

Files and Exceptions

Michael C. Hackett

Assistant Professor, Computer Science

Community
College
of Philadelphia

Lecture Topics

- Writing data to text files
- Reading data from text files
- Appending data to text files
- Random Access
 - Reading data from text files.
 - Reading, Writing, and Appending Data
- Modifying Existing Files.
- Exception Handling
 - Try Clauses
 - Except Clauses
 - Else Clauses
 - Finally Clauses
- Handling Multiple Exceptions
- Raising Exceptions

Colors/Fonts

• Global Variable Names	—	Brown
• Local Variable Names	—	Lt Blue
• Literals	—	Blue
• Keywords	—	Orange
• Operators/Punctuation	—	Black
• Functions	—	Purple
• Parameters	—	Gold
• Comments	—	Gray
• Modules	—	Pink

Source Code	— Consolas
Output	— Courier New

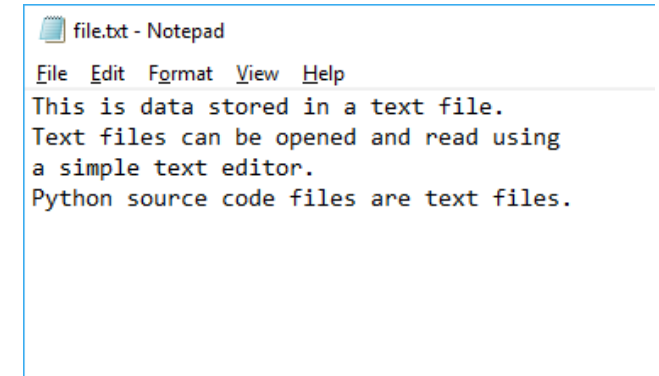
What are files?

- A **file** is an entity of digital information, typically kept on a long-term storage device.
 - Word documents, Powerpoint presentations, and PDFs are all examples of different types of files.
- A file has a name which normally includes an extension.
 - Textfile.**txt**
 - WordDocument.**doc**
 - You can have files without extensions.
 - Extensions are primarily used by the operating system, so it knows what program to use to open and read the file. Some programs will only accept files with certain extensions.

Types of Files

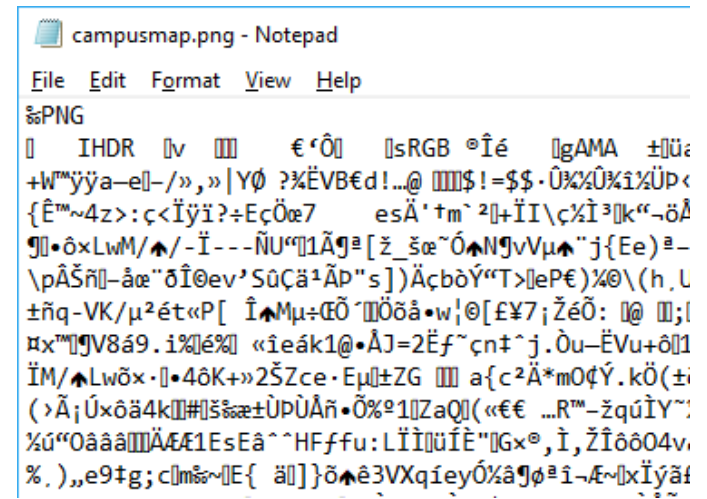
- Text Files

- The information contained in the file is ASCII plaintext.
- Can be opened in any text editor (like Notepad.)
- “Human readable”



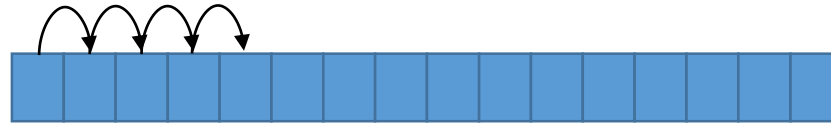
- Binary Files

- Files that are not stored in plaintext, like images and compiled programs.
- Normally cannot be opened in any text editor.
- Raw binary- “Computer readable”

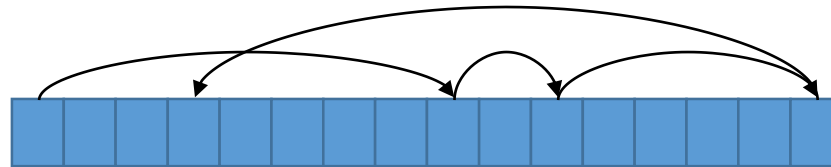


File Access

- Using ***sequential access***, data is read/accessed from the beginning of the file through the end of the file.



- Using ***random access***, data can be accessed from any location in the file. (See supplemental slides on File Access.)



Opening a File

- To open a file, use Python's built-in **open** function.
- The open function accepts two arguments: The file's name and the mode in which the file is being used.
 - Both arguments are strings.

```
my_text_file = open(filename, mode)
```

- The object returned by the open function is a file object.

Specifying the File's name/path

- If the file you wish to access is in the same folder as the Python program opening the file, you only need to supply the file's name.

```
my_text_file = open("file.txt", mode)
```

- If the file is in a subfolder, you'll need to supply the path to the file beginning with the subfolder's name.

```
my_text_file = open("subfolder\\subfolder2\\file.txt", mode)
```

- Remember, a backslash in a String literal indicates an escape sequence.

Specifying the File's name/path

- If the file is in an entirely different folder, you'll need to supply the full path to the file (beginning with the drive letter on Windows).

```
my_text_file = open("C:\\path\\to\\the\\file.txt", mode)
```

- We can prefix a string literal with the letter r to indicate a raw string.
 - Backslashes will be considered normal characters instead of part of an escape sequence.

```
my_text_file = open(r"C:\path\to\the\file.txt", mode)
```

Writing Data to a Text File

- Specifying "w" as the mode will open the file in write mode.

```
my_output_file = open("output.txt", "w")
```

- In write mode, data can be written to the file.
 - If the specified file ***does not*** already exist (you want to make a new file) Python will create it.
 - If the specified file ***does*** exist, its contents will be **ERASED**.

Saving a File

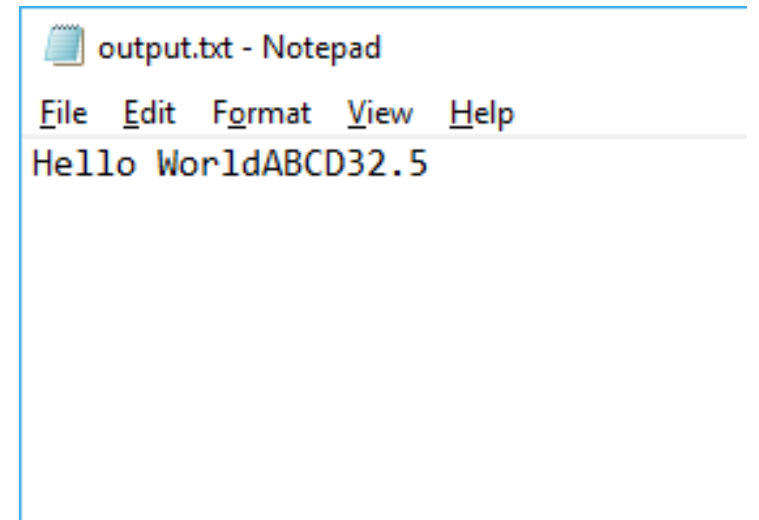
- To when you are finished writing to a file, call the file's **close** function.
- This saves the file.
 - If you do not close the file, the information you wrote will not be saved.

```
my_output_file.close()
```

Writing Data to a New Text File

- Once the file is open, we can write data to the file.
- A file's **write** function will write string values to the file.
 - If the data is numeric (ints or floats) be sure to convert the data to string form.

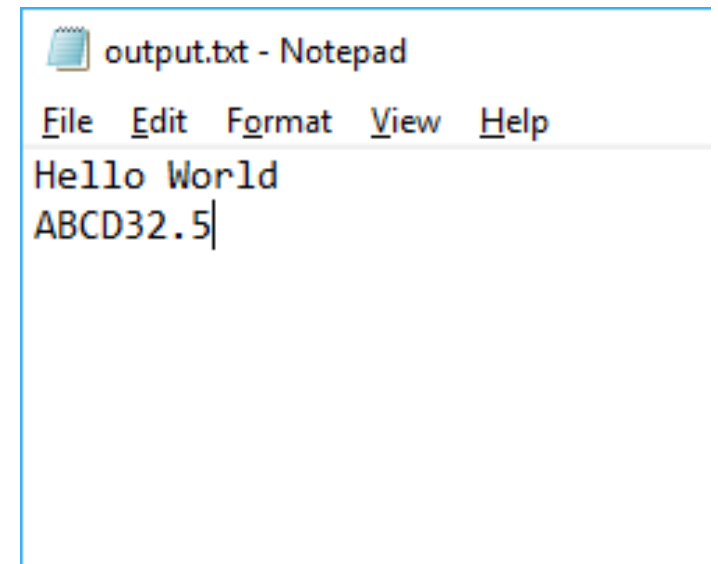
```
my_output_file = open("output.txt", "w")  
my_output_file.write("Hello World")  
my_output_file.write("ABCD")  
my_output_file.write(str(32.5))  
my_output_file.close()
```



Writing Data to a New Text File

- The write function does not add line feeds after each function call.
- To add line feeds, add (or concatenate) `\n` to the end of the line.

```
my_output_file = open("output.txt", "w")
my_output_file.write("Hello World\n")
my_output_file.write("ABCD")
my_output_file.write(str(32.5))
my_output_file.close()
```



Reading Data from a Text File

- Specifying "r" as the mode will open the file in read-only mode.

```
my_text_file = open("file.txt", "r")
```

- No data can be written to a file opened in read-only mode.

Closing a File

- To when you are finished reading a file, call the file's **close** function.
- Python can't have two instances of the same file open.
 - Always close your file when you are done reading from it.

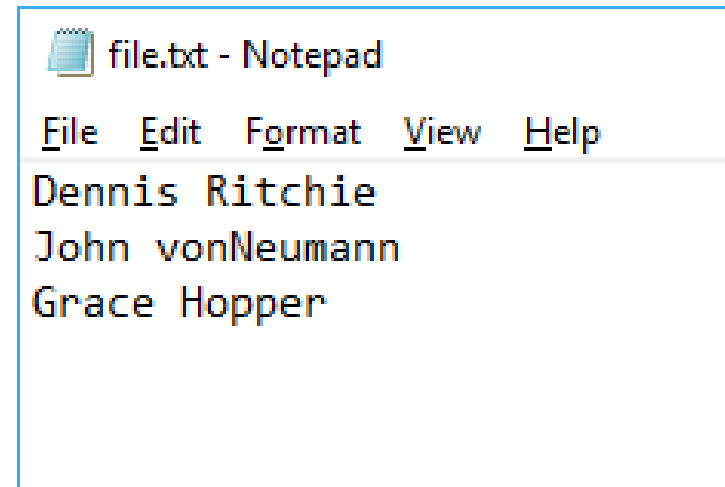
```
my_text_file.close()
```

Reading Data from a Text File

- Once the file is opened in read mode, we can read the contents of the file.
- To read a file, line-by-line, use the file's readline function.
 - The function will return a string containing the next line in the file.

```
my_text_file = open("file.txt", "r")  
line1 = my_text_file.readline()  
print(line1)  
my_text_file.close()
```

Dennis Ritchie



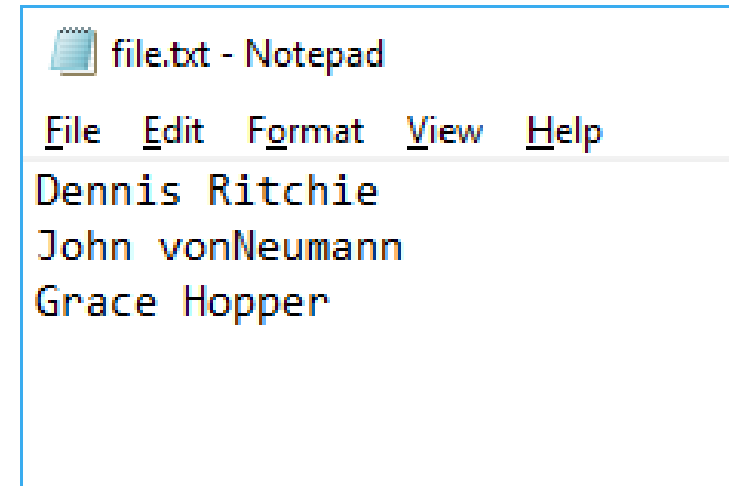
Reading Data from a Text File

```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline()
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

Dennis Ritchie

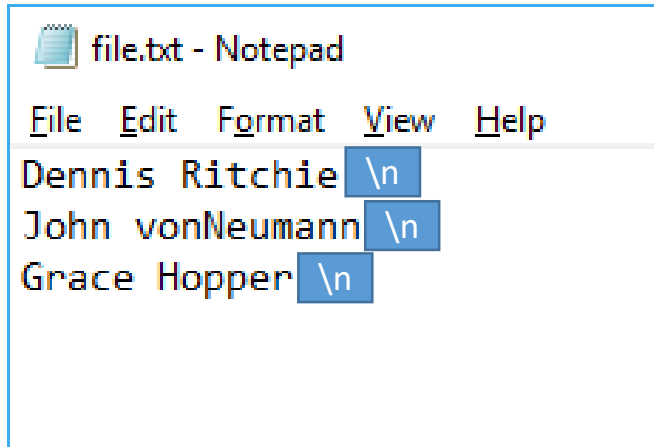
John vonNeumann

Grace Hopper



Reading Data from a Text File

- The extra lines are the result of the non-character line feed (`\n`) at the end of each line in the file.



```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline()
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

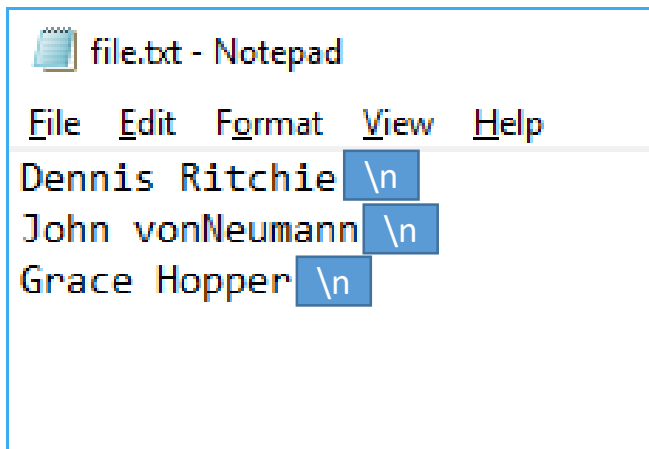
Dennis Ritchie

John vonNeumann

Grace Hopper

Reading Data from a Text File

- To strip away the line feed, we can use the string's **rstrip** function.



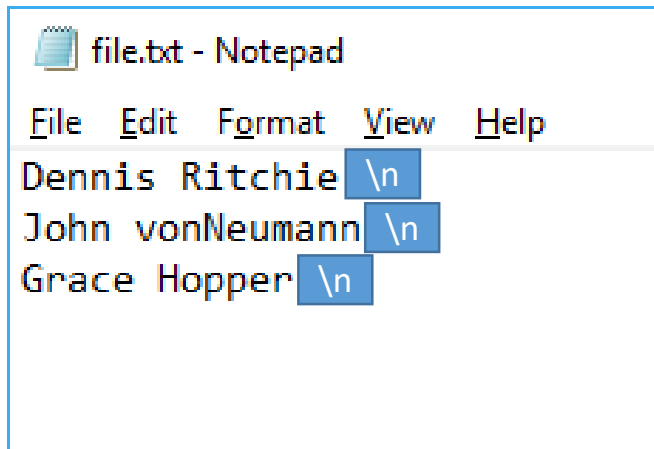
```
file.txt - Notepad
File Edit Format View Help
Dennis Ritchie \n
John vonNeumann \n
Grace Hopper \n
```

```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline().rstrip("\n")
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

Dennis Ritchie
John vonNeumann

Grace Hopper

Reading Data from a Text File



```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline().rstrip("\n")
line2 = my_text_file.readline().rstrip("\n")
line3 = my_text_file.readline().rstrip("\n")
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

```
Dennis Ritchie
John vonNeumann
Grace Hopper
```

Reading Data from a Text File

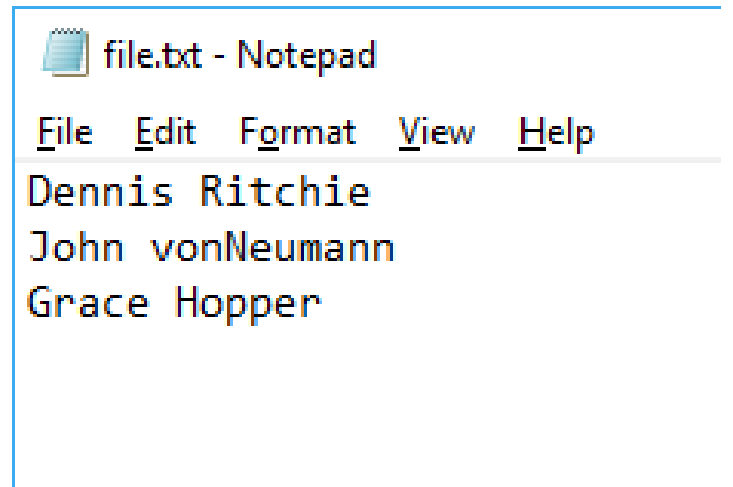
- A for loop can be used to read through a file sequentially.

```
my_text_file = open("file.txt", "r")
```

```
for line in my_text_file :  
    print(line.rstrip("\n"))
```

```
my_text_file.close()
```

```
Dennis Ritchie  
John vonNeumann  
Grace Hopper
```



Appending Data to a Text File

- Specifying "a" as the mode will open the file in append mode.

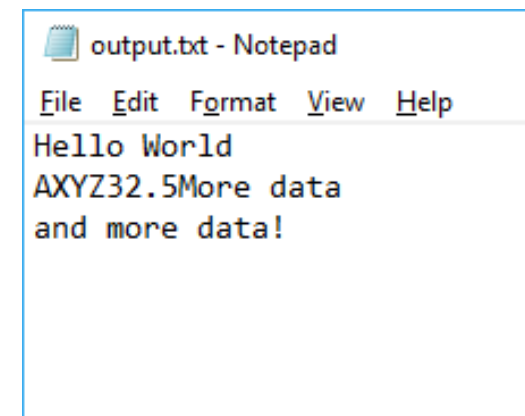
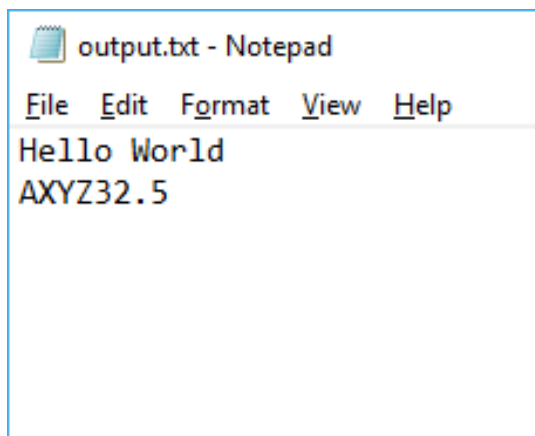
```
my_existing_file = open("output.txt", "a")
```

- In append mode, data can be written to a new or existing file.
 - If the file does not already exist, Python will create it.
 - If the file does exist, the file will be opened and wait for more data to be written to the end of the file.
- Be sure to close the file when you are finished appending to it.

Appending Data to a Text File

- Once the file is open, we can continue writing data to the file.

```
my_output_file = open("output.txt", "a")  
my_output_file.write("More data\n")  
my_output_file.write("and more data!")  
my_output_file.close()
```



Exception Handling

- An ***exception*** is an object that is generated as the result of an error or an unexpected event.
 - When that happens in Python, we say that an exception has been *raised*.
- It is the programmer's responsibility to write code that detects and handles exceptions.

Exception Handling

- Unhandled exceptions cause an error message to be printed and will immediately stop a program.
- The example code below attempts to open and read a file that does not exist.
 - The open function raises an exception (FileNotFoundError) if the specified file isn't found.

```
my_text_file = open("file78.txt", "r")
```

```
Traceback (most recent call last):  
  File "C:\testing\examples.py", line 8, in <module>  
    myTextFile = open("file78.txt", "r")  
FileNotFoundError: [Errno 2] No such file or directory: 'file78.txt'  
>>>
```

Try Clauses

- A ***try...except statement*** responds to exceptions and prevents the program from stopping/crashing.
 - The process of intercepting and responding to exceptions is called *exception handling*.
 - A try...except statement has two required parts: A try clause and (at least one) except clause.
- A ***try clause*** contains statements that may or may not raise exceptions.
 - No limit to the number of statements that can be executed.

```
try :
```

```
    my_text_file = open("file78.txt", "r")
```

Except Clauses

- An ***except clause*** contains statements to execute in the event a specific type of exception occurs in its preceding try clause.
 - Specifies the type of exception to be handled.
 - No limit to the number of statements that can be executed within the clause.

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")
```

Handled Exception

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")
```

Unable to locate file

Before:

```
my_text_file = open("file78.txt", "r")
```

```
Traceback (most recent call last):  
  File "C:\testing\examples.py", line 8, in <module>  
    myTextFile = open("file78.txt", "r")  
FileNotFoundError: [Errno 2] No such file or directory: 'file78.txt'  
>>>
```

Else Clauses

- Try...except statements can have one, optional, else clause.
 - The code in the else clause will be executed only when no exceptions were raised in the try clause.

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")
```

Else Clauses

File does not exist

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")
```

Unable to locate file

File does exist

```
try :  
    my_text_file = open("file.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")
```

File opened successfully

Finally Clauses

- Try...except statements can have one, optional, finally clause.
 - The code in the finally clause will be always be executed, regardless of if any exceptions were raised in the try clause.

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")  
finally :  
    print("Finished")
```

Finally Clauses

File does not exist

```
try :  
    my_text_file = open("file78.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")  
finally :  
    print("Finished")
```

Unable to locate file
Finished

File does exist

```
try :  
    my_text_file = open("file.txt", "r")  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("File opened successfully")  
finally :  
    print("Finished")
```

File opened successfully
Finished

Finally Clauses

- Finally clauses are normally used for any cleanup.
 - Closing any files opened, closing any open network/database connections, etc.

```
try :  
    my_text_file = open("file.txt", "r")  
    for line in my_text_file :  
        print(line.rstrip("\n"))  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("Done reading file.")  
finally :  
    my_text_file.close()  
    print("Finished")
```

Nested Try...Except Statements

- Exceptions can occur anywhere in a try...except statement.
 - Even in except, else, and finally clauses.

```
try :  
    my_text_file = open("file78.txt", "r")  
    for line in my_text_file :  
        print(line.rstrip("\n"))  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("Done reading file.")  
finally :  
    my_text_file.close()  
    print("Finished")
```

If this file doesn't exist, it isn't opened because an exception is raised.

This will cause an error because my_text_file was never initialized

Nested Try...Except Statements

File does not exist

```
try :  
    my_text_file = open("file78.txt", "r")  
    for line in my_text_file :  
        print(line.rstrip("\n"))  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("Done reading file")  
finally :  
    my_text_file.close()  
    print("Finished")
```

```
Unable to locate file  
Traceback (most recent call last):  
  File "C:\testing\examples.py", line 17, in <module>  
    myTextFile.close()  
NameError: name 'myTextFile' is not defined  
>>> |
```

File does exist

```
try :  
    my_text_file = open("file.txt", "r")  
    for line in my_text_file :  
        print(line.rstrip("\n"))  
except FileNotFoundError :  
    print("Unable to locate file")  
else :  
    print("Done reading file")  
finally :  
    my_text_file.close()  
    print("Finished")
```

```
Dennis Ritchie  
John vonNeumann  
Grace Hopper  
Done reading file  
Finished
```

Nested Try...Except Statements

File does not exist

```
try :
    my_text_file = open("file78.txt", "r")
    for line in my_text_file :
        print(line.rstrip("\n"))
except FileNotFoundError :
    print("Unable to locate file")
else :
    print("Done reading file")
finally :
    try :
        my_text_file.close()
    except NameError :
        0 + 0 #Do nothing
    else :
        print("Finished")
```

Unable to locate file

Handling Multiple Exceptions

- A try...except statement can contain multiple except clauses.
 - Code in a try clause may raise a number of different types of exceptions.

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    result = 100 / int(line)
    print(result)
    line = input("Enter a number: ")
```

```
Enter a number: 5
20.0
Enter a number: 10
10.0
Enter a number: 3
33.333333333333336
Enter a number: exit
>>>
```

Handling Multiple Exceptions

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    result = 100 / int(line)
    print(result)
    line = input("Enter a number: ")
```

```
Enter a number: 10
10.0
Enter a number: cat
Traceback (most recent call last):
  File "C:\testing\examples.py", line 10, in <module>
    result = 100 / int(line)
ValueError: invalid literal for int() with base 10: 'cat'
>>>
```

Handling Multiple Exceptions

```
try :  
    line = input("Enter a number: ")  
    while line.lower() != "exit" :  
        result = 100 / int(line)  
        print(result)  
        line = input("Enter a number: ")  
except ValueError :  
    print("Invalid input.")
```

```
Enter a number: 10  
10.0  
Enter a number: cat  
Invalid input.  
>>>
```

Handling Multiple Exceptions

```
try :  
    line = input("Enter a number: ")  
    while line.lower() != "exit" :  
        result = 100 / int(line)  
        print(result)  
        line = input("Enter a number: ")  
except ValueError :  
    print("Invalid input.")
```

```
Enter a number: 10  
10.0  
Enter a number: 5  
20.0  
Enter a number: 0  
Traceback (most recent call last):  
  File "C:\testing\examples.py", line 11, in <module>  
    result = 100 / int(line)  
ZeroDivisionError: division by zero  
>>>
```


Handling Multiple Exceptions

```
try :  
    line = input("Enter a number: ")  
    while line.lower() != "exit" :  
        result = 100 / int(line)  
        print(result)  
        line = input("Enter a number: ")  
except ValueError :  
    print("Invalid input.")  
except ZeroDivisionError :  
    print("Cannot divide by zero.")
```

```
Enter a number: 10  
10.0  
Enter a number: 5  
20.0  
Enter a number: 0  
Cannot divide by zero.  
>>>
```

Handling Multiple Exceptions

- Moving/rewriting the try...except statement into the while loop would allow the program to continue, uninterrupted.

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    try :
        result = 100 / int(line)
    except ValueError :
        print("Invalid input.")
    except ZeroDivisionError :
        print("Cannot divide by zero.")
    else :
        print(result)
    finally :
        line = input("Enter a number: ")
```

```
Enter a number: 10
10.0
Enter a number: 5
20.0
Enter a number: cat
Invalid input.
Enter a number: 6
16.666666666666668
Enter a number: 0
Cannot divide by zero.
Enter a number: 17
5.882352941176471
Enter a number: exit
>>>
```

Handling Multiple Exceptions

- A single except clause can handle multiple exception types.

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    try :
        result = 100 / int(line)
    except (ValueError, ZeroDivisionError) :
        print("Input error.")
    else :
        print(result)
    finally :
        line = input("Enter a number: ")
```

```
Enter a number: 10
10.0
Enter a number: 5
20.0
Enter a number: cat
Input error.
Enter a number: 6
16.666666666666668
Enter a number: 0
Input error.
Enter a number: 17
5.882352941176471
Enter a number: exit
>>>
```

Handling Multiple Exceptions

- Except clauses can be written without specifying an exception type.
 - This will catch *any* exceptions that occur in the try clause.
 - Be careful if you do this! You will have no way of knowing why an exception occurred.

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    try :
        result = 100 / int(line)
    except :
        print("Input error.")
    else :
        print(result)
    finally :
        line = input("Enter a number: ")
```

```
Enter a number: 10
10.0
Enter a number: 5
20.0
Enter a number: cat
Input error.
Enter a number: 6
16.666666666666668
Enter a number: 0
Input error.
Enter a number: 17
5.882352941176471
Enter a number: exit
>>>
```

Handling Multiple Exceptions

- A better solution is to specify the Exception object type in the except clause.
 - This will still catch any exceptions in the try clause, but also gives you a way to determine the exception's cause.

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    try :
        result = 100 / int(line)
    except Exception as error:
        print("Error. Reason: ", error)
    else :
        print(str(result))
    finally :
        line = input("Enter a number: ")
```

Handling Multiple Exceptions

```
line = input("Enter a number: ")
while line.lower() != "exit" :
    try :
        result = 100 / int(line)
    except Exception as error:
        print("Error. Reason: ", error)
    else :
        print(str(result))
    finally :
        line = input("Enter a number: ")
```

```
Enter a number: 5
20.0
Enter a number: cat
Error. Reason:  invalid literal for int() with base 10: 'cat'
Enter a number: 0
Error. Reason:  division by zero
Enter a number: 6
16.666666666666668
Enter a number: exit
```

Raising Exceptions

- It is possible to raise your own exceptions.
- The raise keyword allows the program to create an exception.
 - This is no different than how any other exception is raised.

```
def divide100(value_in) :  
    if value_in < 0 :  
        raise Exception  
    else:  
        return 100 / value_in
```

Raising Exceptions

```
def divide100(value_in) :  
    if value_in < 0 :  
        raise Exception  
    else:  
        return 100 / value_in  
  
line = input("Enter a number: ")  
while line.lower() != "exit" :  
    try :  
        result = divide100(int(line))  
    except ValueError :  
        print("Invalid input.")  
    except ZeroDivisionError :  
        print("Cannot divide by zero.")  
    except :  
        print("An exception occurred.")  
    else :  
        print(result)  
    finally :  
        line = input("Enter a number: ")
```

```
Enter a number: 6  
16.666666666666668  
Enter a number: cat  
Invalid input.  
Enter a number: 0  
Cannot divide by zero.  
Enter a number: -6  
An exception occurred.  
Enter a number: 5  
20.0  
Enter a number: exit  
>>>
```


Raising Exceptions

- When raising an exception yourself, you can supply a string of information to it.

```
def divide100(value_in) :  
    if value_in < 0 :  
        raise Exception("Negative number")  
    else:  
        return 100 / value_in
```

Raising Exceptions

```
def divide100(value_in) :  
    if value_in < 0 :  
        raise Exception("Negative number")  
    else:  
        return 100 / value_in  
  
line = input("Enter a number: ")  
while line.lower() != "exit" :  
    try :  
        result = divide100(int(line))  
    except ValueError :  
        print("Invalid input.")  
    except ZeroDivisionError :  
        print("Cannot divide by zero.")  
    except Exception as error:  
        print("Error. Reason: ", error)  
    else :  
        print(str(result))  
    finally :  
        line = input("Enter a number: ")
```

```
Enter a number: 6  
16.666666666666668  
Enter a number: cat  
Invalid input.  
Enter a number: 0  
Cannot divide by zero.  
Enter a number: -6  
Error. Reason: Negative number  
Enter a number: 5  
20.0  
Enter a number: exit  
>>>
```