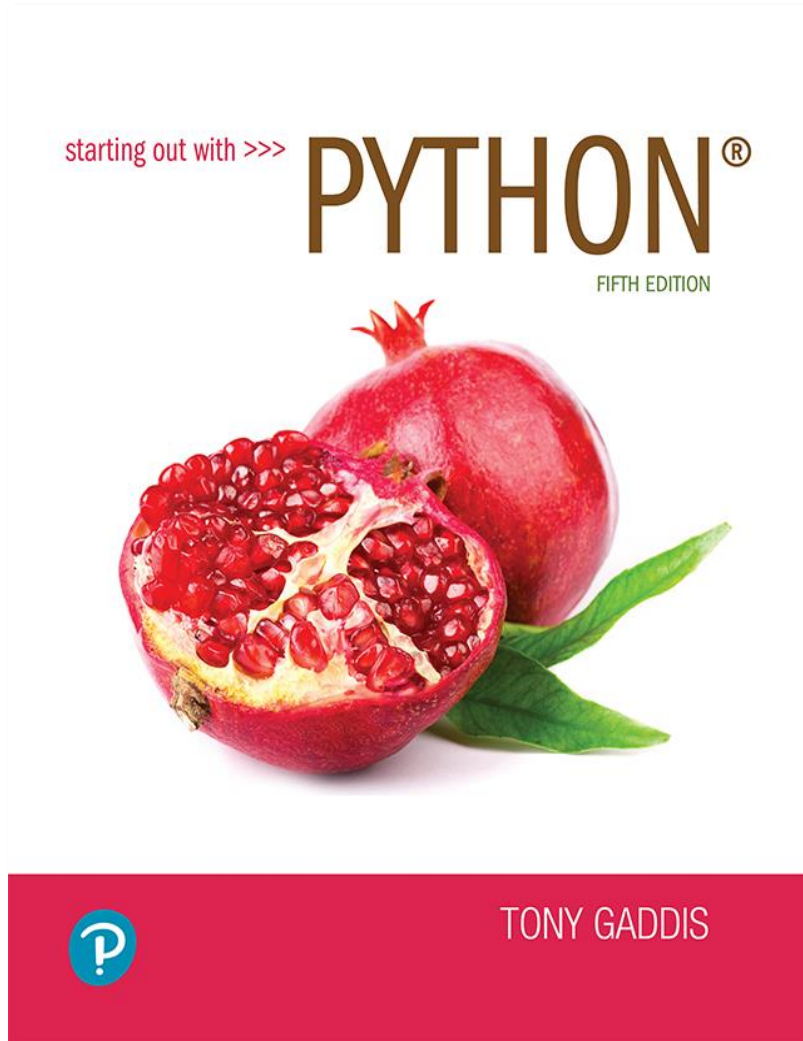# Starting out with Python

Fifth Edition

Chapter 6

Files and Exceptions

Topics

- Introduction to File Input and Output

- Using Loops to Process Files

- Processing Records

- Exceptions

Pearson

Introduction to File Input and Output

- "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

Types of Files and File Access Methods

- In general, two types of files
  - <u>Text file</u>: contains data that has been encoded as text
  - <u>Binary file</u>: contains data that has not been converted to text

- Two ways to access data stored in file
  - <u>Sequential access</u>: file read sequentially from beginning to end, can't skip ahead
  - <u>Direct access</u>: can jump directly to any piece of data in the file

Filenames and File Objects

- <u>Filename extensions</u>: 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

- <u>File object</u>: object associated with a specific file
  - Provides a way for a program to work with the file: file object referenced by a variable
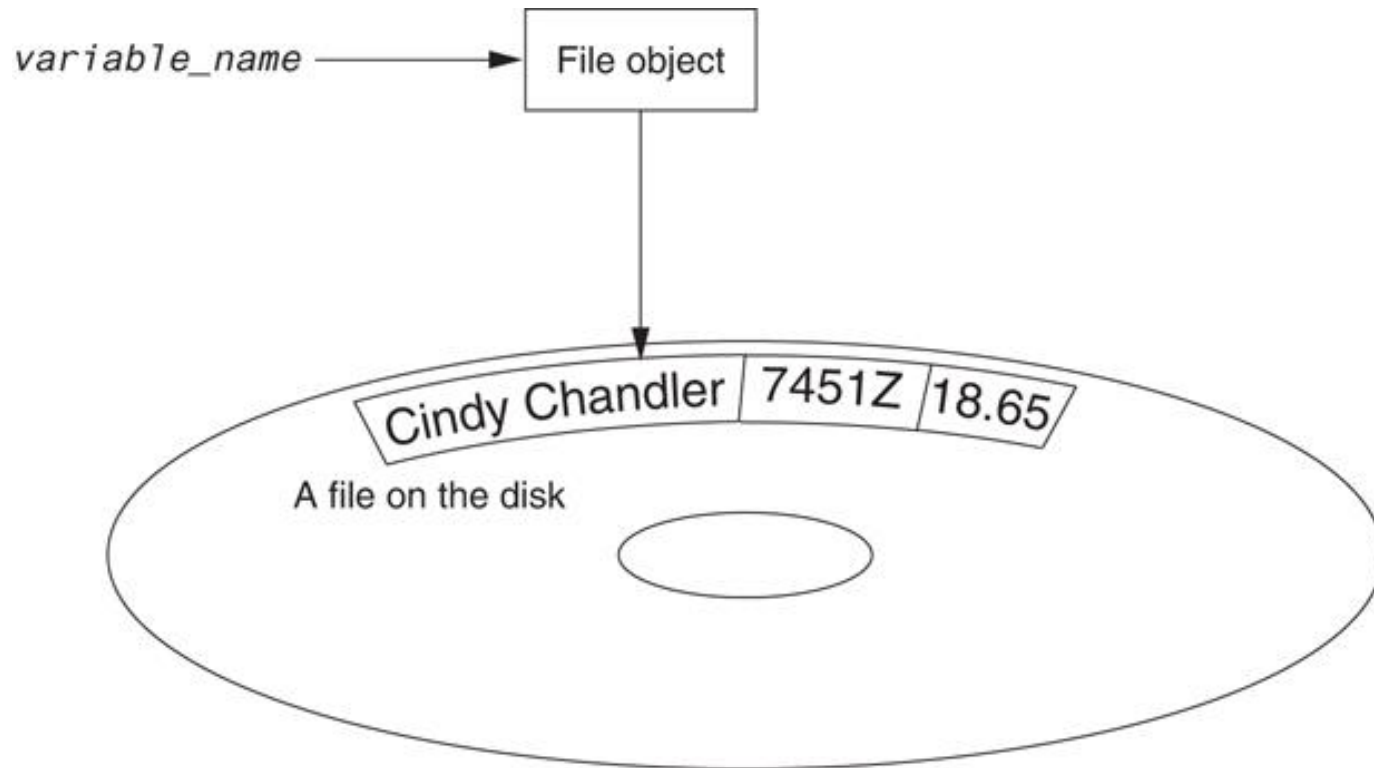
Figure 6-4 A variable name references a file object that is associated with a file

Opening a File

- <u>`open` function</u>: 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`*`)`

- <u>Mode</u>: 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

- <u>Method</u>: 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

6 - 12

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

Using Loops to Process Files

- 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 != ''
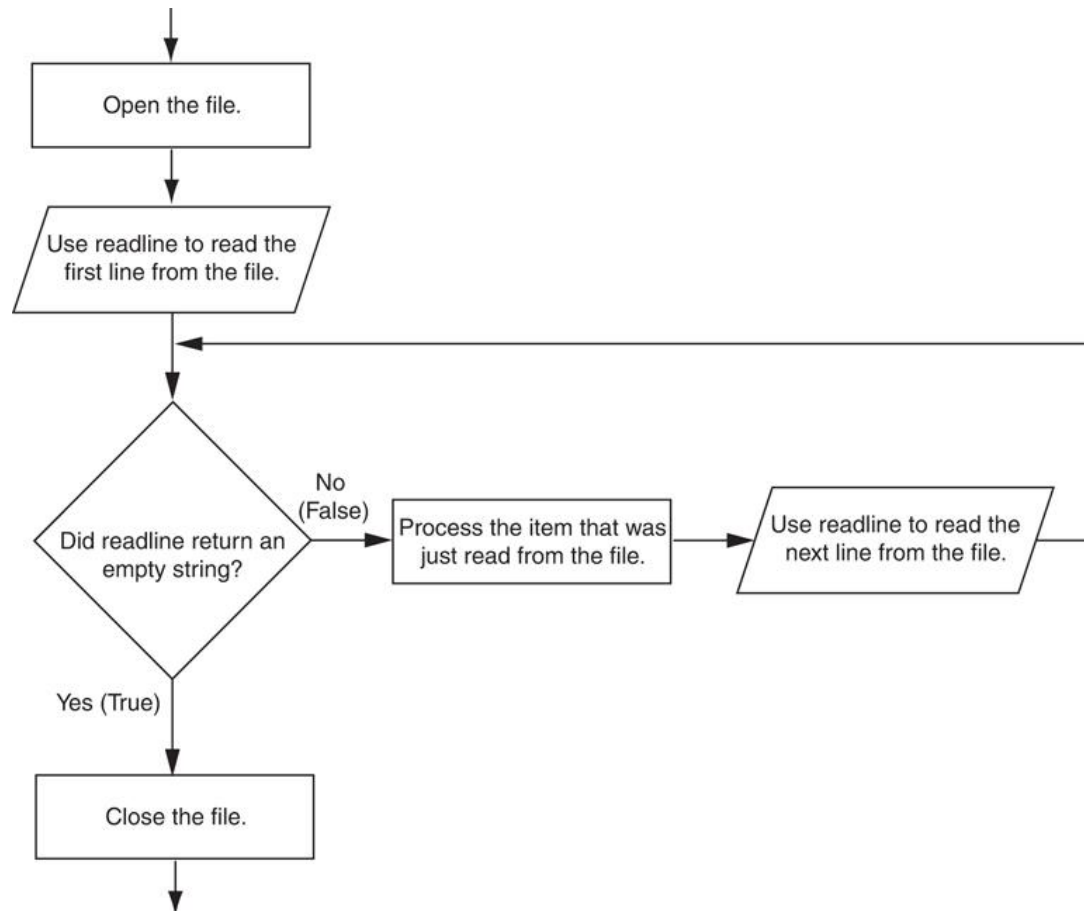      ```

# Using Loops to Process Files



**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: `for line in file_object:`
  - `statements`
  - The loop iterates once over each line in the file

# Processing Records

- <u>Record</u>: set of data that describes one item

- <u>Field</u>: 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

# Processing Records

- 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

6 - 18

# Exceptions (1 of 4)

- <u>Exception</u>: error that occurs while a program is running
  - Usually causes program to abruptly halt

- <u>Traceback</u>: 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

# Exceptions

- 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

# Exceptions (3 of 4)

- <u>Exception handler</u>: 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*

    - <u>Try suite</u>: statements that can potentially raise an exception
    - <u>Handler</u>: statements contained in `except` block

# Exceptions (4 of 4)

- 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

# 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