In this module we will be covering File input and output, using loops to process files, defining file, record, and field, learn how to process records in a file and how to write exceptions.

File Input and Output

When a program needs to save data for later use, the programmer may write a program to save that data into a "file". The file that contains the saved data then may be used by other programs at a later time.

It is important to remember - programs are instructions and data are values that the program statements need to process the correct information. When you first learn programming you are entering data from the computer each time, but consider that you want to print out the names of 5,000 students at your university. Typing in 5,000 names separately from the keyboard would be EXTREMELY time consuming. And what if you have several programs that need those 5,000 student names for other things as well.

We need to be able to store those 5,000 student names (and any other data we need) into a permanent storage area that we can reuse over and over. This is the concept of creating a file.

Definitions

Field - a piece of data - such as a person's first name, or last name, or age, or street address, etc.

Record - a collection of related fields - such as a Student record contains all of the individual pieces of data about that student = last name, first name, middle initial, age, street address, city, state, zip, cell phone, home phone, etc.

File - a collection of related records - this would be the whole group of individual student records stored together.

Think of a file cabinet (file), inside the cabinet - a collection of individual "file" folders (records), inside each individual file folder pieces of data (fields).

Types of Files

There are two types of files - text and binary. Text files contains data that has been stored as text - as a result the file may be opened (accessed) and viewed in a text editor - such as Notepad. A Binary file stores data that is binary format.

File Access Methods

Programs normally create files in two formats - sequential file access, or direct file access.

Sequential file access means that you have a file that contains sequential records physically one right after the other. Thus, is you really only need the 4th record in the file, you need to first "read" the contents of the first three records before you can get to the 4th record. This is like a cassette tape where you have to fast forward to get to a particular record (you are reading all of the other songs, just faster). The computer system keeps track of where your file begins (location of the first record in the file) in an index area.

Direct file access (also called random access) means that the individual records are stored physically in different locations on the storage area. The computer system keeps track of each individual record's location so you can "jump" directly to that record without having to "read" the previous records in the file. This is like a CD where you can jump to a particular song without having to "read" the other previous records in the file.

Filenames and File Objects

We have several types of files stored on our computers. Application Programs - such as word processors, etc. have a list of files that you can use. Operating systems have their own rules for names files - normally with an "extension", such as .jpg, .txt, .doc, .docx etc.

You can call the file where you are storing data a File Object - the space that is associated with a particular file name that has addresses associated with it for locating the records and data inside the file.

Creating a File

It takes a program to create a file. The program needs to get each piece of individual data for one Person or another type of data we want to store. The program "reads" that individual pieces of data - *input* - normally it is still inputted through the keyboard - into the variables for the data - last_name, first_name, etc. These variables will be part of the record definition in your program. The order of your variables in your record description is the order that the fields (variables) will be stored in each record. ORDER IS IMPORTANT!! Once the program has read ONE record of data (all of the data for a particular Person, etc.), the program then outputs that ONE record of data (that contains all of the individual pieces of data in separate variables) to a permanent storage area - such as a disc.

You then go back and input the next Person's individual pieces of data and then output that "record" to the storage area. You are outputting a series of RECORDS that contain individual FIELDS (VARIABLES) to the permanent store area the FILE.

You will create the name of your file, and the names of your fields (variables).

On page 240-242 you will find an example of writing data to a file.

Opening a file

Once you have created your file - outputted the data to your storage device, you can then use that particular file in any program you wish. One thing you need to remember, of course, is the order of your various fields (variables) that were created for that particular file. If you created the file with the following variables in this order: last_name, first_name, street, city, zip, age, then you need to "redefine" those same variable names (fields) in the same order: last_name, first_name, street, city, zip, age.

When you are now going to use your file in a new program, you need to be sure your file definition is he same as the one where you created the file in the first place.

Opening a file means that you want to use that file in your program. The SYNTAX for opening a file is:

file_variable = open*(filename, mode)*

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

filename is a string specifying the name of the file

mode is a string specifying how you will use this file in your program:

"r" open a file for reading only (input) - the file CANNOT be changed to written to

"w" open a file for writing (output) - if the file already exists, erase its contents. If it does not exist, create it (this is used to create a new file as stated above

"a" open a file to be written to the file will be appended to its end

example (of creating a new file): sales_file - open('sales.txt', 'w') remember the string single quotes

Writing Data to the File

Since our data file is called a file object - we have associated functions with objects called "method". A "method" is a function that belongs to an object and performs some operating using that object. Once you have opened a file, you can use the file object's methods to perform operations on the file.

One such method is "write":

file_variable.write(string)

customer_file.write('Charles Pace')

In PYTHON it is necessary to use the \n character at the end of your piece of data - variable in order to get each piece of data, or at the end of a group of variables (record) so that each variable, or record will be a separate entity in the file.

When we have stored all of our data (records), the system will automatically create an "end of file" mark at the end of the file.

Closing the file

Once a program is finished working with a file, it should close the file. Closing a file disconnects it from being used in the program. We have the "close" method:

customer_file.close()

Reading from a File

Once we have created a file - that has been permanently stored, we can then use the file for other programs. We need to open the file in our program so we can use it:

infile = open('philosopher.txt', 'r')

file_contents = infile.read() This will read the complete contents of the file!!!

Normally, we want to read the individual records one at a time - we do this through the readline() method.

infile = open('philosophers.txt', 'r')

line1 = infile.readline()

line2 = infile.readline()

etc.

If we wrote our file using the \n end of data concatenation, we need to "remove" it after reading our data in so we don't use it in our program.

name = 'Joanne Manchester\n'

name = name.rstrip('\n)

Writing and Reading Numeric Data

Numbers MUST be converted to strings before reading them to a file. We use the str function to do this.

Using Loops to Process Files

We will be using loop structures to process both "writing" - creating a new file, or "reading" from an existing file. Since we have multiple records in our files, we need to read one record at a time, process that record, and then go back and get the next record and process it, etc.

We stop the loop by looking for the "end of file" mark - an empty string.

Exceptions

An exception is an error that occurs while a program is running, causing the program to abruptly halt. We want to "handle" these exception errors. We use the *try/except statement* to handle exception errors. We can do this with if-else statement as well.

We also have as part of the try/except statement is the finally clause:

try:

    statement

    statement

    etc.

except ExceptionName:

    statement

    statement

    etc.

finally:

    statement

    statement

    etc.