UNIT 4

Introduction To File Input And Output

When a program needs to save data for later use, it writes the data in a file. The data can be read from the file at a later time.

The programs you have written so far require the user to re-enter data each time the program runs, because data that is stored in RAM (referenced by variables) disappears once the program stops running. If a program is to retain data between the times it runs, it must have a way of saving it. Data is saved in a file, which is usually stored on a computer's disk.

Once the data is saved in a file, it will remain there after the program stops running. Data that is stored in a file can be retrieved and used at a later time.

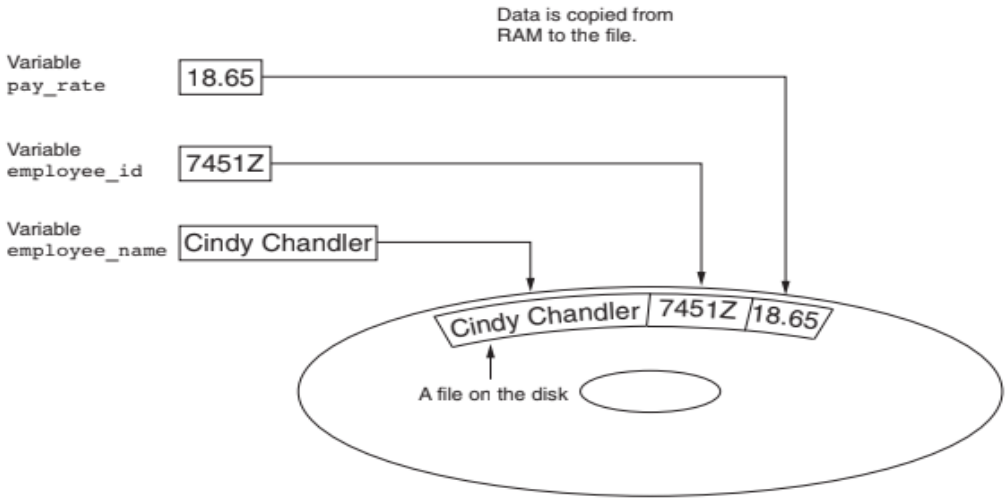
The following are a few examples.

• **Word processors**. Word processing programs are used to write letters, memos, reports, and other documents. The documents are then saved in files so they can be edited and printed.

• **Image editors**. Image editing programs are used to draw graphics and edit images such as the ones that you take with a digital camera. The images that you create or edit with an image editor are saved in files.

• **Spreadsheets.** Spreadsheet programs are used to work with numerical data. Numbers and mathematical formulas can be inserted into the rows and columns of the spread-sheet. The spreadsheet can then be saved in a file for use later.

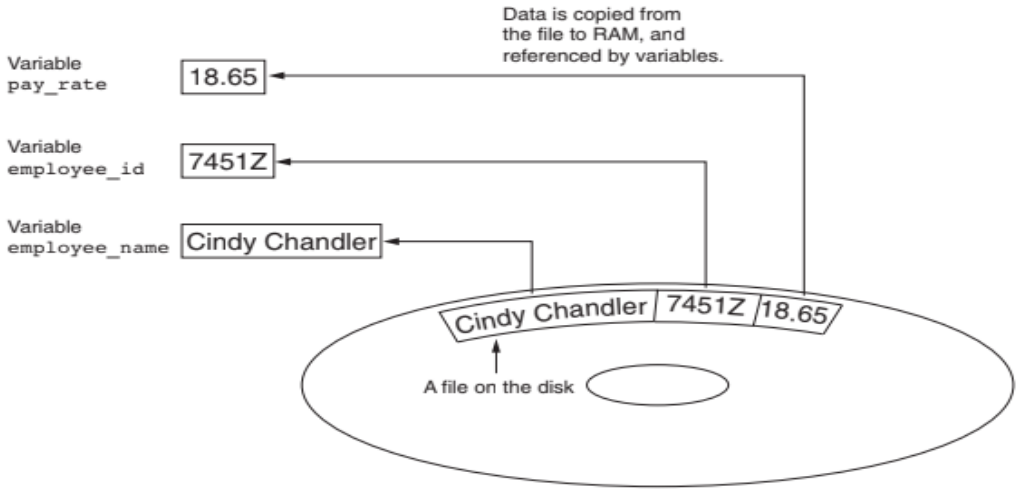
• **Games.** Many computer games keep data stored in files. Web browers. Sometimes when you visit a Web page, the browser stores a small file known as a cookie on your computer. Cookies typically contain information about the browsing session, such as the contents of a shopping cart.



Writing data to a file

The process of retrieving data from a file is known as “reading data from” the file.

When a piece of data is read from a file, it is copied from the file into RAM, and referenced by a variable. The term input file is used to describe a file that data is read from. It is called an input file because the program gets input from the file.



Reading data from a file

There are always three steps that must be taken when a file is used by a program.

1. **Open the file**—Opening a file creates a connection between the file and the program. Opening an output file usually creates the file on the disk and allows the program to write data to it. Opening an input file allows the program to read data from the file.

2. **Process the file**—In this step data is either written to the file (if it is an output file) or read from the file (if it is an input file).

3. **Close the file**—When the program is finished using the file, the file must be closed. Closing a file disconnects the file from the program.

Types of Files

There are two types of files:

- Text
- Binary.

Text File: A text file contains data that has been encoded as text, using a scheme such as ASCII or Unicode. Even if the file contains numbers, those numbers are stored in the file as a series of characters. As a result, the file may be opened and viewed in a text editor such as Notepad.
Binary File: A binary file contains data that has not been converted to text. The data that is stored in a binary file is intended only for a program to read. As a consequence, you cannot view the contents of a binary file with a text editor.

File Access Methods

Two different ways to access data stored in a file:

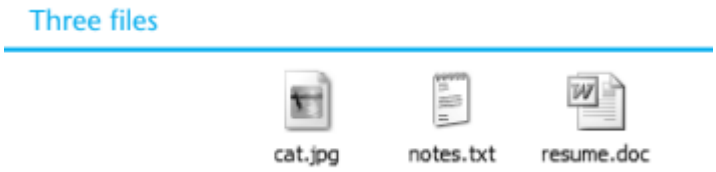
- Sequential access
- Direct access.

sequential access file:

To access data from the beginning of the file to the end of the file. To read a piece of data that is stored at the very end of the file, To read all of the data that comes before it cannot jump directly to the desired data. This is similar to the way cassette tape players work. If you want to listen to the last song on a cassette tape, you have to either fast-forward over all of the songs that come before it or listen to them. There is no way to jump directly to a specific song.

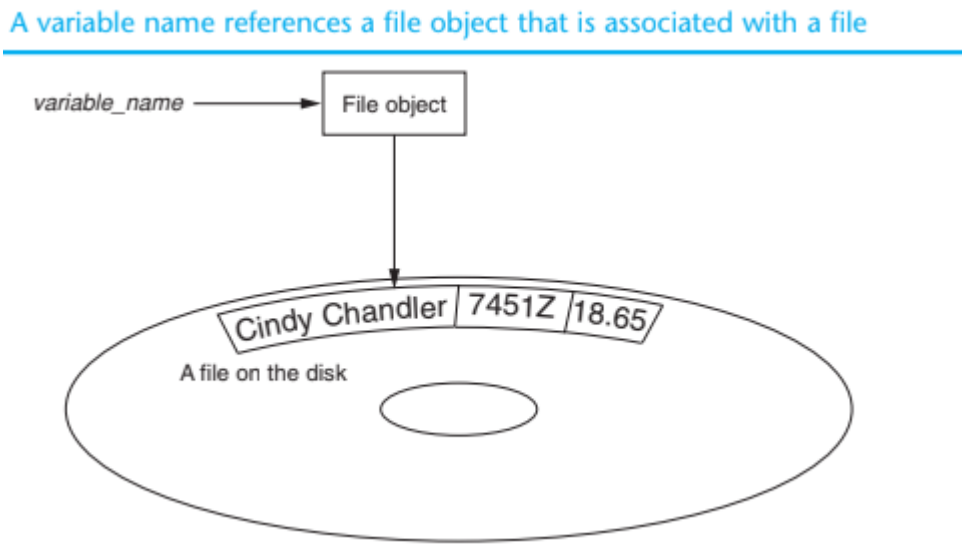
Direct access file (which is also known as a random access file), jump directly to any piece of data in the file without reading the data that comes before it. This is similar to the way a CD player or an MP3 player works. To jump directly to any song that you want to listen to.

Filenames and File Objects



Each operating system has its own rules for naming files. Many systems support the use of filename extensions, which are short sequences of characters that appear at the end of a filename preceded by a period (which is known as a “dot”). The extension usually indicates the type of data stored in the file. For example, the .jpg extension usually indicates that the file contains a graphic image that is compressed according to the JPEG image standard. The .txt extension usually indicates that the file contains text. The .doc extension (as well as the .docx extension) usually indicates that the file contains a Microsoft Word document.

In order for a program to work with a file on the computer’s disk, the program must create a file object in memory. A file object is an object that is associated with a specific file, and provides a way for the program to work with that file. In the program, a variable references the file object. This variable is used to carry out any operations that are performed on the file.



Opening a File

You use the open function in Python to open a file. The open function creates a file object

and associates it with a file on the disk. Here is the general format of how the open function is used:

file_variable = open(filename, mode)

In the general format:

• file_variable is the name of the variable that will reference the file object.

• filename is a string specifying the name of the file.

• mode is a string specifying the mode (reading, writing, etc.) in which the file will be

opened. Table 7-1 shows three of the strings that you can use to specify a mode.

(There are other, more complex modes. The modes shown in Table 7-1 are the ones

we will use in this book.)

Figure 7-4 A variable name references a file object that is associated with a file

Cindy Chandler 7451Z 18.65

A file on the disk

variable_name File object

Table 7-1 Some of the Python file modes

Mode	Description
'r'	Open a file for reading only. The file cannot be changed or written to.
'w'	Open a file for writing. If the file already exists, erase its contents. If it does not exist, create it.
'a'	Open a file to be written to. All data written to the file will be appended to its end. If the file does not exist, create it.

example:

customer_file = open('cusomters.txt', 'r')

Writing Data to a File

For example, file objects have a method named write that can be used to write data to a file.

General format:

file_variable.write(string)

the format, file_variable is a variable that references a file object, and string is a string that will be written to the file. The file must be opened for writing (using the 'w' or 'a' mode) or an error will occur.

Reading Data From a File

If a file has been opened for reading (using the 'r' mode) you can use the file object's read method to read its entire contents into memory. When you call the read method, it returns the file's contents as a string. For example,

```
# This program reads and displays the contents
 def main():
infile = open('philosophers.txt', 'r')
file_contents = infile.read()
infile.close()
print(file_contents)
main()
```

Program Output

John Locke

David Hume

Edmund Burke

Concatenating a Newline to a String

When a program writes data that has been entered by the user to a file, it is usually necessary to concatenate a \n escape sequence to the data before writing it. This ensures that each piece of data is written to a separate line in the file.

```
def main():
print('Enter the names of three friends.')
name1 = input('Friend #1: ')
name2 = input('Friend #2: ')
name3 = input('Friend #3: ')
myfile = open('friends.txt', 'w')
myfile.write(name1 + '\n')
myfile.write(name2 + '\n')
myfile.write(name3 + '\n')
 # Close the file.
 myfile.close()
 print('The names were written to friends.txt.')
# Call the main function.
main()
```

Program Output (with input shown in bold)

Enter the names of three friends.

Friend #1: Joe e

Friend #2: Rose e

Friend #3: Geri e

The names were written to friends.txt.

Appending Data to an Existing File

To use the 'a' mode to open an output file in append mode, which means the following.

- If the file already exists, it will not be erased. If the file does not exist, it will be created.
- When data is written to the file, it will be written at the end of the file's current contents.

For example, assume the file friends.txt contains the following names, each in a separate line:

Joe

Rose

Geri

The following code opens the file and appends additional data to its existing contents.

```
myfile = open('friends.txt', 'a')
myfile.write('Matt\n')
myfile.write('Chris\n')
myfile.write('Suze\n')
myfile.close()
```

After this program runs, the file friends.txt will contain the following data:

Joe

Rose

Geri

Matt

Chris

Suze

Writing and Reading Numeric Data

Strings can be written directly to a file with the write method, but numbers must be converted to strings before they can be written. Python has a built-in function named str that converts a value to a string. For example, assuming the variable num is assigned the value 99, the expression str(num) will return the string '99'.

```
def main():
# Open a file for writing.
outfile = open('numbers.txt', 'w')
```

```
 # Get three numbers from the user.
 num1 = int(input('Enter a number: '))
 num2 = int(input('Enter another number: '))
 num3 = int(input('Enter another number: '))
 # Write the numbers to the file.
 outfile.write(str(num1) + '\n')
 outfile.write(str(num2) + '\n')
 outfile.write(str(num3) + '\n')
 # Close the file.
 outfile.close()
print('Data written to numbers.txt')
# Call the main function.
 main()
```

Program Output

```
Enter a number: 22
Enter another number: 14
Enter another number: 99
```

Using loops to process data

Files usually hold large amounts of data, and programs typically use a loop to process the data in a file. Although some programs use files to store only small amounts of data, files are typically used to hold large collections of data. When a program uses a file to write or read a large amount of data, a loop is typically involved.

```
def main():
# Get the number of days.
num_days = int(input('For how many days do ' + \'you have sales? '))
sales_file = open('sales.txt', 'w')
 # Get the amount of sales for each day and write
 # it to the file.
for count in range(1, num_days + 1):
# Get the sales for a day.
 sales = float(input('Enter the sales for day #' + \
```

```
 str(count) + ': '))
 # Write the sales amount to the file.
 sales_file.write(str(sales) + '\n')
 # Close the file.
 sales_file.close ()
 print('Data written to sales.txt.')
# Call the main function.
 main()
```

Program Output (with input shown in bold)

For how many days do you have sales? 5

Enter the sales for day #1: 1000.0

Enter the sales for day #2: 2000.0

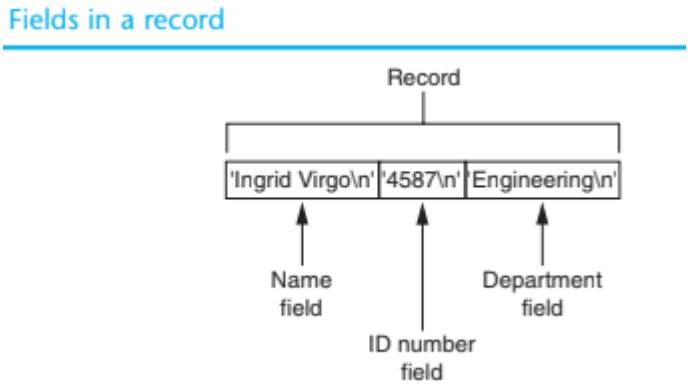
Enter the sales for day #3: 3000.0

Enter the sales for day #4: 4000.0

Enter the sales for day #5: 5000.0

Processing record

The data that is stored in a file is frequently organized in records. A record is a complete set of data about an item, and a field is an individual piece of data within a record.



Each time you write a record to a sequential access file, you write the fields that make up the record, one after the other. For example, a file that contains three employee records. Each record consists of the employee's name, ID number, and department.

```
main():
# Open the employees.txt file.
emp_file = open('employees.txt', 'r')
# Read the first line from the file, which is
# the name field of the first record.
name = emp_file.readline()
# If a field was read, continue processing.
while name != '':
# Read the ID number field.
id_num = emp_file.readline()
# Read the department field.
 dept = emp_file.readline()
 # Strip the newlines from the fields.
 name = name.rstrip('\n')
 id_num = id_num.rstrip('\n')
dept = dept.rstrip('\n')
# Display the record.
 print('Name:', name)
 print('ID:', id_num)
 print('Dept:', dept)
 print()

 # Read the name field of the next record.
 name = emp_file.readline()
 # Close the file.
 emp_file.close()
 # Call the main function.
 main()
```

Program Output

```
Name: Ingrid Virgo
ID: 4587
Dept: Engineering
```



```
Name: Julia Rich
ID: 4588
Dept: Research
Name: Greg Young
ID: 4589
Dept: Marketing
```