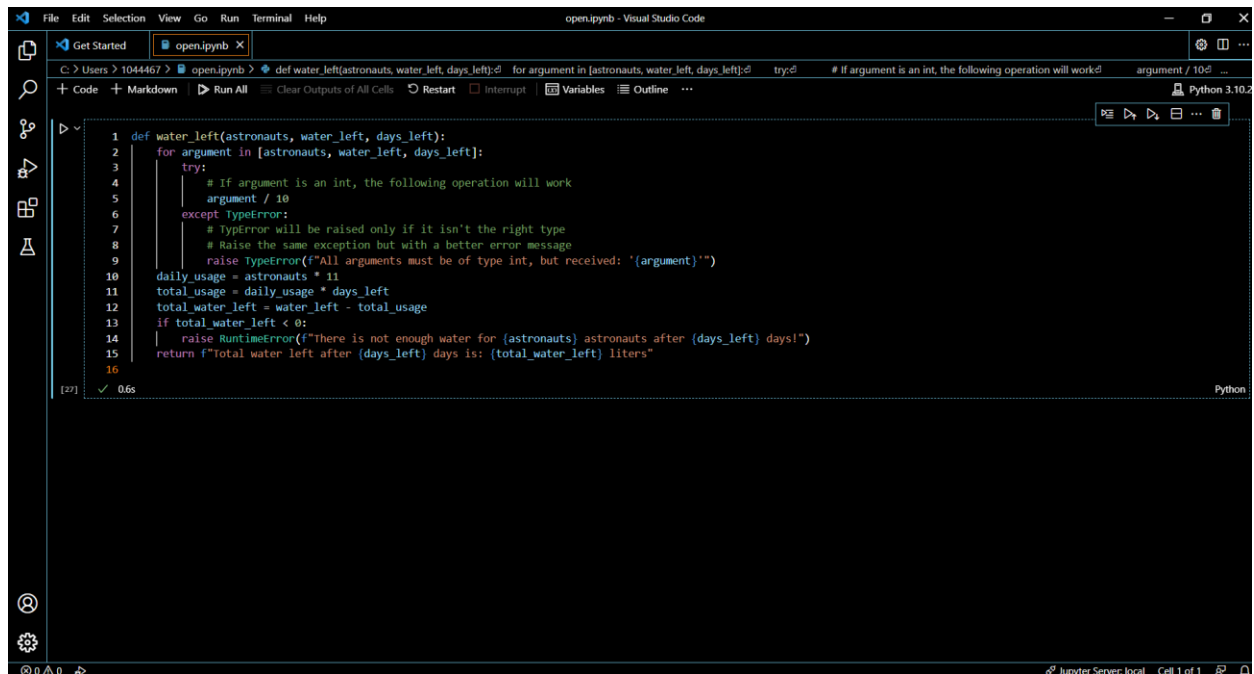
```
File Edit Selection View Go Run Terminal Help
open.py - Visual Studio Code
C:\Users\1044467> open.py
try:
    open("config.txt")
except OSError as err:
    if err.errno == 2:
        print("couldn't find the config.txt file")
    elif err.errno == 13:
        print("found config.txt but couldn't read it")

couldn't find the config.txt file
```

Generación de excepciones

La generación de excepciones también puede ayudar en la toma de decisiones para otro código. Como hemos visto antes, en función del error, el código puede tomar decisiones inteligentes para resolver, solucionar o ignorar un problema.

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:



```
File Edit Selection View Go Run Terminal Help
open.ipynb - Visual Studio Code

C > Users > 1044467 > open.ipynb > 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"

[27] ✓ 0.6s Python
```