CHAPTER 6

Files and Exceptions

Topics

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

Introduction to File Input and Output 1/4

- Storage of data in variables, arrays, and structures is temporary (RAM)-such data is *lost* when a program terminates.

- Files are used for *permanent retention* of data. Computers store files on secondary storage devices, especially disk storage devices.

- In this chapter, we explain how data files are created, updated and processed with Python.

Introduction to File Input and Output 2/4

Three steps to be followed when a program uses a file

Open the file: A file must be opened before processing/accessing.

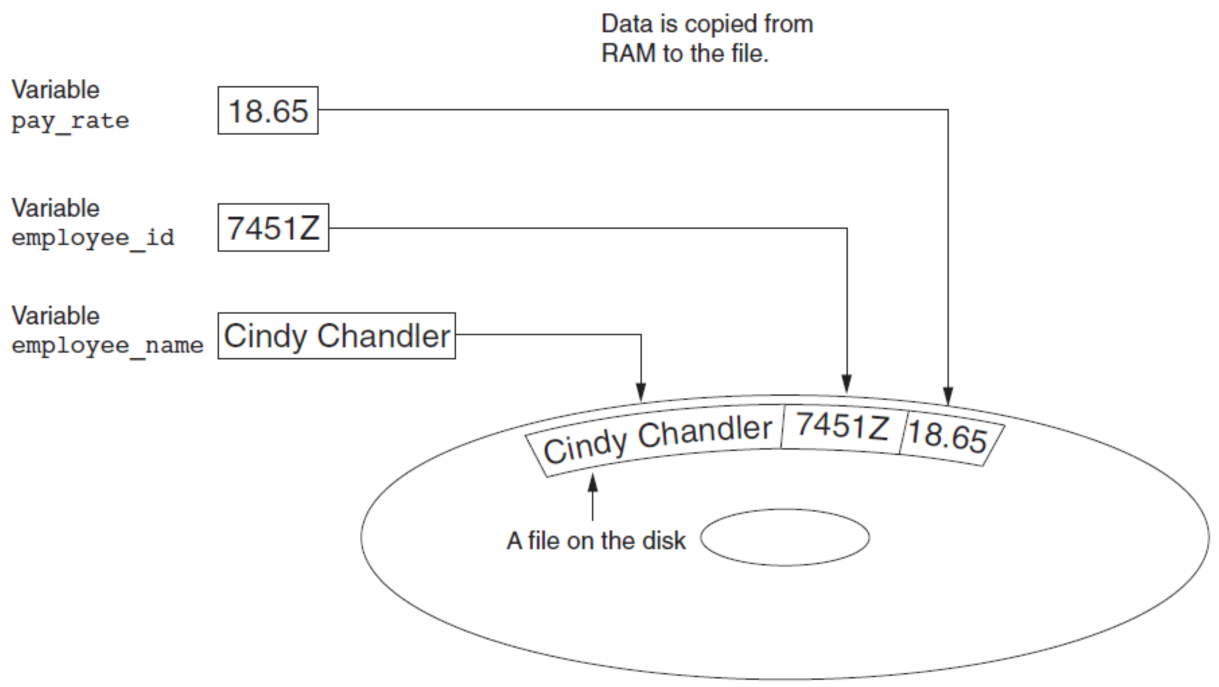
Process the file: After opening a file you can process the file.
It could be writing/storing some data to file or reading/retrieving some data from the a file.

Close the file: A file should be closed after finishing processing the file.

Introduction to File Input and Output 3/4

- **For program to store 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
- **"<u>Writing data to</u>": saving data on a file**
- **<u>Output file</u>: a file that data is written to**

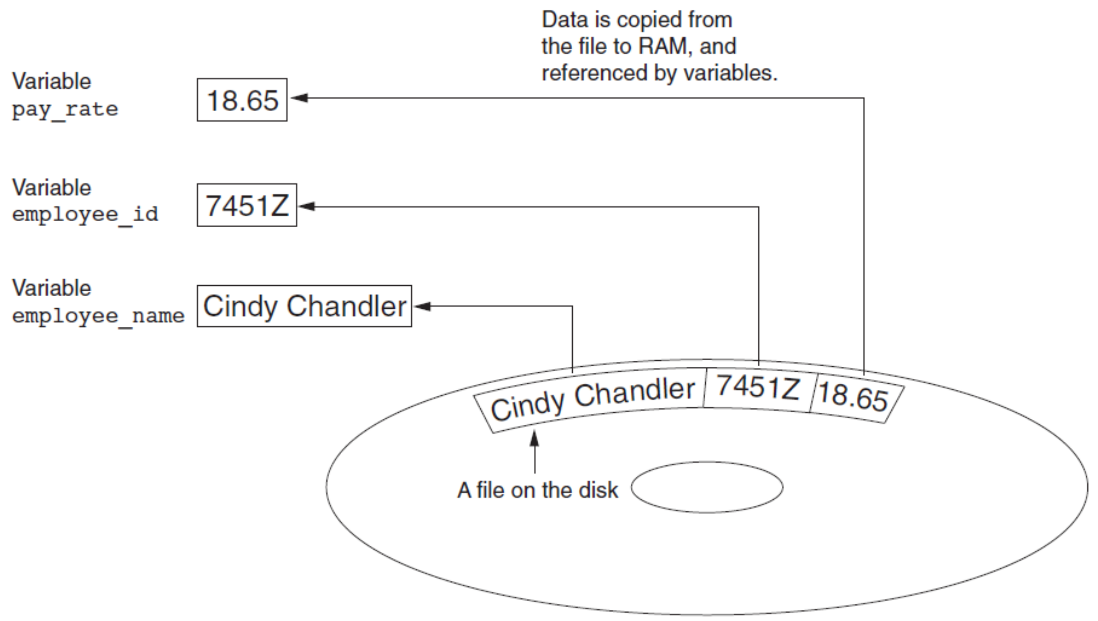
Figure 6-1 Writing data to a file



Introduction to File Input and Output 4/4

- "**Reading data from**": process of retrieving data from a file
- **Input file**: a file from which data is read

Figure 6-2 Reading data from a file



Types of Files and File Access Methods

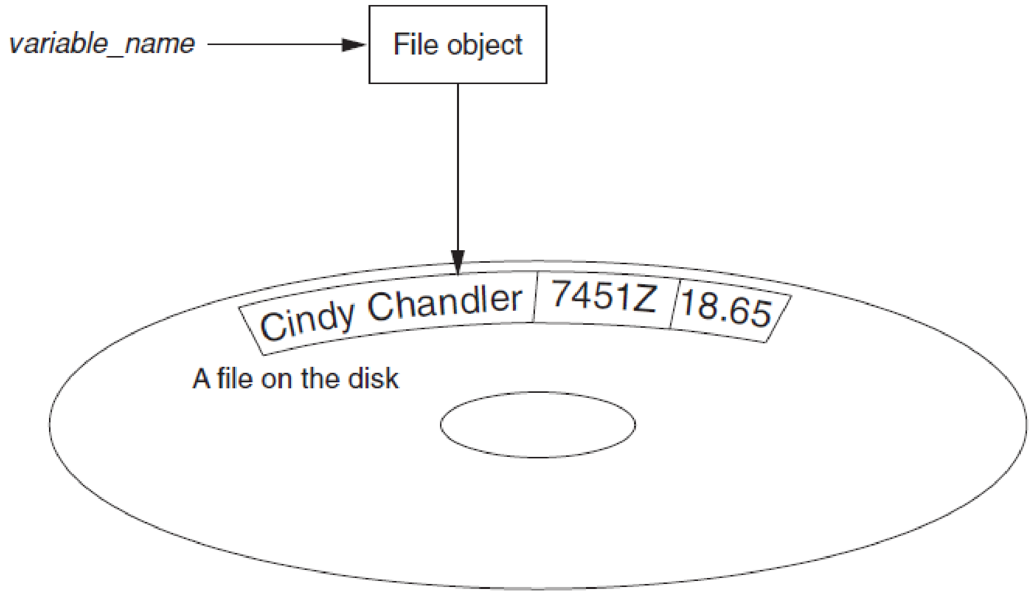
- **In general, two types of files**
  - **<u>Text file</u>:** contains data that has been encoded as text
  - **<u>Binary file</u>:** contains data that has not been converted to text
- **Two ways to access data stored in file**
  - **<u>Sequential access</u>:** file read sequentially from beginning to end, can't skip ahead – have to go in the order the data is written to it.
  - **<u>Direct access</u>:** can jump directly to any piece of data in the file – also known as random access files.

As an analogy, cassette players serve like a sequential access whereas mp3 players serve like a direct access.

Filenames and File Objects

- **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 – Python Syntax:

```
file_object = open(filename, mode)
```

- **Mode: string specifying how the file will be opened**

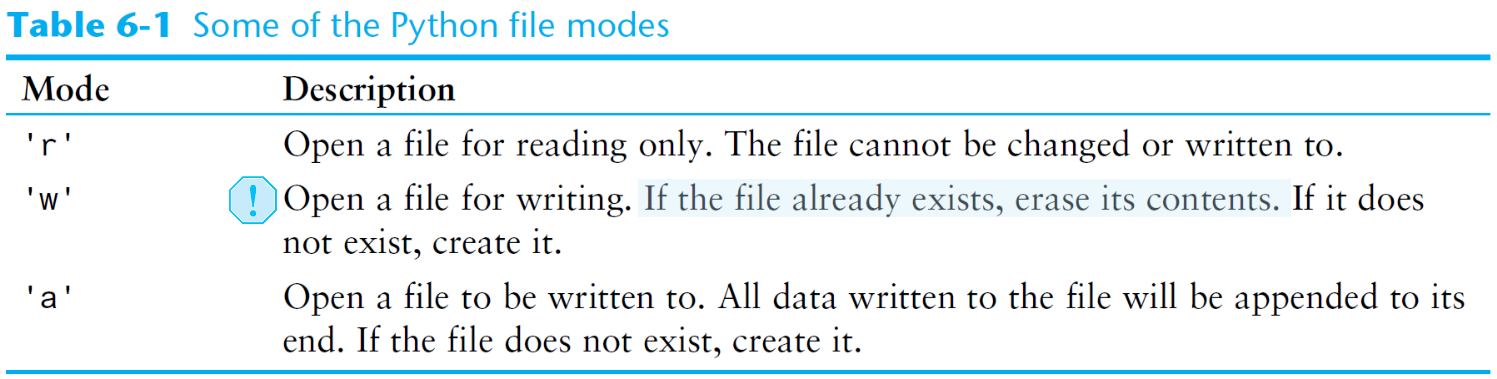


Table 6-1 Some of the Python file modes

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: Opening the file `customers.txt` contains customer data, and we want to open it for reading (file has be in the same directory):

```
customer_file = open('customers.txt', 'r')
```

Opening another file in a specific location named `test.txt` to write

```
test_file = open('C:\Users\Blake\temp\test.txt', 'w')
```

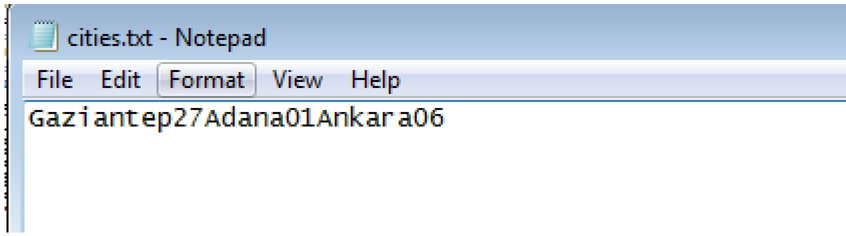
Writing Data to a File 1/2

- **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()`

Example Program:

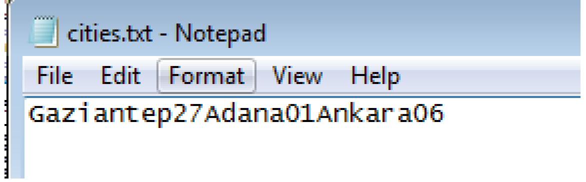
```
def main():
      city_file = open('cities.txt', 'w')
      city='Gaziantep'
      plate=27
      city_file.write(city)
      city_file.write(str(plate))
      city_file.write('Adana')
      city_file.write('01')
      city_file.write('Ankara')
      city_file.write('06')
      city_file.close() #Closing the file
#Calling Main Function
main()
```

At the end of the Program:



```
Gaziantep27Adana01Ankara06
```

Writing Data to a File 2/2



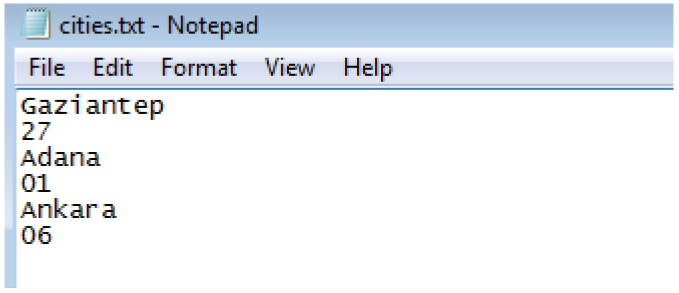
```
Gaziantep27Adana01Ankara06
```

- **If we need to read from such file to process the content – it is quite difficult for a program to identify the data in it.**
- **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

Example Program:

```
def main():
        city_file = open('cities.txt', 'w')
        city='Gaziantep'
        plate=27
        city_file.write(city + '\n')
        city_file.write(str(plate) + '\n')
        city_file.write('Adana\n')
        city_file.write('01\n')
        city_file.write('Ankara\n')
        city_file.write('06\n')
        city_file.close() #Closing the file
#Calling Main Function
main()
```

At the end of this Program:



```
Gaziantep
27
Adana
01
Ankara
06
```

Note: **Data written file is** `Gaziantep\n27\nAdana\n01\nAnkara\n06\n` **but Notepad processes the \n newline characters and shows as data as in separate lines.**

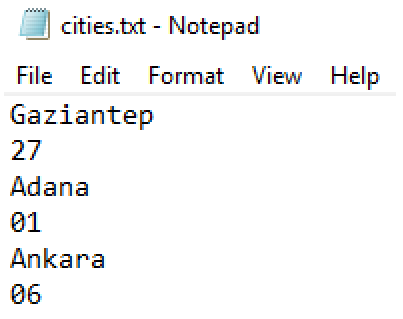
Reading Data From a File

The following three file object methods can be used to read 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**

Reading Data From a File – `read` method

- **`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



```
Gaziantep
27
Adana
01
Ankara
06
```

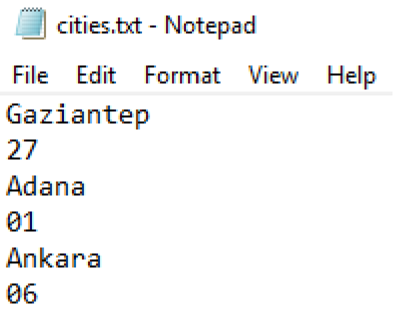
```python
# This program demonstrates how to read
# the complete content of the file
def main():
        city_file = open('cities.txt', 'r')
        # Read the file's contents.
        content=city_file.read()
        print('CONTENT OF THE FILE cities.txt')
        print(content)
        city_file.close()
#Calling Main Function
main()
```

Program Output

```
CONTENT OF THE FILE cities.txt
Gaziantep
27
Adana
01
Ankara
06
```

Reading Data From a File – `readline` method

- **`readline` method: file object method that reads a line from the file**
  - Line returned as a string, including `'\n'`



```
Gaziantep
27
Adana
01
Ankara
06
```

```
# This program demonstrates how to read
# the complete data line by line from a file
def main():
        city_file = open('cities.txt', 'r')
        # Read the 6 lines from the file contents.
        line=city_file.readline()
        print(line)
        line=city_file.readline()
        print(line)
        line=city_file.readline()
        print(line)
        line=city_file.readline()
        print(line)
        line=city_file.readline()
        print(line)
        line=city_file.readline()
        print(line)
        city_file.close()
#Calling Main Function
main()
```

Program Output

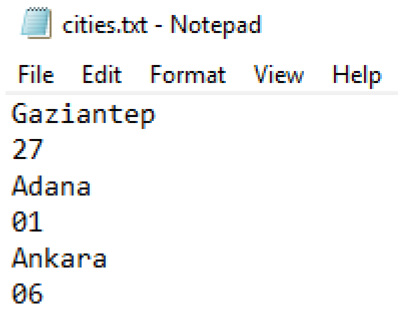
```
Gaziantep

27

Adana

01

Ankara

06
```

Note: **Spaces between each line is due to \n in the data and \n from print function.**

Stripping a Newline Character From a String

- **In many cases need to remove `'\n'` from string after it is read from a file**
  - **<u>`rstrip` method</u>**: string method that strips specific characters from end of the string

```
# This program demonstrates how to strip
# \n from a string read from a file
def main():
        city_file = open('cities.txt', 'r')
        # Read only 2 lines from the file
        line1=city_file.readline()
        line2=city_file.readline()
        print(line1,line2) # Print both
        # Now let's strip \n from the lines
        line1=line1.rstrip() #stripping from line1
        line2=line2.rstrip() #stripping from line2
        print(line1,line2) # Print both
        city_file.close()
#Calling Main Function
main()
```



```
Gaziantep
27
Adana
01
Ankara
06
```

Program Output

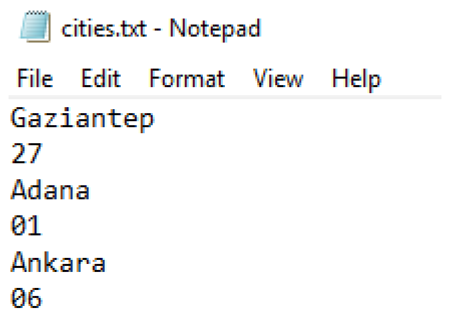
```
Gaziantep
 27

Gaziantep 27
```

Appending Data to an Existing File

- **When open file with `'w'` mode, if the file already exists it is overwritten – all data is lost/deleted.**
- **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
  - In other words, read position mars the ned.

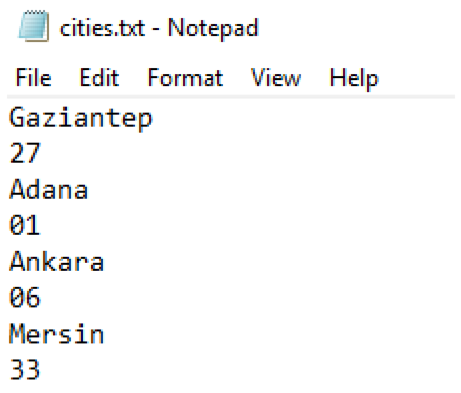
```
# This program demostrates how to append new data
# to an existing file
def main():
        # File must be opened in 'a' mode.
        city_file = open('cities.txt', 'a')
        city='Mersin'
        plate=33
        city_file.write(city + '\n')
        city_file.write(str(plate) + '\n')
        city_file.close() # Closing the file
#Calling Main Function
main()
```

Before:



```
Gaziantep
27
Adana
01
Ankara
06
```

After:



```
Gaziantep
27
Adana
01
Ankara
06
Mersin
33
```

Writing and Reading Numeric Data

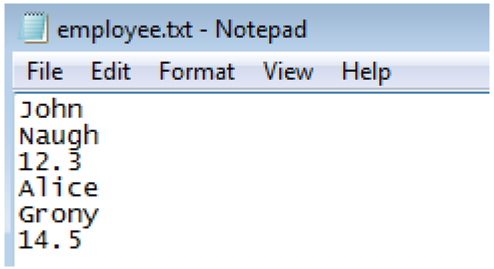
- **Numbers must be converted to strings before they are written to a file**
- **`str` function: converts value to string**
- **Number are read from a text file as strings**
  - Must be converted to numeric type in order to perform mathematical operations
  - Use `int` and `float` functions to convert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

Example: Reading Numeric Data and Processing

Consider that a file named `employee.txt` **includes two employee names, last names and the numbers of hours worked. Each info is separated by** `\n` **character. Let's now write a program that reads these info and shows these info in tabular form. At the end, your program should display the total number of hours.**

Content of the file employee.txt

```
John\nNaugh\n12.3\nAlice\nGrony\n14.5\n
```



```
John
Naugh
12.3
Alice
Grony
14.5
```

```python
# This program demonstrates reading numberical
# data and processing numberical data
def main():
        emp_file = open('employee.txt', 'r')
        total_hour=0
        # Read the file using readline method
        name=emp_file.readline()
        last=emp_file.readline()
        hour=emp_file.readline()
        name = name.rstrip()
        last = last.rstrip()
        hour = float(hour.rstrip())
        print(name,'\t',last,'\t',hour)
        total_hour+=hour
        name=emp_file.readline()
        last=emp_file.readline()
        hour=emp_file.readline()
        name = name.rstrip()
        last = last.rstrip()
        hour = float(hour.rstrip())
        print(name,'\t',last,'\t',hour)
        total_hour+=hour
        print('Total hour:',format(total_hour,'.2f'))
        emp_file.close()
#Calling Main Function
main()
```

Program Output

```
John        Naugh     12.3
Alice       Grony     14.5
Total hour: 26.80
```

Checkpoint

6.1 – 6.2 What is an output file and input file?

an output file that data is written to and an intput file from which data is read

6.3 What three steps must be taken by a program when it uses a file?

Opening the file, Processing the file and Closing the file.

6.4 In general, what are the two types of files? What is the difference between these two types of files?

Text and Binary files. In the text files, content is encoded as text so we browse by using text editors like Notepad.

6.5 What are the two types of file access? What is the difference between these two?

Sequential and Direct Access. Sequential access is done processing line by line. In the DA, you can jump to any data.

6.6 When writing a program that performs an operation on a file, what two file associated names do you have to work with in your code?

filename and file_object. We use file_object associated with the file to access to the file.

6.7 If a file already exists, what happens to it if you try to open it as an output file (using the 'w' mode)?

If the file already, then 'w' mode erases the content of the file.

6.8 What is the purpose of opening a file?

To access a file for processing, we need to open the file.

6.9 What is the purpose of closing a file?

At the end, we must close the file. Closing a file disconnects the file from the program.

6.10 What is a file's read position? Initially, where is the read position when an input file is opened?

A file's read position marks the location of the next item that will be read from the file. Initially, the read position is set to the beginning of the file.

6.11 In what mode do you open a file if you want to write data to it, but you do not want to erase the file's existing contents? When you write data to such a file, to what part of the file is the data written?

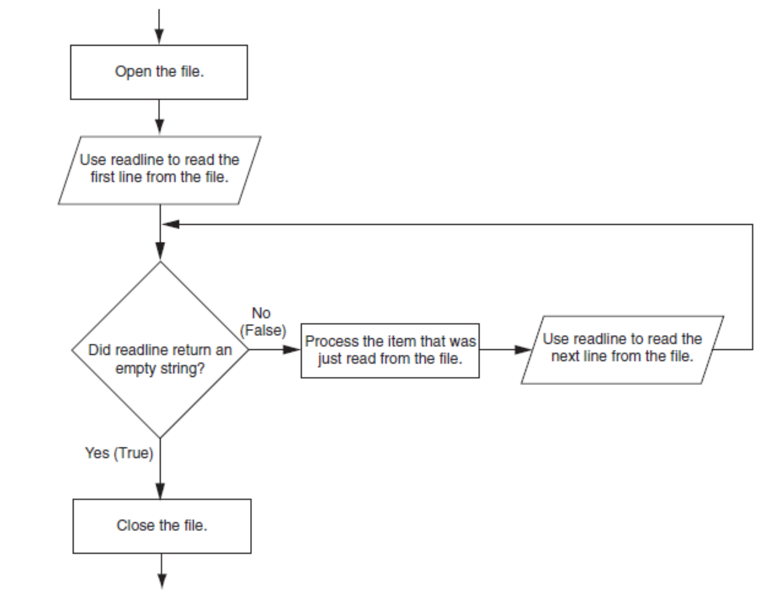
In 'a' mode, we open a file to be written to. All data written to the file is appended to its end.

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 returns/gives an empty string as a sentinel when end of file is reached
    - This information can be used in a conditional-repetition to read all the data from beginning to the end
    - Can write a while loop with the condition

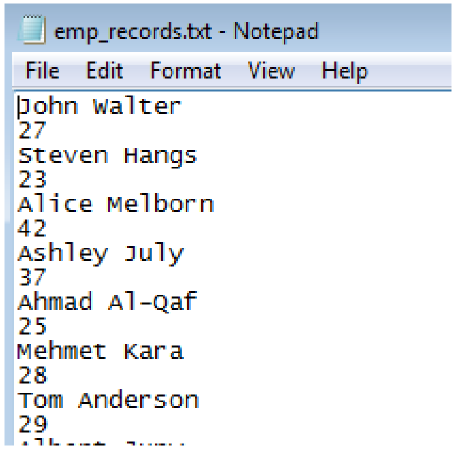
```
while line != ''
```

Figure 6-17 General logic for detecting the end of a file


Open the file.
Use readline to read the first line from the file.
Did readline return an empty string?
No (False)
Process the item that was just read from the file.
Use readline to read the next line from the file.
Yes (True)
Close the file.

Example: Using Loops to Process Files

Consider that a file named `emp_records.txt` includes the names and ages for an unknown number employees. Each info is separated by `\n` character. Write a program that lists all the information in tabular form and at the end of your program it also displays the average of the employee ages.



```
John Walter
27
Steven Hangs
23
Alice Melborn
42
Ashley July
37
Ahmad Al-Qaf
25
Mehmet Kara
28
Tom Anderson
29
```

Content of the file emp_records.txt

```
John Walter\n27\nSteven Hangs\n23\nAlice Melborn\n42\n...
```

```python
# This program demonstrates reading
# and processing a file by using while loop
def main():
        emp_file = open('emp_records.txt', 'r')
        total_age=0
        noe=0  #Number of employee - needed for average
        # Read the file's contents.
        name=(emp_file.readline()).rstrip()
        while name !='':
                age=int(emp_file.readline())
                total_age+=age
                noe+=1
                print(name,'\t','\t',age)
                # Now reading name for the next data
                name=(emp_file.readline()).rstrip()
        #Done with reading all info - print the average
        print('Age Average of',noe,'employees',format(total_age/noe,'.2f'))
        emp_file.close()
#Calling Main Function
main()
```

Program Output

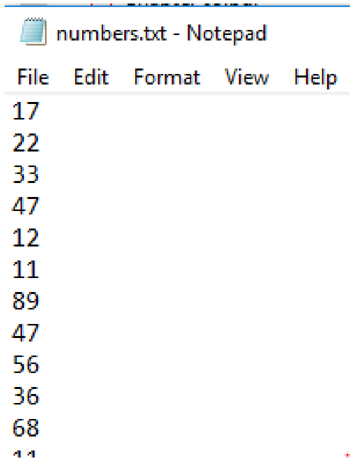
```
John Walter                 27
Steven Hangs                23
Alice Melborn               42
Ashley July                 37
Ahmad Al-Qaf                25
Mehmet Kara                 28
Tom Anderson                29
Albert Jury                 34
Janet Quincy                37
Age Average of 9 employees 31.33
```

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

Example: Using `for` Loop to Process a File

A file named `numbers.txt` includes an unknown number of integer numbers. Each number is separated by `\n` character. Write a program that displays all the numbers in a line then the average of the numbers.



```
17
22
33
47
12
11
89
47
56
36
68
```

Content of the file numbers.txt

```
17\n22\n33\n47\n12\n11\n89\n47\n56\n36...
```

```
# This program demonstrates reading
# and processing a file by using for loop
def main():
        num_file = open('numbers.txt', 'r')
        total=0
        non=0  #Number of numbers
        # Processing file_object by using for repetition.
        for number in num_file:
                number=int(number)
                total+=number
                non+=1
                print(number,end=' ') # Printing all in one line
        #Done with reading all info - print the average
        print('\nAge Average of',non,'numbers',format(total/non,'.2f'))
        num_file.close() #Closing the file
#Calling Main Function
main()
```

$\longrightarrow$ **We don't need to strip '\n'**
int or float gets rid of '\n' during conversion.

Program Output

```
17 22 33 47 12 11 89 47 56 36 68 11 79
Age Average of 13 numbers 40.62
```

Checkpoint

6.12 Write a short program that uses a for loop to write the numbers 1 through 10 to a file.

```
for i in range(1,11):
       file_object.write(str(i)+'\n')
```

6.13 What does it mean when the `readline` method returns an empty string?

it means the end of the file is reached.

6.14 Assume the file `data.txt` exists and contains several lines of text. Write a short program using the while loop that displays each line in the file.

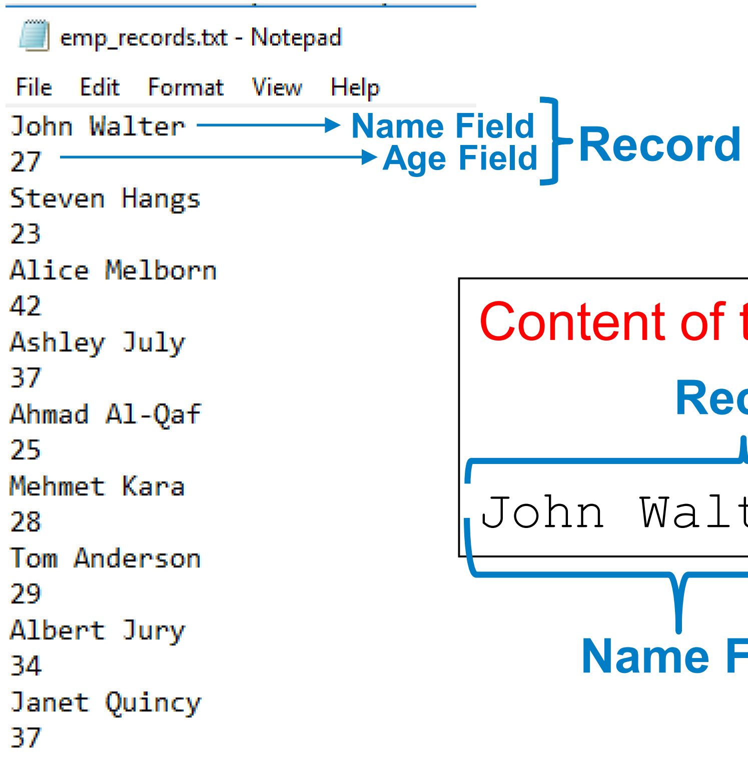
```
data_file = open(data.txt,'r')
text=data_file.readline()
while text != '':
       text=text.rstrip('\n')
       print(text)
       text=data_file.readline()  # Reading the next line
```

6.15 Revise the program that you wrote for **Checkpoint 6.14** to use the for loop instead of the while loop.

```
data_file = open(data.txt,'r')
for line in data_file:
       line=line.rstrip('\n')
       print(line)
```

Processing Records

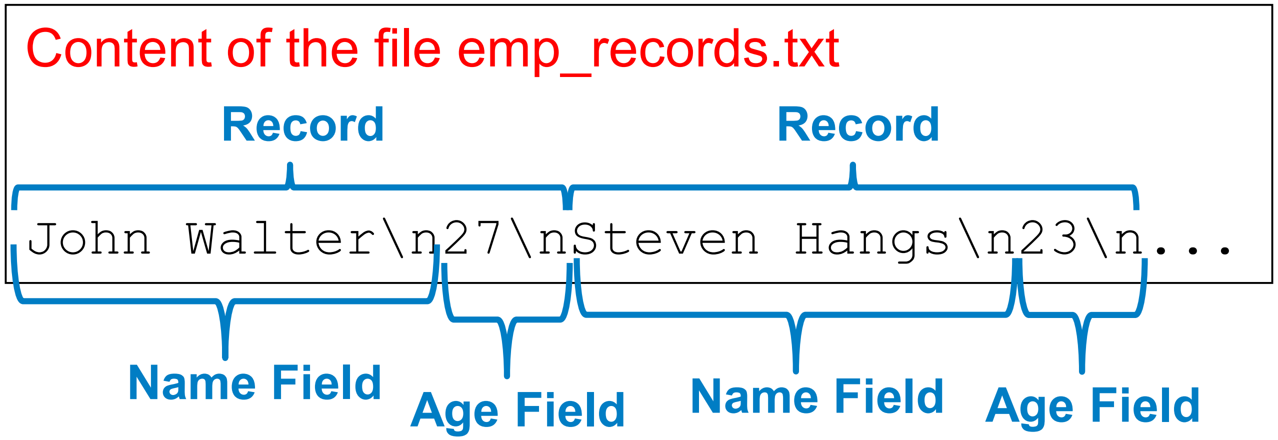
- **Record**: set of data that describes one item
- **Field**: single piece of data within a record



emp_records.txt - Notepad

File Edit Format View Help

```
John Walter
27
Steven Hangs
23
Alice Melborn
42
Ashley July
37
Ahmad Al-Qaf
25
Mehmet Kara
28
Tom Anderson
29
Albert Jury
34
Janet Quincy
37
```



Content of the file emp_records.txt

```
John Walter\n27\nSteven Hangs\n23\n...
```

In a sequential Access File

- Write record by writing the fields one after the other
- Read record by reading each field until record complete

Processing Records (cont'd.)

- **When working with records, it is also important to be able to:**
  - **Create records for first time**
    - To creating records, file must be opened in `'w'` mode
  - **Add-Append records**
    - To add record, file must be opened in `'a'` mode
  - **Display records**
    - To display record/s, file must be opened in `'r'` mode
  - **Search for a specific record**
    - To search record/s, file must be opened in `'r'` mode
  - **Modify records**
    - To display record/s, file must be opened in `'r'` mode
    - **(see example:** `modify_coffee_records.py` **at Page 342)**
  - **Delete records**
    - Deleting could be done by backing up the data with excluding the one/s to be skipped.

Example: Create employees.txt – Add record

Write a program that creates `employees.txt` and add a certain number of records given by the user. Employee record should include the information: <u>name</u>, <u>id</u> and <u>department</u> in which the employee works.

```python
# This program gets employee data from the user and
# saves it as records in the employees.txt file.
def main():
    # Get the number of employee records to create.
    num_emps = int(input('How many employee records?'))

    # Open a file for writing.
    emp_file = open('employees.txt', 'w')

    # Get each employee's data and write it to the file
    for count in range(1, num_emps + 1):
        # Get the data for an employee.
        print('\nEnter data for employee #', count, sep='')
        name = input('Name: ')
        id_num = input('ID number: ')
        dept = input('Department: ')

        # Write the data as a record to the file.
        emp_file.write(name + '\n')
        emp_file.write(id_num + '\n')
        emp_file.write(dept + '\n')

    # Close the file.
    emp_file.close()
    print('Employee records written to employees.txt.')

# Call the main function.
main()
```

$\rightarrow$ **`num_emps` keeps how many records to be entered.**

$\rightarrow$ **Opening `employees.txt` File in 'w' mode**

$\rightarrow$ **Repeating `num_emps` times.**

Getting data for a Record

Writing data for a Record

$\rightarrow$ **Closing `employees.txt` File**

Example: Create employees.txt – Add record

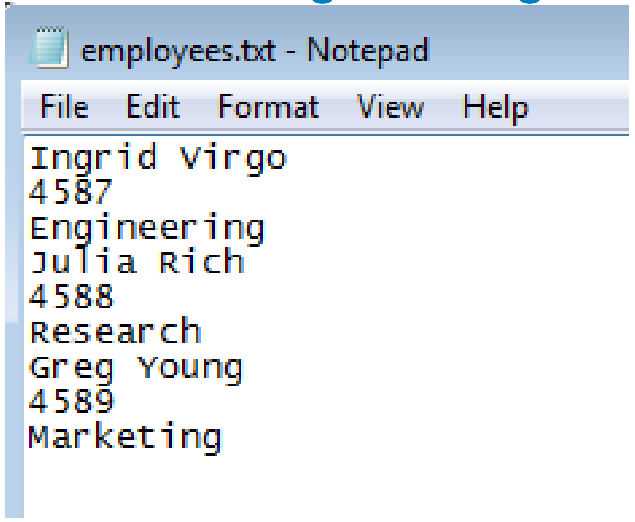
Write a program that creates `employees.txt` and add certain number of records given by the user. Employee record should include the information: <u>name</u>, <u>id</u> and <u>department</u> in which the employee works.

Program Run:

Program Output (with input shown in bold)

```
How many employee records? 3 [Enter]

Enter the data for employee #1
Name: Ingrid Virgo [Enter]
ID number: 4587 [Enter]
Department: Engineering [Enter]

Enter the data for employee #2
Name: Julia Rich [Enter]
ID number: 4588 [Enter]
Department: Research [Enter]

Enter the data for employee #3
Name: Greg Young [Enter]
ID number: 4589 [Enter]
Department: Marketing [Enter]

Employee records written to employees.txt.
```

After Running the Program



```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
```

Example: Add-Append Record to employees.txt

Write a program that appends a single record to `employees.txt`.

Again employee record consist of <u>name</u>, <u>id</u> and <u>department</u> information.

```
# This program gets an employee data from the user and
# appends it as records to the employees.txt file.
def main():

    # Open a file for writing.
    emp_file = open('employees.txt', 'a')

    # Get the data for the employee.
    print('Enter data for the employee')
    name = input('Name: ')
    id_num = input('ID number: ')
    dept = input('Department: ')

    # Write the data as a record to the file.
    emp_file.write(name + '\n')
    emp_file.write(id_num + '\n')
    emp_file.write(dept + '\n')

    # Close the file.
    emp_file.close()

# Call the main function.
main()
```

Opening `employees.txt` File
File is being opened in append 'a' mode!

Getting data for the Record

Writing data for the Record

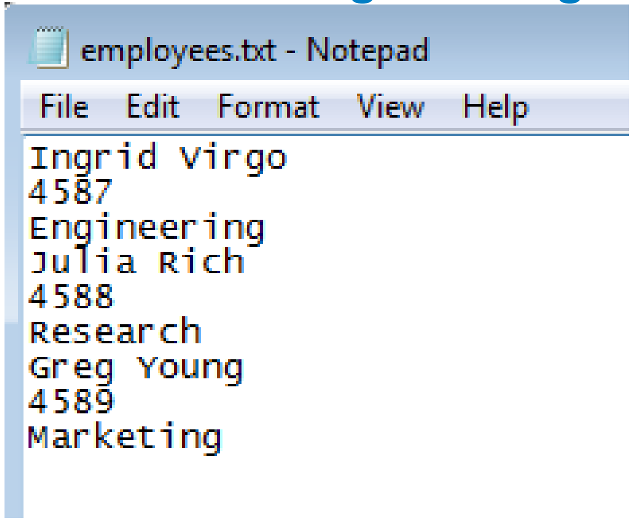
Closing `employees.txt` File

Example: Add-Append Record to employees.txt

Write a program that appends a single record to `employees.txt`.

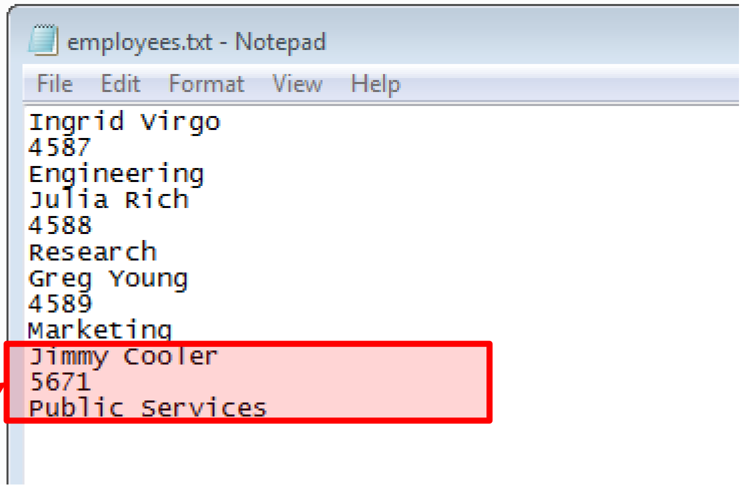
Again employee record consist of <u>name</u>, <u>id</u> and <u>department</u> informations.

Before Running the Program



```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
```

After Running the Program



```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
Jimmy Cooler
5671
Public Services
```

New record is appended to the end!

Program Run:

```
Enter data for the employee
Name: Jimmy Cooler
ID number: 5671
Department: Public Services
```

Example: Displaying Records in employees.txt

Write a program that displays-reads the records in the `employees.txt`.

```
# This program displays the records that are
# in the employees.txt file.

def 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()
```

Opening `employees.txt` File
File is being opened in read 'r' mode!

Reading first line/field from the file which is name field.

Using name field for sentinel controlled repetition

Reading rest of the fields in the record

Stripping '\n' character from each field
Then displaying the record

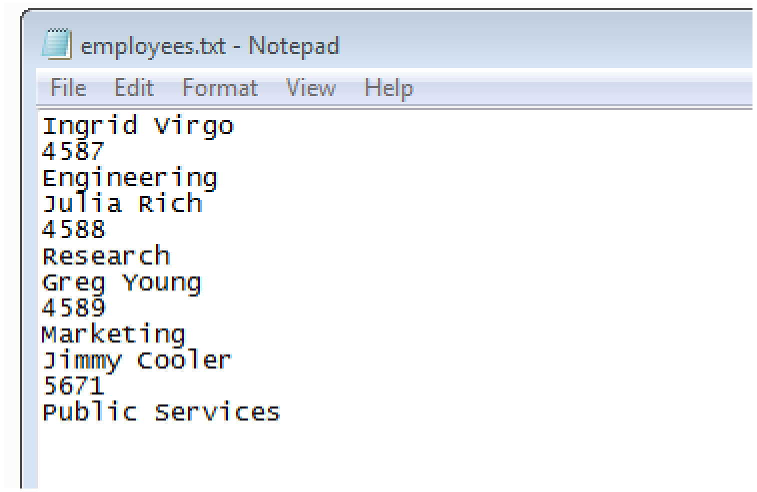
Reading name field from of the next record

Closing `employees.txt` File

Example: Displaying Records in employees.txt

Write a program that displays-reads the records in the `employees.txt`.

Remember content of employees.txt



```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
Jimmy Cooler
5671
Public Services
```

Program Run:

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

Name: Julia Rich
ID: 4588
Dept: Research

Name: Greg Young
ID: 4589
Dept: Marketing

Name: Jimmy Cooler
ID: 5671
Dept: Public Services
```

Here records are displayed in row.
One may also print them in tabular form too.

Example: Searching Records in employee.txt

Write a program that searches the records in the `employees.txt` for a given employee id.

```
# This program searches for a specific id ...
def main():
    found = False  # To check if found or not
    id = input('Enter the id to search:')
    # 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')
        if id == id_num: # Display the record.
            print('Name:', name)
            print('ID:', id_num)
            print('Dept:', dept)
            found = True

        # Read the name field of the next record.
        name = emp_file.readline()

    # Close the file.
    emp_file.close()
    if not found:
        print(id,'can not be found in the database.')

# Call the main function.
main()
```

Defining a Boolean variable to see found or not.

Getting id number to be search from the user.

Reading/Processing All Records

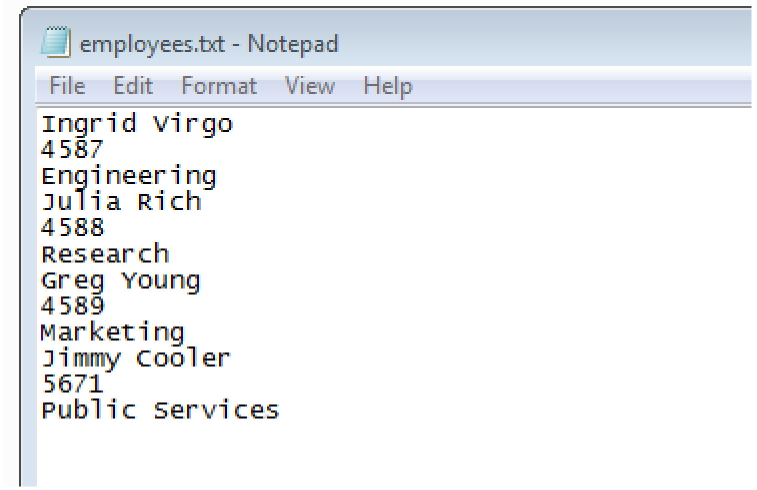
Displaying the record if it's id same as the entered id. Assign `True` to `found` variable.

If `found` is `False`, prints no data is found!

Example: Searching Records in employees.txt

Write a program that searches the records in the `employees.txt` for a given employee id.

Remember content of employees.txt



```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
Jimmy Cooler
5671
Public Services
```

Example Program Runs:

```
Enter the id to search:1001
1001 can not be found in the database.
```

```
Enter the id to search:4589
Name: Greg Young
ID: 4589
Dept: Marketing
```

```
Enter the id to search:3587
3587 can not be found in the database.
```

```
Enter the id to search:5671
Name: Jimmy Cooler
ID: 5671
Dept: Public Services
```

Example: Deleting Records in employees.txt

Write a program that deletes a record in the `employees.txt` for a given employee id.

```
# This program allows the user to delete a record in the employees.txt file.
import os  # Needed for the remove and rename functions
def main():
    # Create a bool variable to use as a flag.
    found = False
    # Get the employee id to be deleted.
    id = input('Employee id to be deleted? ')
        # Open the original file and a new temporary file.
    emp_file = open('employees.txt', 'r')
    temp_file = open('temp.txt', 'w')

    # Read the first line from the file, which is
    name = emp_file.readline()
    # If a field was read, continue processing the rest...
    while name != '':
        # Read the ID number field.
        id_num = emp_file.readline()
        id_num = id_num.rstrip('\n') #Strip '\n'
        # Read the department field.
        dept = emp_file.readline()
        # If this is not the record to delete, then
        # write it to the temporary file.
        if id != id_num:
            # Write the record to the temp file.
            temp_file.write(name)
            temp_file.write(id_num + '\n') # add back '\n'
            temp_file.write(dept)
        else:
            # Set the found flag to True.Don't write this to tmp
            found = True
        # Read the name field of the next record.
        name = emp_file.readline()

    # Close the employees file and the temporary file.
    emp_file.close()
    temp_file.close()
    # Delete the original file then rename temp file
    os.remove('employees.txt')
    os.rename('temp.txt', 'employees.txt')
    if found:    # display a message if found or not
        print('The file has been updated.')
    else:
        print('That item was not found in the file.')
# Call the main function.
main()
```

→ importing `os` module for remove and rename functions.

→ Getting id number to be deleted from the user.

→ opening `employees.txt` in read mode.

→ creating a temporary file named `temp.txt` with 'w' mode.

Reading records from `employees.txt`

If `id_num` of the record is not `id` to be deleted then write it to `temp.txt` otherwise skip – don't write it to `temp.txt`

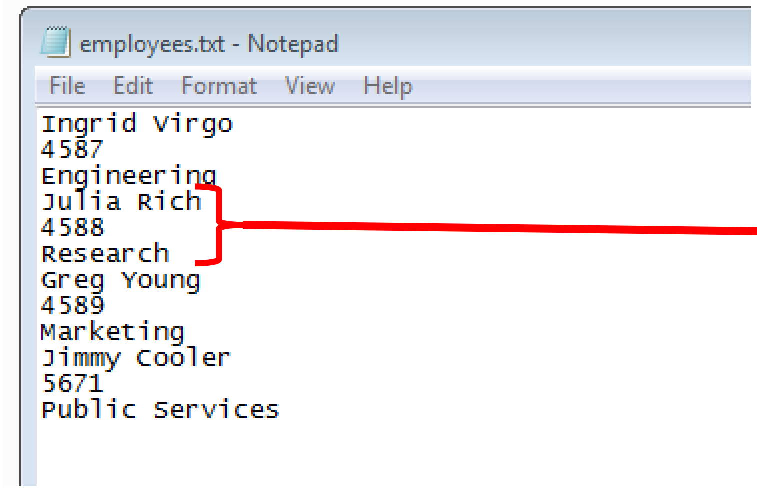
Close the files.
Remove/Delete the `employees.txt`
Rename `temp.txt` as `employees.txt`

Print a message to user by checking `found` variable

Example: Deleting Records in employee.txt

Write a program that deletes a record in the `employees.txt` for a given employee id.

BEFORE: Original employees.txt



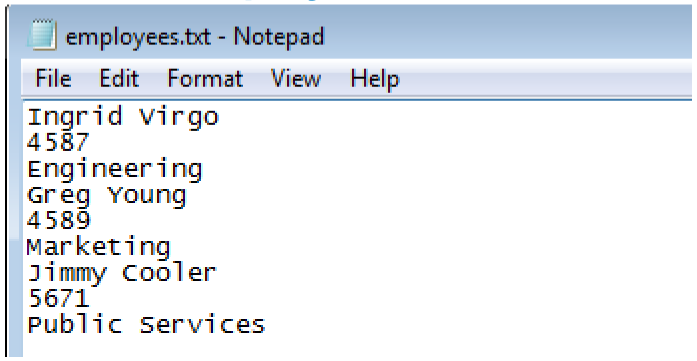
```
Ingrid Virgo
4587
Engineering
Julia Rich
4588
Research
Greg Young
4589
Marketing
Jimmy Cooler
5671
Public Services
```

Example Program Runs:

```
Employee id to be deleted? 1453
That item was not found in the file.

Employee id to be deleted? 4588
The file has been updated.
```

The matching record is now deleted!

AFTER: employees.txt



```
Ingrid Virgo
4587
Engineering
Greg Young
4589
Marketing
Jimmy Cooler
5671
Public Services
```

Checkpoint

6.16 What is a record? What is a field?

set of data that describes one item. one piece of data in a record called field.

6.17 Describe the way that you use a temporary file in a program that modifies a record in a sequential access file.

Temporary files are used for updating a file in the following order:
1.The data is re-written to temporary file, by modifying the data.
2.The original file is removed.
3.The temporary file is renamed to original file's name.

6.18 Describe the way that you use a temporary file in a program that deletes a record from a sequential file.

Temporary files are used for deleting a record in the following order:
1.The data wanted to be kept is re-written to a temporary file by skipping the data to be deleted.
2.The original file is removed.
3.The temporary file is renamed to original file's name.

Summary of Thins about File Operations

- **file=open(filename,mode) :** open a certain file with desired more and assign link to file object/variable.
- **Methods with files: write, read, readline, close**
  - **file.write(var+'\n'):** write string variable along with '\n'
  - **file.read():** Reading whole content
  - **file.readline():** read one line with '\n\
    - **string.rstrip()** is used to remove '\n'
  - **file.close():** Closing the file
- **File Processing with loops**
  - `line = file.readline()` **then** `while` *`line`* `!= ''`
  - `for` *`line`* `in` *`file_object`*`:`
- **Records** (all info) **and Fields** (piece of the record)
- **OS library:** `import os –` changing or removing info from a file, followings methods used .
  - `os.remove('filename')`
  - `os.rename('oldname',newname')`

Exceptions

- **Exception: error that occurs while a program is running**
  - Usually unexpected termination of the program.
  - 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.
  - When a program terminates with an exception, a corresponding traceback appears as a red message in the terminal.

What can be done to avoid 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 or float
    - Trying to open for reading a file that doesn’t exist

Example: Exception and Traceback

Let's consider a program that calculates the ratio of two integers.

```
# This program divides a number by another number.

def main():
    # Get two numbers.
    num1 = int(input('Enter a number: '))
    num2 = int(input('Enter another number: '))

    # Divide num1 by num2 and display the result.
    result = num1 / num2
    print(num1, 'divided by', num2, 'is', result)

# Call the main function.
main()
```

It works perfectly but what if you enter 0 value for the second variable, `num2`?

```
Enter a number: 123
Enter another number: 0
Traceback (most recent call last):
  File "C:\Python37\division.py", line 13, in <module>
    main()
  File "C:\Python37\division.py", line 9, in main
    result = num1 / num2
ZeroDivisionError: division by zero
```

Traceback Message indicates the line number Exception/Error type

Error message

Name of the exception is ZeroDivisionError

Example: Handling ZeroDivisionError by code

Let's try to fix this problem with coding

```python
# This program divides a number by another number.
# Program make sure second number is non-zero.
def main():
    # Get two numbers.
    num1 = int(input('Enter a number: '))
    num2 = int(input('Enter another number: '))

    # If num2 is not zero, divide num1 by num2
    # and display the result.
    if num2 != 0:
        result = num1 / num2
        print(num1, 'divided by', num2, 'is', result)
    else:
        print('Cannot divide by zero.')

# Call the main function.
main()
```

Check `num2`
if it is not zero than do division!

It loos working now

```
Enter a number: 1453
Enter another number: 0
Cannot divide by zero.
```

What if you enter a non-numerical values for any of the variables?

```
Enter a number: 1456
Enter another number: 12z
Traceback (most recent call last):
  File "C:\Python37\division.py", line 18, in <module>
    main()
  File "C:\Python37\division.py", line 7, in main
    num2 = int(input('Enter another number: '))
ValueError: invalid literal for int() with base 10: '12z'
```

Another Traceback Message for line 7

Name of the exception is ValueError

Handling Exceptions

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

If any error raises in within the try suite, the program jumps to handler/except part with the exception name.

Try Suite and Handler (Except)

- **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 – statements under except clause 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

```
try:
        statement
        statement
except ValueError:
        statement
except ZeroDivisionError:
        statement
except:
        statement
```

$\longrightarrow$ **All exceptions other than ValueError and ZeroDivisionError are handled here.**

Example: Try/Except Suite Usage

Let's try to include exception handling by using try/except Suites

```
# This program divides a number by another number.
def main():

    try:
        # Get two numbers.
        num1 = int(input('Enter a number: '))
        num2 = int(input('Enter another number: '))

        # Divide num1 by num2 and display the result.
        result = num1 / num2


        print(num1, 'divided by', num2, 'is', result)

    except ValueError:
        print('ERROR: Invalid input entered!')

    except ZeroDivisionError:
        print('ERROR: Donaminator can not be zero!')

# Call the main function.
main()
```

`main` statements are in a `try` suite

two `except` clauses are listed

Let's test the program and see how it works:

```
Enter a number: 18
Enter another number: ten
ERROR: Invalid input entered!
```

```
Enter a number: 12r
ERROR: Invalid input entered!
```

```
Enter a number: 145
Enter another number: 0
ERROR: Donaminator can not be zero!
```

```
Enter a number: 146
Enter another number: 4
146 divided by 4 is 36.5
```

No traceback messages raises. Program terminates with an error message.

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 ValueError as err:`
  - Can pass *exception object variable as err variable* to `print` function to display the default error message

Let's consider the integer division program:

```
except ValueError as err:
    print(err)

except ZeroDivisionError as err:
    print(err)
```

Example Program Runs:

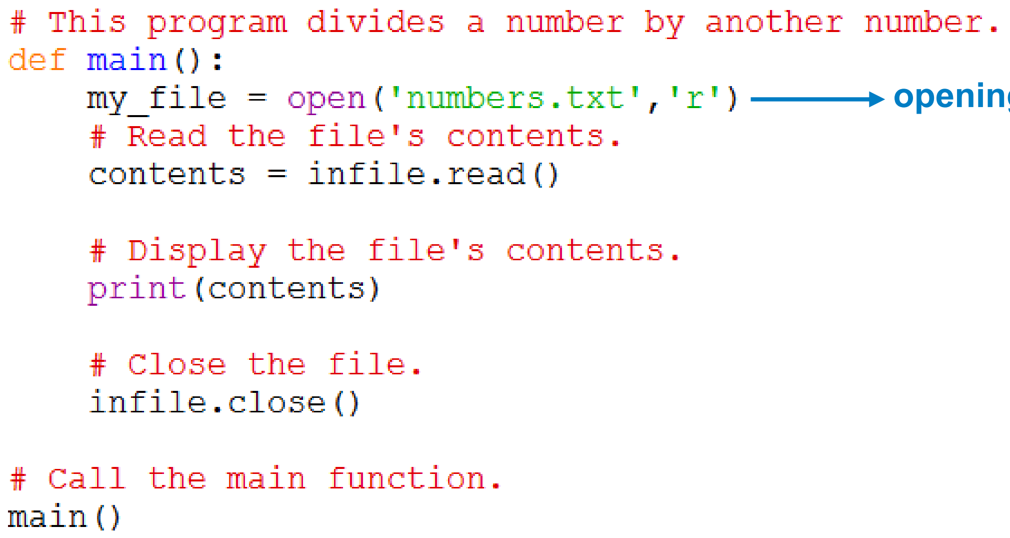
```
Enter a number: 145
Enter another number: 0
division by zero
```

```
Enter a number: 134
Enter another number: five
invalid literal for int() with base 10: 'five'
```

Other Exception Errors

- Different exception errors can raise in different programs.
- visit https://docs.python.org/3/library/exceptions.html

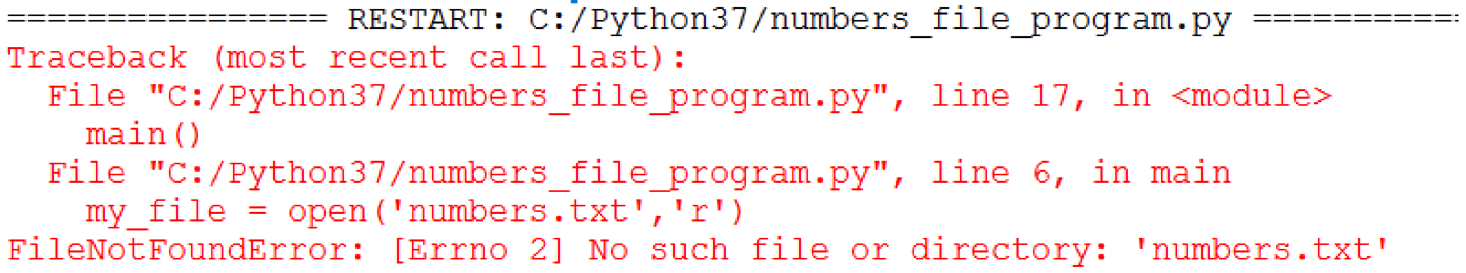
Example: **While opening a file if the file doesn't exist then the interpreter gives an exception too.**



```
# This program divides a number by another number.
def main():
    my_file = open('numbers.txt','r')
    # Read the file's contents.
    contents = infile.read()

    # Display the file's contents.
    print(contents)

    # Close the file.
    infile.close()

# Call the main function.
main()
```

opening `numbers.txt` in 'r' read mode.

FileNotFoundError exception error is raised if file doesn't exist.



```
================= RESTART: C:/Python37/numbers_file_program.py ==========
Traceback (most recent call last):
  File "C:/Python37/numbers_file_program.py", line 17, in <module>
    main()
  File "C:/Python37/numbers_file_program.py", line 6, in main
    my_file = open('numbers.txt','r')
FileNotFoundError: [Errno 2] No such file or directory: 'numbers.txt'
```

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
  - **<u>Else suite</u>:** block of statements executed after statements in try suite, only if no exceptions were raised
    - If exception was raised, the else suite is skipped

```
try:
        statement
        statement
        etc.
except ExceptionName:
        statement
        statement
        etc.
else:
        statement
        statement
        etc.
```

executed if no exception raised

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`
  - <u>Finally suite</u>: block of statements after the `finally` clause
    - Execute whether an exception occurs or not
    - Purpose is to perform cleanup before exiting

```
try:
        statement
        etc.
except ExceptionName:
        statement
        etc.
finally:
        statement  }
        etc.       }  execute whether an exception or not
```

Example: else and finally in Exception handling.

- **The following program reads numbers from numbers.txt and adds them. All expectations are handled with default message.**

```python
# This program displays the total of the
# amounts in the numbers.txt file.

def main():
    total = 0.0 # Initialize an accumulator.
    try:
        # Open the numbers.txt file.
        infile = open('numbers.txt', 'r')

        # Read the values from the file and
        # accumulate them.
        for line in infile:
            amount = float(line)
            total += amount

        # Close the file.
        infile.close()

    except Exception as err:
        print(err)
```

all exceptions are handled here.
Default message is saved in `err` variable.

```python
    else:
        # Print the total.
        print('Total is',format(total, ',.2f'))
```

executed if NO EXCEPTION raised.

```python
    finally:
        print('Program has been terminated!')
```

executed whether if an exception or not. Executed all the time.

```python
# Call the main function.
main()
```

Checkpoint

6.20 If an exception is raised and the program does not handle it with a try/except statement, what happens?

Program halts abruptly with traceback error message.

6.21 What type of exception does a program raise when it tries to open a nonexistent file?

If we try to open a non-existent file in 'r' mode then the program raises `FileNotFoundError` exception.

6.22 What type of exception does a program raise when it uses the float function to convert a non-numeric string to a number?

it raises a `ValueError` exception.

What If an Exception Is Not Handled?

- In this section, you've seen examples of programs that can raise

  `ZeroDivisionError` exceptions

  `IOError` exceptions

  `ValueError` exceptions.

There are many different types of exceptions that can occur visit https://docs.python.org/3/library/exceptions.html

- **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

Review Questions

Page 359-360-361

Multiple Choice

3. Before a file can be used by a program, it must be ______________.
   - a. formatted
   - b. encrypted
   - c. closed
   - [✓ d. opened]

7. When working with this type of file, you access its data from the beginning of the fil
   to the end of the file.
   - a. ordered access
   - b. binary access
   - c. direct access
   - [✓ d. sequential access]

12. This is a single piece of data within a record.
    - [✓ a. field]
    - b. variable
    - c. delimiter
    - d. subrecord

13. When an exception is generated, it is said to have been __________.
    - a. built
    - [✓ b. raised]
    - c. caught
    - d. killed

15. You write this statement to respond to exceptions.
    - a. `run/handle`
    - [✓ b. `try/except`]
    - c. `try/handle`
    - d. `attempt/except`

Algorithm Workbench

6. Write code that opens an output file with the filename `number_list.txt`, but does not erase the file's contents if it already exists.

```
outfile = open( number_list.txt, 'r' )
```

or

```
outfile = open( number_list.txt, 'a' )
```

7. A file exists on the disk named `students.txt`. The file contains several records, and each record contains two fields: (1) the student's name, and (2) the student's score for the final exam. Write code that deletes the record containing "John Perz" as the student name.

```
std_file = open('students.txt', 'r')
 temp_file = open('temp.txt', 'w')
name = std_file.readline()
while name != '':
        score = std_file.readline()
                name = name.rstrip('\n')   #Strip '\n'
        if name != 'John Perz' :  # Don't Write this data to temp.txt
                temp_file.write(name+'\n')
                temp_file.write(score)
        name = std_file.readline()   #read the name in the next record
std_file.close()
temp_file.close()
os.remove('employees.txt')
os.rename('temp.txt', 'employees.txt')
```

Programming Exercises

4. High Score

Assume that a file named `scores.txt` exists on the computer's disk. It contains a series of records, each with two fields – a name, followed by a score (an integer between 1 and 100). Write a program that displays the name and score of the record with the highest score, as well as the number of records in the file.

Hint: *Use a variable and an "if" statement to keep track of the highest score found as you read through the records, and a variable to keep count of the number of records.*

Programming Exercises

4. High Score -PROGRAM-

```
# This program reads name and scores-finds highest score
def main():
    # Declare variables.
    high_score = 0
    high_scorer = ''
    counter = 0
    # Open scores.txt file for reading.
    score_file = open('scores.txt', 'r')

    # Read the first data -name- in the file.
    name = score_file.readline()

    # Read data from the file until no more data.
    while name != '':
        counter += 1 # Increment the counter.
        # Read the score and convert it to an integer.
        score = int(score_file.readline())
        # Check for high score if greater than high score
        if score > high_score:
            high_score = score
            high_scorer = name
        # Read the name of the next record.
        name = score_file.readline()

    score_file.close() # Close file.
    # Display the results.
    print('High Score:', high_score)
    print('Held By:', high_scorer)
    print('Number of Scores:', counter)
# Call the main function.
main()
```

Programming Exercises

5. Sum of Numbers

Assume a file containing a series of integers is named `numbers.txt` and exists on the computer’s disk. Write a program that reads all of the numbers stored in the file and calculates their total.

Programming Exercises

5. Sum of Numbers – PROGRAM-

```
# This program finds the total of the numbers in a file

def main():
    # Declare Accumulator Variable
    total = 0.0

    # Open numbers.txt file for reading
    in_file = open('numbers.txt', 'r')
    #Processing whole file with for repetition
    #One may also use while repetition as well.
    for line in in_file:
        number = float(line)
        total += number

    # Close file
    in_file.close()

    # Display the total of the numbers in the file
    print('Total: ', total)

# Call the main function.
main()
```

Programming Exercises

10. Golf Scores

The Springfork Amateur Golf Club has a tournament every weekend. The club president has asked you to write two programs:

1. A program that will read each player’s name and golf score as keyboard input, then save these as records in a file named `golf.txt`. (Each record will have a field for the player’s name and a field for the player’s score.)

2. A program that reads the records from the `golf.txt` file and displays them.

Programming Exercises

10. Golf Scores –PROGRAM 1 TO WRITE-

```python
# This program writes player names and scores to golf.txt
# Part I

def main():
    # Prompt user for the number of players
    num_players = int(input('Enter the number of ' \
                            'players in the tournament: '))
    # Open golf.txt for writing
    out_file = open('golf.txt', 'w')

    # Write num_players data to the file in for repetition
    for i in range(num_players):
        # Get the name and score from the user
        name = input('Enter the name of the player: ')
        golf_score = int(input('Enter the golf score: '))

        # Write the data to file separated by '\n'
        out_file.write(name + '\n')
        out_file.write(str(golf_score) + '\n')

    # Close file at the end
    out_file.close()

# Call the main function.
main()
```

Programming Exercises

10. Golf Scores –PROGRAM 2 TO READ AND DISPLAY-

```
# This program displays data stored in golf.txt
# Part II
def main():
    # Define counter to count how many data read
    num_players = 0
    # Open golf.txt for reading
    in_file = open('golf.txt', 'r')

    # Read first name
    name = in_file.readline()

    # Read until no data by using while repetition
    while name != '':
        # Read score as integer number - so not stripping needed
        golf_score = int(in_file.readline())
        num_players+=1 #increase number of player counter by 1

        name = name.rstrip('\n') # Strip '\n' from name

        # Display data with one line space between the data
        # for every two players
        print ('\nName:', name)
        print ('Golf Score:', golf_score)

        # Read next name
        name = in_file.readline()
    # Print the number of data read from the file
    print('Number of Players:',num_players)
    # Close file
    in_file.close()

# Call the main function.
main()
```

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