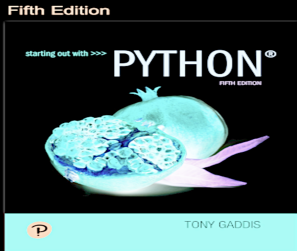# *COSC 1306 - Prog for Non-Majors*
Files and Exceptions

Dr. Mohan

McMurry University

October 31, 2023

- Introduction to File Input and Output.
- Using Loops to Process Files.
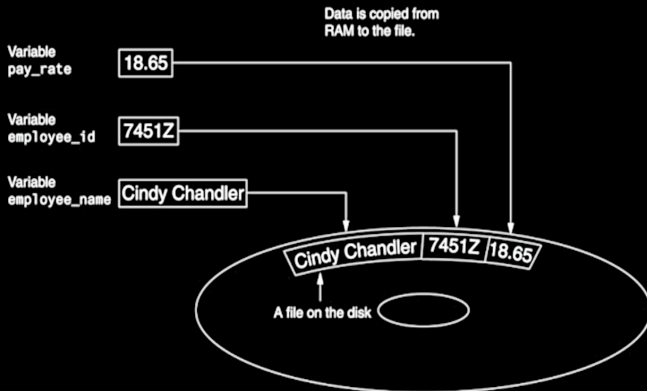- Processing Records.
- Exceptions.

Fifth Edition

starting out with >>> **PYTHON**®
FIFTH EDITION

TONY GADDIS

Chapter 6
Files and Exceptions

- For program to retain data between the times it is run, you must save the data.
    - Data is saved to a file, typically on computer disk.
    - Saved data can be retrieved and used at a later time,
- **Writing data to:** saving data on a file.
- **Output file:** a file that data is written to.

Figure 6-1 Writing data to a file
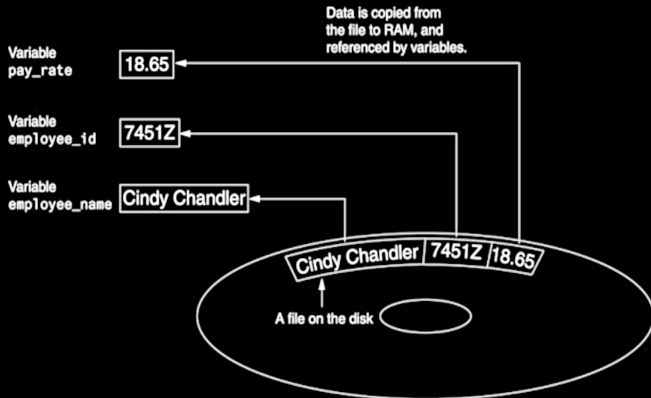
- **Reading data from:** process of retrieving data from a file.
  - **Input file:** a file from which data is read.
  - **Three steps** when a program uses a file.
    - Open the file.
    - Process the file.
    - Close the file.

Figure 6-2 Reading data from a file

Types of Files and File Access Methods

- In general, two types of files:
  - **Text file:** contains data that has been encoded as text.
  - **Binary file:** contains data that has not been converted to text.
- Two ways to access data stored in file:
  - **Sequential access:** file read sequentially from beginning to end, can't skip ahead.
  - **Direct access:** can jump directly to any piece of data in the file.

- **Filename extensions:** short sequences of characters that appear at the end of a filename preceded by a period.
  - Extension indicates type of data stored in the file.
- File object: object associated with a specific file.
  - Provides a way for a program to work with the file: file object referenced by a variable.

Figure 6-4 A variable name references a file object that is associated with a file
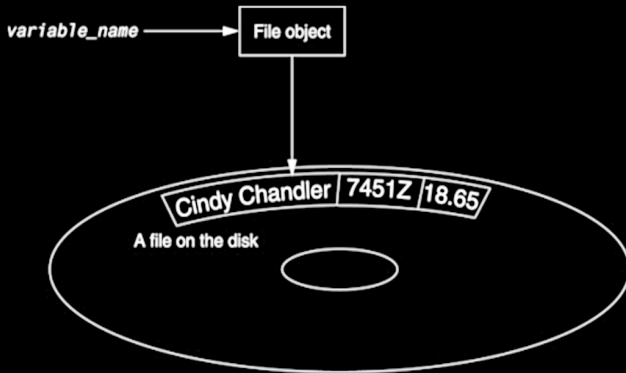
Opening a File

- **open function:** used to open a file.
  - Creates a file object and associates it with a file on the disk.
  - General format:
    file_object = open(filename, mode)
- **Mode:** string specifying how the file will be opened.
  - Example: reading only ('r'), writing ('w'), and appending ('a').

Specifying the Location of a File

- If **open function** receives a filename that does not contain a path, assumes that file is in same directory as program.
- If program is running and file is created, it is created in the same directory as the program.
  - Can specify alternative path and file name in the open function argument.
    - Prefix the path string literal with the letter r.

Writing Data to a File

- **Method:** a function that belongs to an object.
  - Performs operations using that object.
- File object's write method used to write data to the file.

  Format: file_variable.write(string)

- File should be closed using file object close method.

  Format: file_variable.close()

Reading Data From a File

- **read method:** file object method that reads entire file contents into memory.
  - Only works if file has been opened for reading.
  - Contents returned as a string.
- **readline method:** file object method that reads a line from the file.
  - Line returned as a string, including '\n'.
- **Read position:** marks the location of the next item to be read from a file.

Concatenating a Newline to and Stripping it From a String

- In most cases, data items written to a file are values referenced by variables.
  - Usually necessary to concatenate a '\n' to data before writing it.
    - Carried out using the + operator in the argument of the write method.
- In many cases need to remove '\n' from string after it is read from a file.
  - **rstrip method:** string method that strips specific characters from end of the string.

Appending Data to an Existing File

- When open file with 'w' mode, if the file already exists it is overwritten.
- To append data to a file use the 'a' mode.
  - If file exists, it is not erased, and if it does not exist it is created.
  - Data is written to the file at the end of the current contents.

Writing and Reading Numeric Data

- Numbers must be converted to strings before they are written to a file.
- **str function:** converts value to string.
- Number are read from a text file as strings.
  - Must be converted to numeric type in order to perform mathematical operations.
 - Use int and float functions to convert string to numeric value.

- **Files typically** used to hold large amounts of data.
  - Loop typically involved in reading from and writing to a file.
- Often **the number of items** stored in file is unknown.
  - The readline method uses an empty string as a sentinel when end of file is reached.
 - Can write a while loop with the condition.
    - **while line != ''**

Figure 6-17 General logic for detecting the end of a file
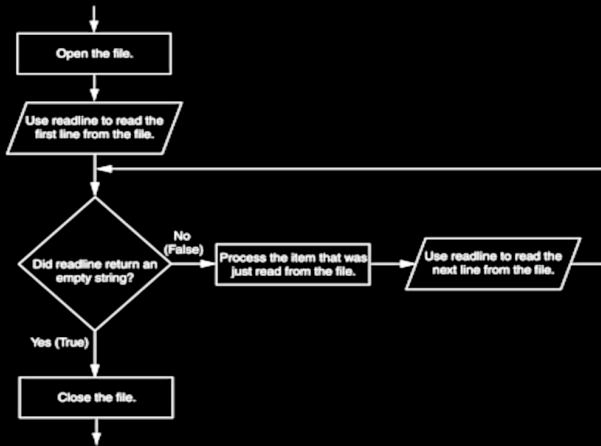
Using Python's for Loop to Read Lines

- **Python** allows the programmer to write a for loop that automatically reads lines in a file and stops when end of file is reached.
- Format:

  colorgreenfor line in file_object:

  colorgreenstatements

- The **loop iterates** once over each line in the file.
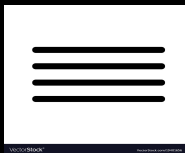
- **Record:** set of data that describes one item.
- **Field:** single piece of data within a record.
- Write record to sequential access file by writing the fields one after the other.
- Read record from sequential access file by reading each field until record complete.

- When working with **records**, it is also important to be able to:
  - Add records.
  - Display records.
  - Search for a specific record.
  - Modify records.
  - Delete records.

- **Exception:** error that occurs while a program is running.
  - Usually causes program to abruptly halt.
- **Traceback:** error message that gives information regarding line numbers that caused the exception.
  - Indicates the type of exception and brief description of the error that caused exception to be raised.

- Many **exceptions** can be prevented by careful coding.
  - **Example:** input validation.
  - Usually involve a simple decision construct.
- Some exceptions cannot be avoided by careful coding.
  - **Examples:**
    - Trying to convert non-numeric string to an integer.
    - Trying to open for reading a file that doesn't exist.

- **Exception handler:** code that responds when exceptions are raised and prevents program from crashing.
  - In Python, written as try/except statement.
    - General format:
      ```
      try:
          statements
      except exceptionName:
          statements
      ```

    - **Try suite:** statements that can potentially raise an exception.
    - **Handler:** statements contained in except block.

- **If statement** in try suite raises exception:
  - Exception specified in **except clause**:
    - Handler immediately following except clause executes.
    - Continue program after try/except statement.
  - Other exceptions:
    - Program halts with traceback error message.
- If **no exception** is raised, handlers are skipped.

Handling Multiple Exceptions

- Often code in try suite can throw more than one type of exception.
    - Need to write except clause for each type of exception that needs to be handled.
- An **except clause** that does not list a specific exception will handle any exception that is raised in the try suite.
    - Should always be last in a series of except clauses

Displaying an Exception's Default Error Message

- **Exception object:** object created in memory when an exception is thrown.
  - Usually contains default error message pertaining to the exception.
  - Can assign the exception object to a variable in an except clause.
    - Example:
      except ValueErroraserr:
  - Can pass exception object variable to print function to display the default error message.
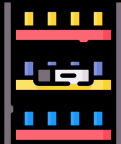
The else Clause

- **try/except** statement may include an optional else clause, which appears after all the except clauses.
  - Aligned with try and except clauses.
  - Syntax similar to else clause in decision structure.
  - **Else suite:** block of statements executed after statements in try suite, only if no exceptions were raised.
    - If exception was raised, the else suite is skipped.

The finally Clause

- **try/except statement** may include an optional finally clause, which appears after all the except clauses.
  - Aligned with try and except clauses.
  - General format:

        finally:
            statements

  - **Finally suite:** block of statements after the finally clause.
    - Execute whether an exception occurs or not.
    - Purpose is to perform cleanup before exiting.

What If an Exception Is Not Handled?

- **Two ways** for exception to go unhandled:
    - No except clause specifying exception of the right type.
    - Exception raised outside a try suite.
- In both cases, **exception** will cause the program to halt.
    - Python documentation provides information about exceptions that can be raised by different functions.

Lesson Summary

This chapter covered:

- Types of files and file access methods.
- Filenames and file objects.
- Writing data to a file.
- Reading data from a file and determining when the end of the file is reached.
- Processing records.
- Exceptions, including:
  - Traceback messages.
  - Handling exceptions,

- Read Textbook Chapter-6.
- Practice the exercise problems at the end of Chapter-6.

Please ask your Questions to clarify!