

ICS-104

# Chapter 7

# Notes

By: Naif Alqahtani

 Twitter: @NaifAlqahtani

 Youtube.com/MWNLLT

## • Files:

- It is sometimes useful to read information from files and to write information to files to save them.
- The `open()` function is used to open files.
- The first argument that goes into `open()` is the file name.
- Second argument is the mode. It can be "r", "w", "a" and more.
- You assign a name to the file like this → `name = open(" ", " ")`

---

### Read mode:

- To open a file in read mode, use "r" as the second argument for `open()`.
- The file name you want to open must exist or an error will rise.
- You cannot modify the contents of the file in read mode at all.

---

### Write mode:

- To open a file in write mode, use "w" as the second argument for `open()`.
- If the file does not exist, `open()` will create the file with the given name.
- If the file exists, `open()` will clear and delete everything inside before opening it.

- 
- Do not forget to use `.close()` to close the file after you finish using it.
  - Examples:

→ `infile = open("notes.txt", "r")`

↑                      ↑                      ↑                      ↑  
file object    open    file name to    mode of  
name to be   function   be opened   opening  
used in code.

→ `infile.close()`

↑                      ↑  
using same    method  
name we    to close  
assigned file   file

# • File Operations:

## Read mode:

- To read 1 line at a time, use `.readline()` method.
- `.readline()` reads a file until it sees the invisible newline "`\n`" char
- The number of times `.readline()` is called, determines which line to read.
- `.readline()` returns the text in file at that line as a string.
- **IMPORTANT:** `.readline()` will also include the invisible newline "`\n`" char!
- To read Multiple lines at once, use `.readlines()` method.
- `.readlines()` returns a List of all lines in the file, where each line is an element in the list. So you can index the list to access the line you want.
- You only need to use `.readlines()` once.
- You can iterate the returned list to print each line individually.
- Do not forget, the lines read contain the invisible newline "`\n`" char!
- You can remove them by using the reverse strip method: `.rstrip()`
- Alternatively, you can use `.readline()` inside a loop and append each line to a list one at a time.
- Keep in mind, both `.readlines()` and `.readline()` return lines of the type: `string`, only.
- You can use the `.split()` method on a line that you read to split the line string into multiple strings of the words in that line
- This is helpful if you want to read 1 word in the file at a time.
- If you want to read 1 character at a time, use the `.read()` method.
- `.read()` takes the number of characters to read everytime it is called.
- you can use the same operations on `.readline()` to read multiple chars.

Note: This is not included in the slides but it is extremely helpful.

## • File Operations:

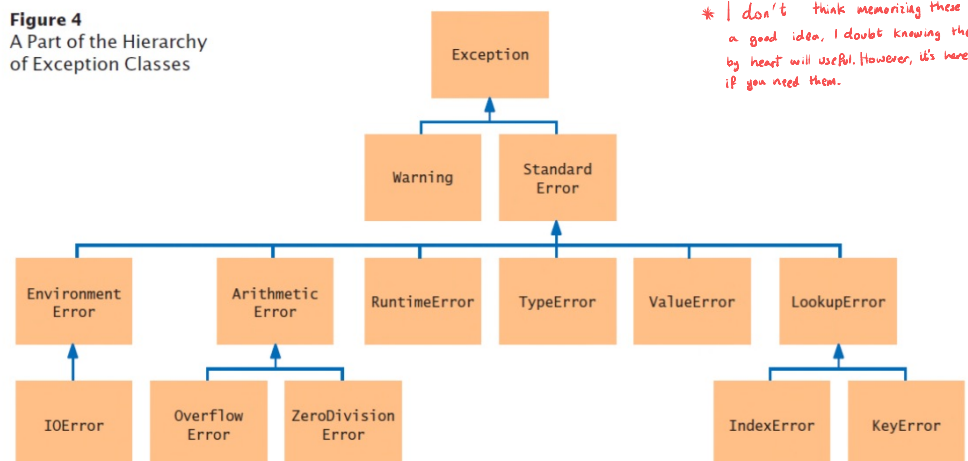
### Write mode:

- To write 1 line to a file, use `.write()` method
- To write multiple lines, you can iterate a list and write each line to the file.
- Do not forget to add `"\n"` at the end of your line so that each string has its own line inside the file.

## • Exception Handling:

- Basically: What to do if an error might occur.
- Types of errors:

**Figure 4**  
A Part of the Hierarchy  
of Exception Classes



\* I don't think memorizing these is a good idea, I doubt knowing them by heart will be useful. However, it's here if you need them.

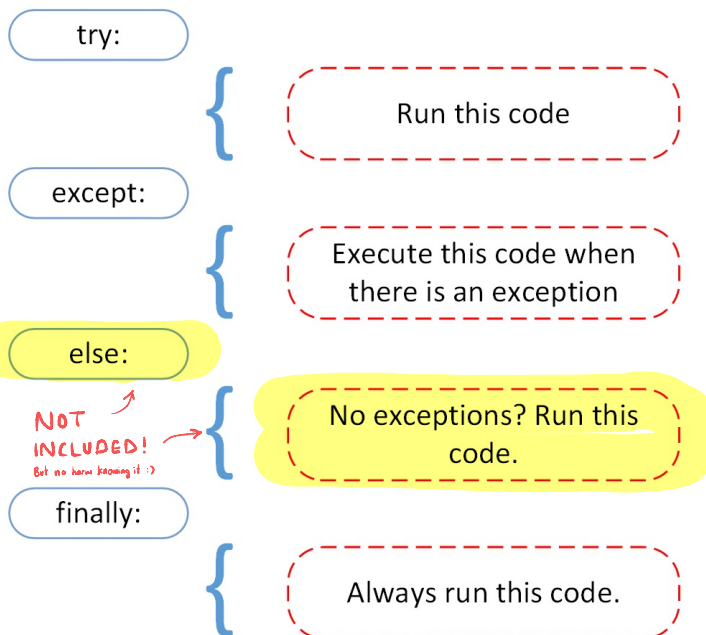
## • Raising exceptions:

- To raise an exception, use one of the exceptions in Figure 4, and pass the error message as an argument.
- Eg: `raise ValueError("Not correct!")`
- When an exception is raised, the program terminates.
- Exception blocks prevent the program from terminating by providing alternative code to run.
- If some part of your code might cause an error in some cases, you need to handle the exception.

## • Handling Exceptions:

- To handle exceptions, use a `try... except` block.
- You move parts of your code where an exception (error) might occur inside the `try` block.
- Then, you provide the code you want to run when an exception (error) is raised, inside the `except` block.
- Another clause that is sometimes used after the `try... except` clause, is the `finally` clause.
- Code inside `finally` will always run regardless if an exception is raised or not.

# • Exception Handling Syntax:



**Syntax**

```
try :
    statement
    statement
    . . .
except ExceptionType :
    statement
    statement
    . . .
except ExceptionType as varName :
    statement
    statement
    . . .
```

This function can raise an IOError exception.

```
try :
    infile = open("input.txt", "r")

    line = inFile.readline()
    process(line)
```

When an IOError is raised, execution resumes here.

```
except IOError :
    print("Could not open input file.")
```

Additional except clauses can appear here. Place more specific exceptions before more general ones.

```
except Exception as exceptObj :
    print("Error:", str(exceptObj))
```

This is the exception object that was raised.

**Syntax**

```
try :
    statement
    statement
    . . .
finally :
    statement
    statement
    . . .
```

This code may raise exceptions.

```
outfile = open(filename, "w")
```

```
try :
    writeData(outfile)
    . . .
finally :
    outfile.close()
    . . .
```

This code is always executed, even if an exception is raised in the try block.

The file must be opened outside the try block in case it fails. Otherwise, the finally clause would try to close an unopened file.