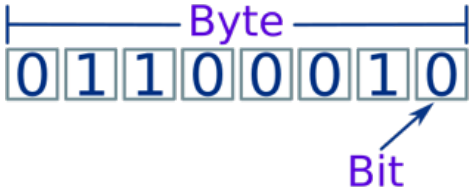
Topic: Files Reading/Writing and Exceptions

Goals: By the end of this topic, we will…

- *discuss storing and retrieving information in file*
- *exceptions*

Acknowledgements: These class notes build on the content of my previous courses as well as the work of R. Jordan Crouser, and Jeffrey S. Castrucci.

Working with files



- Containers of bits, organized into bytes
- Could represent text, images, music, movies, programs, applications, list of files (folders)… but underneath, they're all the same: 0s and 1s
- We'll start by playing with text files

Reading a text file

- In order to bring data stored in a text file into a Python program, we need to read it:

```
def main():
    # Open file for reading
    file = open("test.txt", "r")        #"r" for read

    # Read the file and print its contents
    text = file.read()
    print(text)

    # Close the files
    file.close()

main()
```

- "r" stands for READ-MODE
- before you can read something from a file you need to `open` it first
- once opened `file` is an object
  - the `.read()` method that returns a string
  - the `.readlines()` method that returns a list of strings
  - the `.readline()` method that returns a single line of the file as a string
- after you `.read()`, `.readline()` or `.readlines()` a file you need to `.close()` it

Key points for reading files

- Three-step process:
  - `.open()`
  - `.read()` or `.readlines()`
  - `.close()`
- All three steps, always in that order
- If you want to .read() a file multiple times, you have to repeat the whole process

```
file = open("test.txt", "r")
text1 = file.read()
text1 = file.read()        # EMPTY!
file.close()
```

Exercise: Reading and Writing Files

Write a program that:

- reads the file `horizontal.txt` (from course website)
- print it to the console

Writing data to a text file

- The process looks very similar when we want to write data to a file:

```
def main():
    # Open file for reading
    file = open("test2.txt", "w")        #"w" for write

    # Write a string to the file
    text = "Output this string into a file..."
    file.write(text)

    # Close the files
    file.close()

main()
```

- exactly the same function to `.open()` a file
- "w" stands for WRITE-MODE
- before you can write something from a file you need to `open` it first
  - if the file does not exist, Python will create it (e.g. test2.txt)
  - if the file does exist, Python will overwrite it
- once opened `file` is an object
  - the `.write()` method that takes in a string
- after you `.write()` to a file you need to `.close()` it

Note:

- if you want to add to an existing file instead of overwriting it,
  - "a" stands for APPEND-MODE

Key points for writing files

- Three-step process:
  - `.open()`
  - `.write()`
  - `.close()`
- Unlike `.read()`, you can `.write()` to an `.open()` file as many times as you want (appending each time)
- If you want a new line, you have to add it yourself! (\n)

```
file = open("test2.txt", "w")
file.write("Hello")
file.write("there!")
file.close()
```

Exercise: Reading and Writing Files

Write a program that:

- reads the file `horizontal.txt` (from course website)
- breaks it into individual words
- and writes the words to a new file `vertical.txt`, each one on its own line

Handling Exceptions

```
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

Recall:

- This is an Exception
- The kind of error gives you a club about what the problem is...
- It also tells you where the problem is.

The drawbacks to using exceptions is that the program stops and crashes.

Example: What happens if the user enters a negative number?

```
import math
def main():
     x = int(input("Enter a number greater than 0: "))
     print("The log is:", math.log(x))

if __name__ == "__main__":
    main()
```

Possible work around:

```
import math
def main():
     x = int(input("Enter a number greater than 0: "))
     if x > 0:
          print("The log is:", math.log(x))
     else:
          print(x, "is out of range, sorry. Try again.")
```

but...what happens if the user enters a string?

The try...except block

- There are some cases where avoiding an Exception isn't possible
- In this case, we want tell Python:
  - what we want to happen
  - how to handle it if things go wrong

For example:

```
import math
def main():

    try:
        x = int(input("Enter a number greater than 0: "))
        print("The log is:", math.log(x))

    except ValueError:
        print("Not a valid input.")
```

- Even if you can't avoid all errors, you can design your program to fail gracefully
- You can handle multiple different kinds of Exceptions, and you can handle them differently
- Think about edge cases to provide specific feedback about what went wrong