# Catching Errors in file handling

Errors are inevitable. How to catch errors in python while reading or writing files

WE'LL COVER THE FOLLOWING

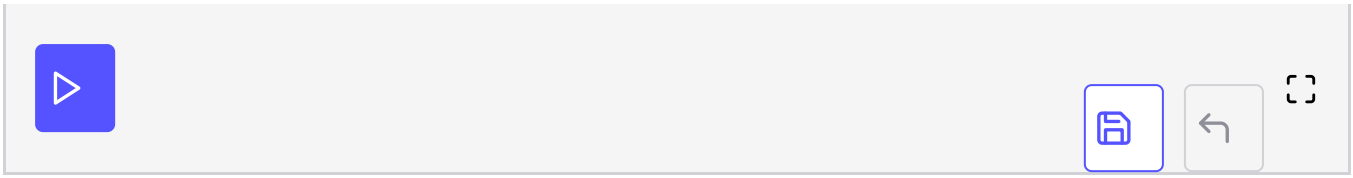


Sometimes when you are working with files, bad things happen. The file is locked because some other process is using it or you have some kind of permission error. When this happens, an **IOError** will probably occur. In this section, we will look at how to catch errors the normal way and how to catch them using the **with** operator. Hint: the idea is basically the same in both!

```
file_handler = None
try:
    file_handler = open("test2.txt")
    for line in file_handler:
        print(line)
except IOError:
    print("An IOError has occurred!")
finally:
    if file_handler is not None:
        file_handler.close()
```

In the example above, we wrap the usual code inside of a **try/except** construct. If an error occurs, we print out a message to the screen. Note that we also make sure we close the file using the **finally** statement. Now we're ready to look at how we would do this same thing using **with**:

```
try:
    with open("test2.txt") as file_handler:
        for line in file_handler:
            print(line)
except IOError:
    print("An IOError has occurred!")
```



As you might have guessed, we just wrapped the **with** block in the same way as we did in the previous example. The difference here is that we do not need the **finally** statement as the context manager handles that for us.

## Wrapping Up #

At this point you should be pretty well versed in dealing with files in Python. Now you know how to read and write files using the older style and the newer **with** style. You will most likely see both styles in the wild. In the next chapter, we will learn how to import other modules that come with Python. This will allow us to create programs using pre-built modules. Let's get started!