

Data Structures and Algorithms

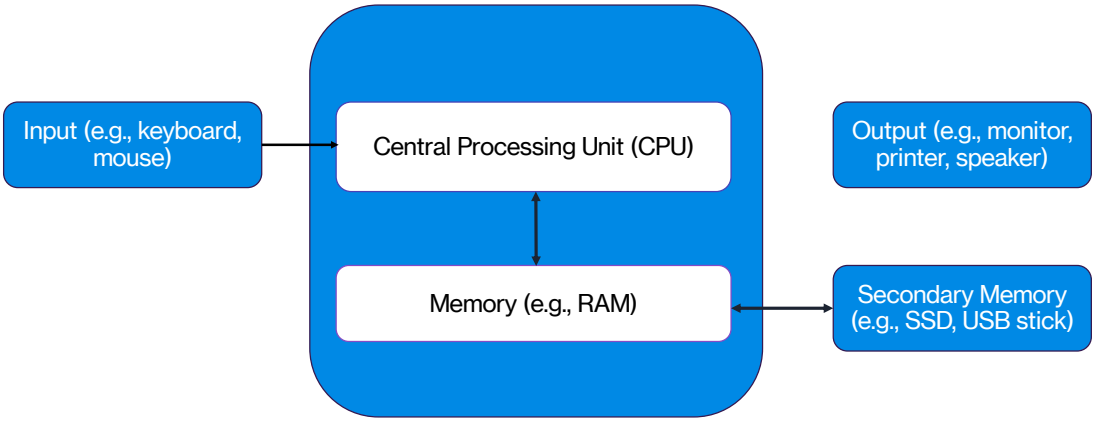
Files and Errors

CS51 – Fall 2025

Today we will talk about how to work with files, specifically text files, and how to handle common errors in Python.

Files

- Programs are stored in main memory, i.e. RAM.
- Files are data stored in secondary memory, such as the hard drive, USB sticks. etc.



2

As we saw during the Systems unit in class, programs are stored in (main) memory, that is the RAM. That means all the variables they need to do their work are quickly accessible because RAM is fast. But once the program exits or RAM is not powered, we lose access to the program data. In contrast, if we want information to be permanently stored we would save it in files. Files are chunks of data in secondary memory, for example a solid state drive or a USB stick.

Files

- Files store data in the form of bytes.
- Although we make a distinction between text and binary files, text files are binary files that follow a specific encoding, most likely UTF-8, to interpret the bytes as valid characters (letters, numbers, symbols, emojis).

3

Files store data in the form of bytes. Although we make a distinction between text and binary files, text files are binary files that follow a specific encoding, most likely UTF-8, to interpret the bytes as valid characters (letters, numbers, symbols, emojis).

File system

- A **filesystem** is the way that an operating system organizes and provides access to store files.
- Files have a **filename** that we can identify them by.
- Files also reveal information about their type or appropriate applications to read them through their **extension**. E.g., .pdf shows that this is a pdf file and can be read with pdf readers such as Adobe Reader.
- They also contain **metadata** about their size, when they were created, last accessed, who created them or last modified them, who can read them/change them etc.
- Files are organized in **folders** or **directories** which themselves can have nested folders.
- We specify where a file resides through its **path**. For example, in my laptop, this presentation is in the directory /Users/apaa2017/Documents/cs051/lectures

4

The operating system provides a service called the filesystem that dictates how files are stored, organized, and accessed. A file is identified by its name. Its extension reveals its type and appropriate application to open it so that it is not read as raw binary data. Files also contain information like their size etc in metadata. They are organized in directories and their position is determined through a path.

Reading text files line by line

- To read a file line by line, we use the following template:

```
reader = open('filename.txt', 'r')
for line in reader:
      # do something
reader.close()
```

- `reader` is a reference to a file object.
- `filename.txt` gets replaced with the actual name of the file we want to read.
- The parameter `r` stands for read. That means we want to open the file to read it.
- The for in loop allows us to go line by line in the file. We could have done for example `print(line)`
- Don't forget to close the once you open it.

5

We will only focus on text files here. Python provides a very concise way of opening a file to read it. Once we open a file, we can read it line by line with a simple for loop. Don't forget to close a file reader once you open it.

Reading files as a list of strings

- Python also supports reading all lines of a file into a list of strings.
- Each string in the list represents a line from the file, and it includes the newline character ('\n') at the end of each line.
- ```
  reader = open('filename.txt', 'r')
  list_of_lines = reader.readlines()
  ```

6

Alternatively, we can also read all lines into a list of strings, with each string representing a file line.
  ```

(Over) Writing to files

- To write in a file, we use the following template:

```
writer = open('filename.txt', 'w')
writer.write('First line')
writer.write('Second line')
writer.close()
```

- `writer` is a reference to a file object.
- `filename.txt` gets replaced with the actual name of the file we want to write in. If it already exists, it overwrites it. If the file does not exist, creates a new file for writing.
- The parameter `w` stands for write. That means we want to open the file to write in it.
- We use the `write` method to write in the file object.
- Don't forget to close a file once you open it.

7

To write to a file, we open it and pass the 'w' parameter. Be mindful that if there is already a file with this name, its contents will be replaced that is overwritten this way.

(Appending when) Writing to files

- To write at the end of a file, we use the following template:

```
writer = open('filename.txt', 'a')
writer.write('Append to the end of the file')
writer.close()
```

- `writer` is a reference to a file object.
- `filename.txt` gets replaced with the actual name of the file we want to write in. If it already exists, the file pointer is at the end of the file. If the file does not exist, creates a new file for writing.
- The parameter *a* stands for append. That means we want to open the file to write at the end of it so that we don't lose existing content.
- We use the `write` method to write in the file object.
- Don't forget to close a file once you open it.

8

If you want to maintain the original file and append to it, you should use the 'a' option instead.

Practice Time

- Define a function `count_capital_letters` that takes on parameter `filename` and returns the number of capital letters in that file.
- You may use the `isupper` method to test if a character is uppercase.

9

Your time to write a function that counts the number of capital letters in a provided file.

Answer

- Define a function `count_capital_letters` that takes on parameter `filename` and returns the number of capital letters in that file.

```
def count_capital_letters(filename):
    count = 0
    opener = open(filename, 'r')
    for line in opener:
         for char in line:
             if char.isupper():
                 count += 1
    opener.close()
    return count
```

10

Practice Time

- Define a function `unique_words` that takes on parameter `filename` and returns a list with all the words in the file. If a word exists multiple times, it should only include it once in the list. You can split words by whitespace and ignore capitalization (i.e. use the method `lower`)

Answer

```
def unique_words(filename):
    words = []
    reader = open(filename, 'r')
    for line in reader:
       for word in line.split():
           clean_word = word.strip().lower()
           if clean_word not in words:
               words.append(clean_word)
    reader.close()
    return words
```

12

Errors

- You already have experience with things going wrong when programming. Maybe:
  - you forgot to pass an argument
  - tried to print an int
  - tried to access a list item out of the valid range of indices
- When working with files, you may also face trouble:
  - try to read a file in the wrong directory
  - try to write in a file without having permission to do so

Exceptions

- Exceptions are errors that we know how to handle.
- The syntax is

```
try:
    # some potentially dangerous code
except:
    # handle the error
finally:
    # code you want to execute no matter what
```

14

When there are errors like this that we know how to handle or at least we want to warn the user with a bit more information, you can use exceptions. To handle exceptions, we use the try except statement. In the try block, we put the code for which we know that things may go poorly. In the except block we handle the error. Optionally, we can also specify a finally block where we have code that should be executed no matter what.

Exceptions

- For example, when working with files

```
reader = open('filename.txt', 'r')
try:
  for line in reader:
       # do something
except:
    print('There was an error with the file when reading it line by line')
finally:
   reader.close()
```

15

For example, we may know that something might go wrong as we read a file line by line.