

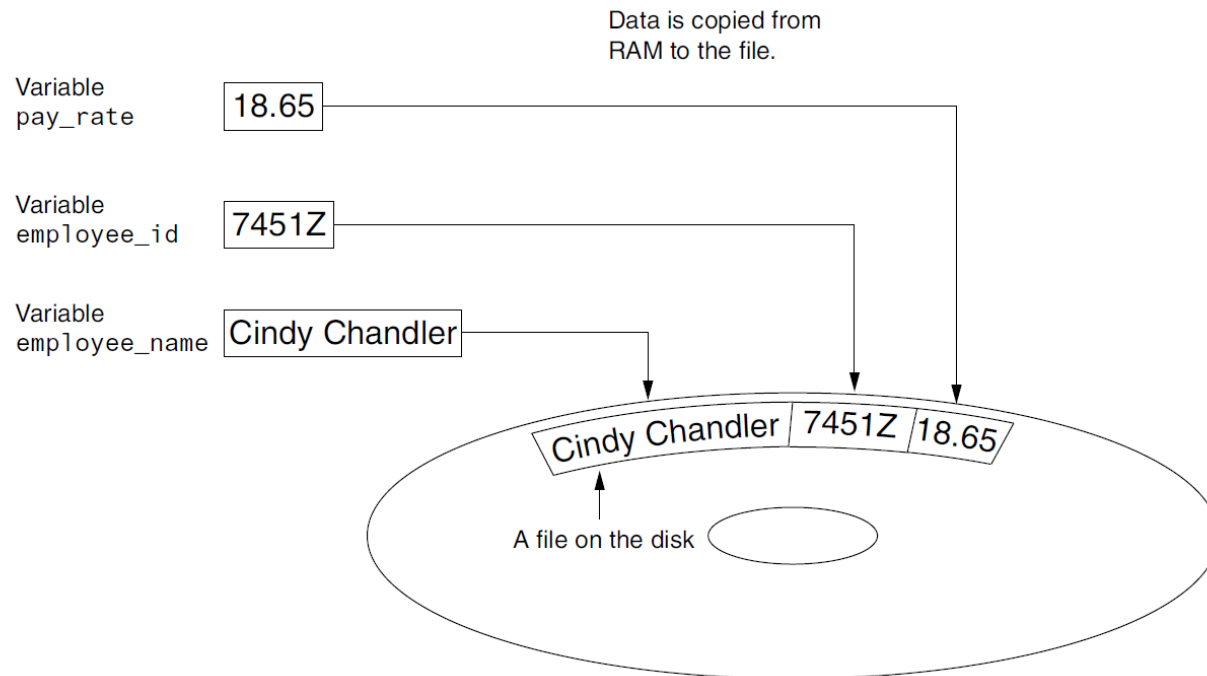
Files and Exceptions

陳建良



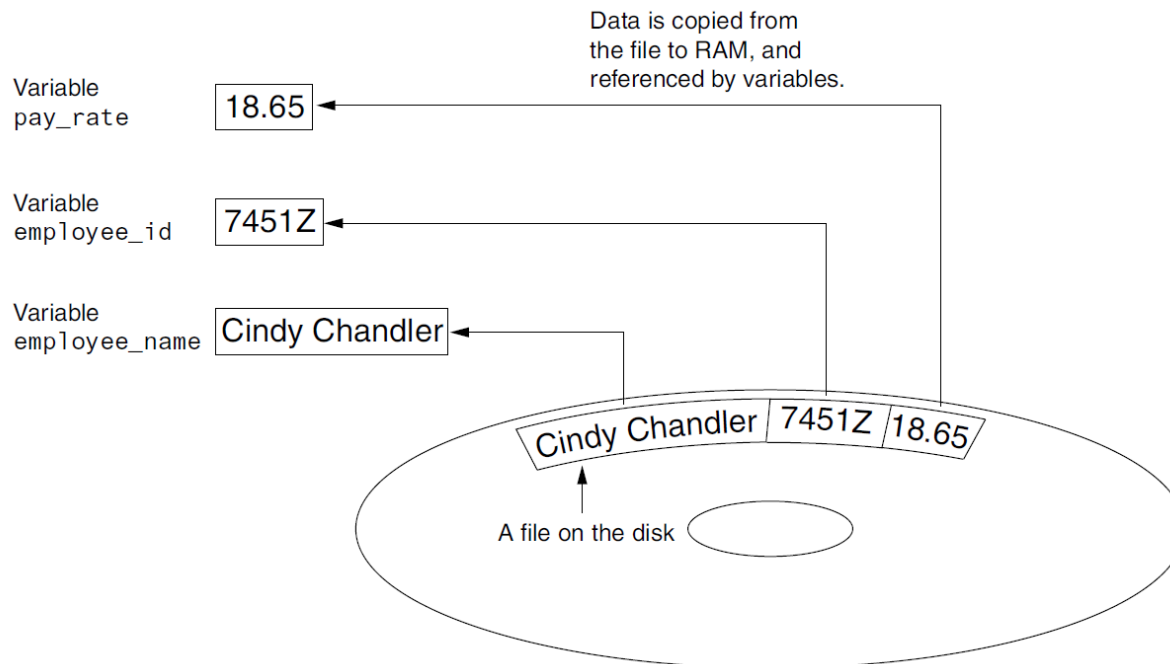
Introduction to File Input and Output

- Programmers usually refer to the process of saving data in a file as “writing data” to the file.
- When a piece of data is written to a file, it is copied from a variable in RAM to the file.



Introduction to File Input and Output

- 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.



Types of Files

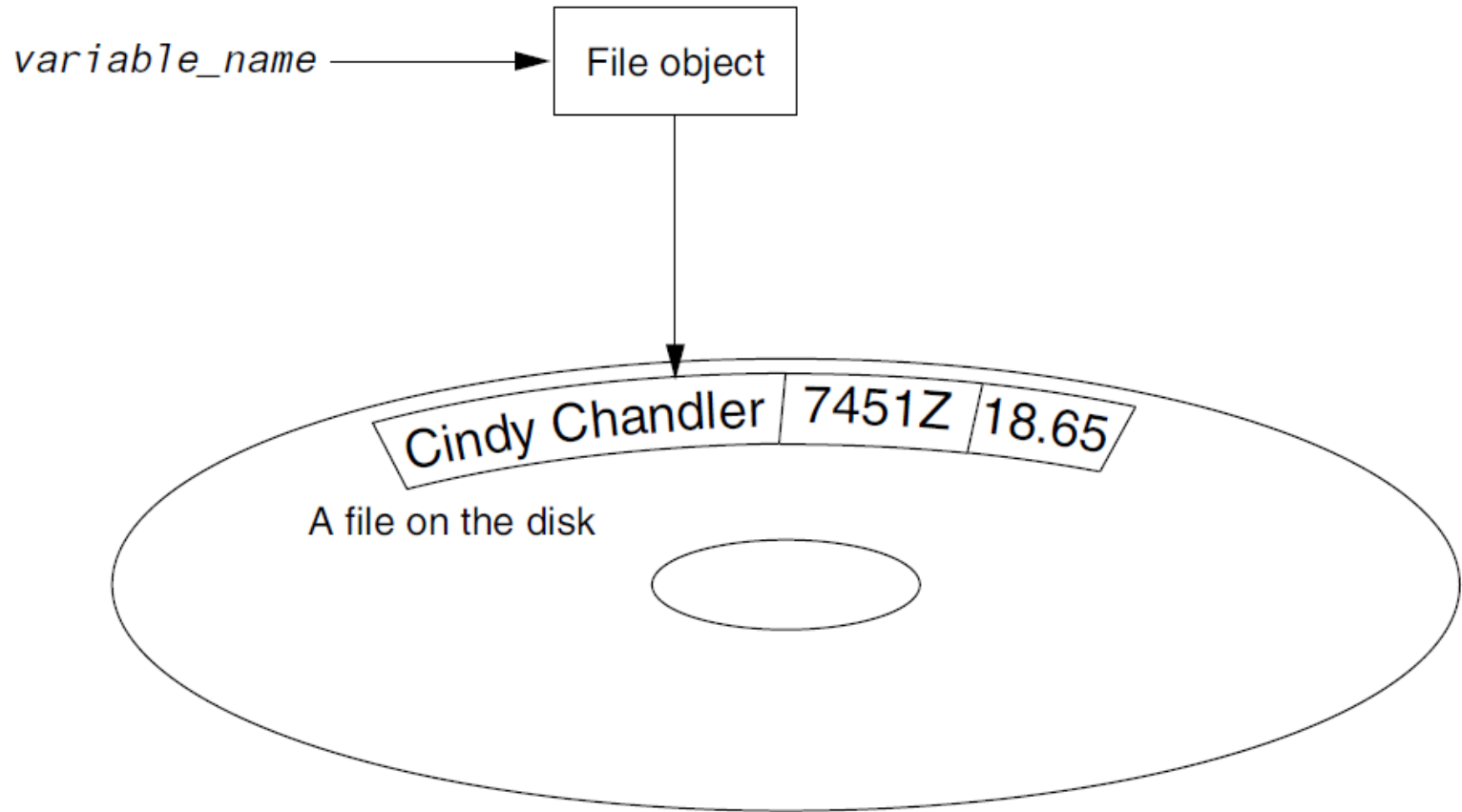
- In general, there are two types of files: text and binary.
- A *text file* contains data that has been encoded as text, using a scheme such as ASCII or Unicode.
- A *binary file* contains data that has not been converted to text. The data that is stored in a binary file is intended only for a program to read.
- Although Python allows you to work both text files and binary files, we will work only with text files.



File Objects

- 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.

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

- *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.



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.

- suppose the file `customers.txt` contains customer data, and we want to open it for reading. Here is an example of how we would call the `open` function:

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

- Suppose we want to create a file named `sales.txt` and write data to it. Here is an example of how we would call the `open` function:

```
sales_file = open('sales.txt', 'w')
```



Specifying the Location of a File

- When you pass a file name that does not contain a path as an argument to the open function, the Python interpreter assumes the file's location is the same as that of the program.
- If you want to open a file in a different location, you can specify a path as well as a filename in the argument that you pass to the open function.

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

- The r prefix specifies that the string is a *raw string*.



Writing Data to A File

- Now, we will introduce you to another type of function, which is known as a method.
- A *method* is a function that belongs to an object and performs some operation using that object.
- Once you have opened a file, you use the file object's methods to perform operations on the file.
- For example, file objects have a method named `write` that can be used to write data to a file.

`file_variable.write(string)`

- `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.



- Once a program is finished working with a file, it should close the file.
- In some systems, failure to close an output file can cause a loss of data.
- When the buffer is full, the system writes the buffer's contents to the file.
- In Python, you use the file object's close method to close a file. For example, the following statement closes the file that is associated with `customer_file`:

```
customer_file.close()
```

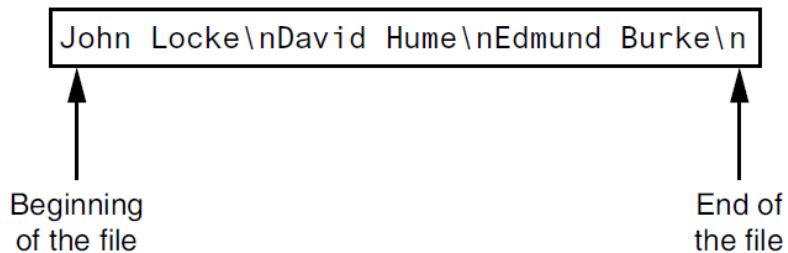


```
# This program writes three lines of data
# to a file.
def main():
    # Open a file named philosophers.txt.
    outfile = open('philosophers.txt', 'w')

    # Write the names of three philosophers
    # to the file.
    outfile.write('John Locke\n')
    outfile.write('David Hume\n')
    outfile.write('Edmund Burke\n')

    # Close the file.
    outfile.close()

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



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.

```
# This program reads and displays the contents  
# of the philosophers.txt file.
```

```
def main():
```

```
    # Open a file named philosophers.txt.  
    infile = open('philosophers.txt', 'r')
```

```
    # Read the file's contents.  
    file_contents = infile.read()  
    # Close the file.  
    infile.close()
```

```
    # Print the data that was read into  
    # memory.  
    print(file_contents)
```

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

The `file_contents` variable references the string that was read from the file

`file_contents` → John Locke\nDavid Hume\nEdmund Burke\n



■ In Python, you can use the readline method to read a line from a file. (A line is simply a string of characters that are terminated with a `\n`.)

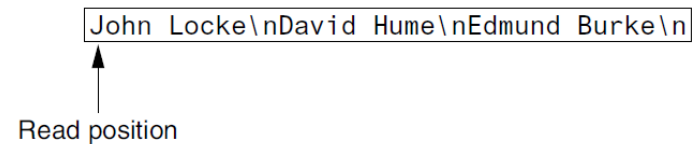
```
# This program reads the contents of the
# philosophers.txt file one line at a time.
def main():
    # Open a file named philosophers.txt.
    infile = open('philosophers.txt', 'r')

    # Read three lines from the file.
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()
    # Close the file.
    infile.close()

    # Print the data that was read into
    # memory.
    print(line1)
    print(line2)
    print(line3)

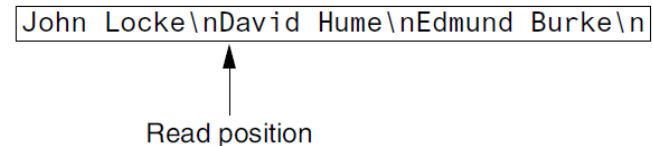
# Call the main function.
main()
```

Initial read position



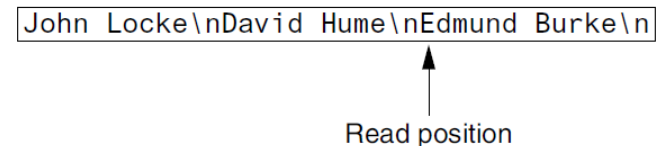
A rectangular box contains the text "John Locke\nDavid Hume\nEdmund Burke\n". An upward-pointing arrow is positioned below the box, pointing to the very beginning of the text. The label "Read position" is placed to the left of the arrow.

Read position advanced to the next line



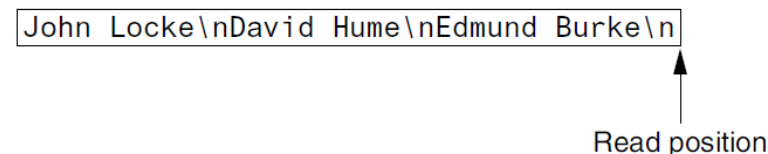
A rectangular box contains the text "John Locke\nDavid Hume\nEdmund Burke\n". An upward-pointing arrow is positioned below the box, pointing to the start of the second line, "David Hume\n". The label "Read position" is placed to the left of the arrow.

Read position advanced to the next line



A rectangular box contains the text "John Locke\nDavid Hume\nEdmund Burke\n". An upward-pointing arrow is positioned below the box, pointing to the start of the third line, "Edmund Burke\n". The label "Read position" is placed to the left of the arrow.

Read position advanced to the end of the file



A rectangular box contains the text "John Locke\nDavid Hume\nEdmund Burke\n". An upward-pointing arrow is positioned below the box, pointing to the end of the text, after the final newline character. The label "Read position" is placed to the left of the arrow.

Concatenating a Newline to a String

```
def main():
    # Get three names.
    print('Enter the names of three friends.')
    name1 = input('Friend #1: ')
    name2 = input('Friend #2: ')
    name3 = input('Friend #3: ')

    # Open a file named friends.txt.
    myfile = open('friends.txt', 'w')

    # Write the names to the file.
    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 
Friend #2: Rose 
Friend #3: Geri 
The names were written to friends.txt.
```

The friends.txt file

```
Joe\nRose\nGeri\n
```



Reading a String and Stripping the Newline from it

- The `\n` serves a necessary purpose inside a file: it separates the items that are stored in the file.
- You want to remove the `\n` from a string after it is read from a file.
- Each string in Python has a method named `rstrip` that removes, or “strips,” specific characters from the end of a string.

```
name = 'Joanne Manchester\n'
```

```
name = name.rstrip('\n')
```




```
def main():
    # Open a file named philosophers.txt.
    infile = open('philosophers.txt', 'r')

    # Read three lines from the file.
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()

    # Strip the \n from each string.
    line1 = line1.rstrip('\n')
    line2 = line2.rstrip('\n')
    line3 = line3.rstrip('\n')

    # Close the file.
    infile.close()

    # Print the data that was read into
    # memory.
    print(line1)
    print(line2)
    print(line3)

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



Appending Data to an Existing File

- When you use the 'w' mode to open an output file and a file with the specified filename already exists on the disk, the existing file will be deleted and a new empty file with the same name will be created.
- Sometimes you want to preserve an existing file and append new data to its current contents. Appending data to a file means writing new data to the end of the data that already exists in the file.
- In Python, you can use the 'a' mode to open an output file in *append mode*.



- 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()
```



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.



```
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 (with input shown in bold)

```
Enter a number: 22 
Enter another number: 14 
Enter another number: -99 
Data written to numbers.txt
```

Contents of the numbers .txt file

```
22\n14\n-99\n
```

- When you read numbers from a text file, they are always read as strings. For example, suppose a program uses the following code to read the first line from the numbers.txt file.

```
infile = open('numbers.txt', 'r')
```

```
value = infile.readline()
```

```
infile.close()
```

- Reading a string from a file with the readline method then converting that string to an integer with the int function.

```
value = int(infile.readline())
```

```
def main():
    # Open a file for reading.
    infile = open('numbers.txt', 'r')

    # Read three numbers from the file.
    num1 = int(infile.readline())
    num2 = int(infile.readline())
    num3 = int(infile.readline())

    # Close the file.
    infile.close()

    # Add the three numbers.
    total = num1 + num2 + num3

    # Display the numbers and their total.
    print('The numbers are:', num1, num2, num3)
    print('Their total is:', total)

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

Program Output

The numbers are: 22 14 -99
Their total is: -63



Using Loops to Process Files

- Files usually hold large amounts of data, and programs typically use a loop to process the data in a file.

```
def main():
    # Get the number of days.
    num_days = int(input('For how many days do ' +
    # Open a new file named sales.txt.
    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 
Data written to sales.txt.
```



Reading a File with a Loop and Detecting the End of the File

- In Python, the readline method returns an empty string ("") when it has attempted to read beyond the end of a file.
- This makes it possible to write a while loop that determines when the end of a file has been reached.

Open the file

Use readline to read the first line from the file

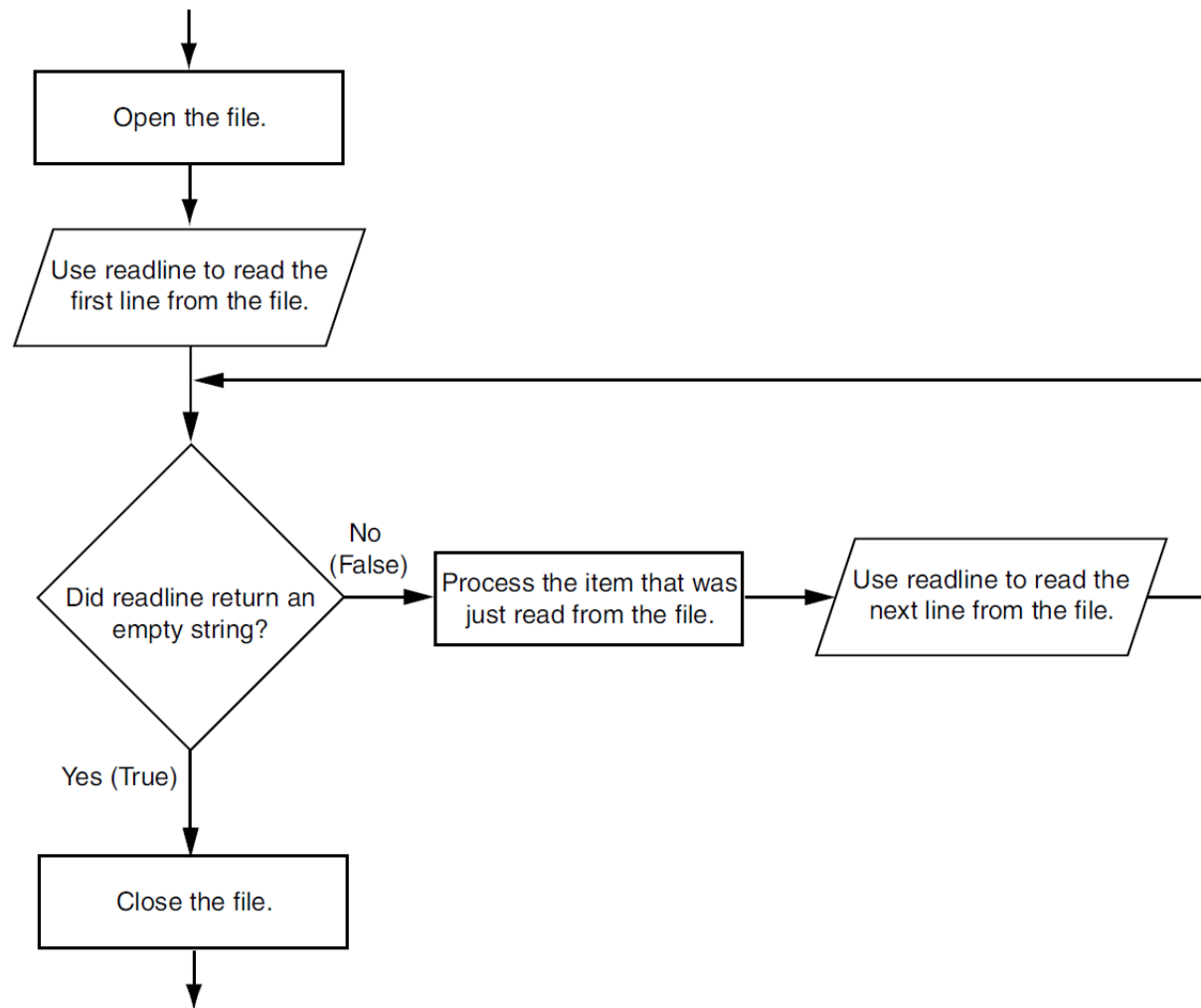
While the value returned from readline is not an empty string:

Process the item that was just read from the file

Use readline to read the next line from the file.

Close the file





```
def main():
    # Open the sales.txt file for reading.
    sales_file = open('sales.txt', 'r')

    # Read the first line from the file, but
    # don't convert to a number yet. We still
    # need to test for an empty string.
    line = sales_file.readline()

    # As long as an empty string is not returned
    # from readline, continue processing.
    while line != '':
        # Convert line to a float.
        amount = float(line)

        # Format and display the amount.
        print(format(amount, '.2f'))

        # Read the next line.
        line = sales_file.readline()

    # Close the file.
    sales_file.close()

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

Program Output

```
1000.00
2000.00
3000.00
4000.00
5000.00
```



Using Python's **for** Loop to Read Lines

- The Python language also allows you to write a for loop that automatically reads the lines in a file without testing for any special condition that signals the end of the file.
- Here is the general format of the loop:

for variable in file_object:

statement

statement

etc.



```
def main():
    # Open the sales.txt file for reading.
    sales_file = open('sales.txt', 'r')

    # Read all the lines from the file.
    for line in sales_file:
        # Convert line to a float.
        amount = float(line)
        # Format and display the amount.
        print(format(amount, '.2f'))

    # Close the file.
    sales_file.close()

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

Program Output

```
1000.00
2000.00
3000.00
4000.00
5000.00
```



■ Kevin is a freelance video producer who makes TV commercials for local businesses. When he makes a commercial, he usually films several short videos. Later, he puts these short videos together to make the final commercial. He has asked you to write the following two programs.

1. A program that allows him to enter the running time (in seconds) of each short video in a project. The running times are saved to a file.
2. A program that reads the contents of the file, displays the running times, and then displays the total running time of all the segments.

■ Here is the general algorithm for the first program, in pseudocode:

Get the number of videos in the project.

Open an output file.

For each video in the project:

Get the video's running time.

Write the running time to the file.

Close the file.



```
def main():
    # Get the number of videos in the project.
    num_videos = int(input('How many videos are in the project? '))

    # Open the file to hold the running times.
    video_file = open('video_times.txt', 'w')

    # Get each video's running time and write
    # it to the file.
    print('Enter the running times for each video.')
    for count in range(1, num_videos + 1):
        run_time = float(input('Video #' + str(count) + ': '))
        video_file.write(str(run_time) + '\n')

    # Close the file.
    video_file.close()
    print('The times have been saved to video_times.txt.')

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

Program Output (with input shown in bold)

```
How many videos are in the project? 6 
Enter the running times for each video.
Video #1: 24.5 
Video #2: 12.2 
Video #3: 14.6 
Video #4: 20.4 
Video #5: 22.5 
Video #6: 19.3 
The times have been saved to video_times.txt.
```



■ Here is the general algorithm for the second program:

Initialize an accumulator to 0.

Initialize a count variable to 0.

Open the input file.

For each line in the file:

Convert the line to a floating-point number. (This is the running time for a video.)

Add one to the count variable. (This keeps count of the number of videos.)

Display the running time for this video.

Add the running time to the accumulator.

Close the file.

Display the contents of the accumulator as the total running time.


```
def main():
    # Open the video_times.txt file for reading.
    video_file = open('video_times.txt', 'r')

    # Initialize an accumulator to 0.0.
    total = 0.0

    # Initialize a variable to keep count of the videos.
    count = 0

    print('Here are the running times for each video:')

    # Get the values from the file and total them.
    for line in video_file:
        # Convert a line to a float.
        run_time = float(line)

        # Add 1 to the count variable.
        count += 1

        # Display the time.
        print('Video #', count, ': ', run_time, sep='')

        # Add the time to total.
        total += run_time

    # Close the file.
    video_file.close()

    # Display the total of the running times.
    print('The total running time is', total, 'seconds.')

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

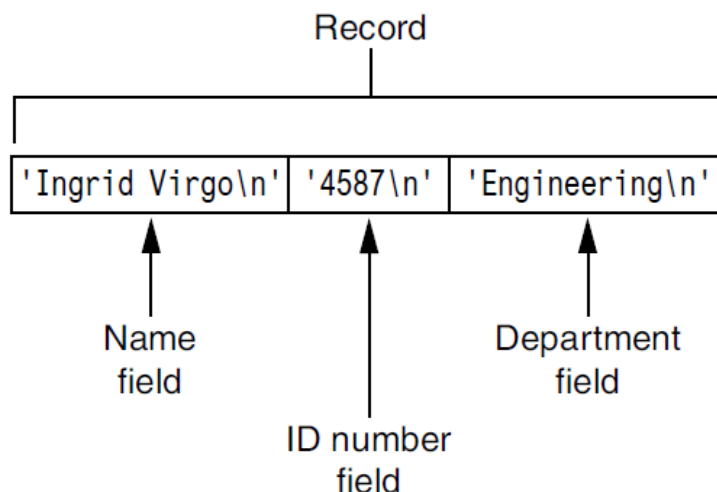
Program Output

```
Here are the running times for each video:
Video #1: 24.5
Video #2: 12.2
Video #3: 14.6
Video #4: 20.4
Video #5: 22.5
Video #6: 19.3
The total running time is 113.5 seconds.
```



Processing Records

- When data is written to a file, it is often organized into records and fields.
- A **record** is a complete set of data that describes one item, and a **field** is a single 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.

```
def main():
    # Get the number of employee records to create.
    num_emps = int(input('How many employee records ' +
                        'do you want to create? '))

    # 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('Enter 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')

        # Display a blank line.
        print()

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

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

Program Output (with input shown in bold)

```
How many employee records do you want to create? 3 
Enter the data for employee #1
Name: Ingrid Virgo 
ID number: 4587 
Department: Engineering 
Enter the data for employee #2
Name: Julia Rich 
ID number: 4588 
Department: Research 
Enter the data for employee #3
Name: Greg Young 
ID number: 4589 
Department: Marketing 
Employee records written to employees.txt.
```





Record			Record			Record		
'Ingrid Virgo\n'	'4587\n'	'Engineering\n'	'Julia Rich\n'	'4588\n'	'Research\n'	'Greg Young\n'	'4589\n'	'Marketing\n'



```
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()
```

Program Output

Name: Ingrid Virgo

ID: 4587

Dept: Engineering

Name: Julia Rich

ID: 4588

Dept: Research

Name: Greg Young

ID: 4589

Dept: Marketing



Aletheia University
資訊工程學系

- Midnight Coffee Roasters, Inc. is a small company that imports raw coffee beans from around the world and roasts them to create a variety of gourmet coffees. Julie, the owner of the company, has asked you to write a series of programs that she can use to manage her inventory. After speaking with her, you have determined that a file is needed to keep inventory records. Each record should have two fields to hold the following data:
 - Description. A string containing the name of the coffee
 - Quantity in inventory. The number of pounds in inventory, as a floating-point number
- Your first job is to write a program that can be used to add records to the file. Note the output file is opened in append mode. Each time the program is executed, the new records will be added to the file's existing contents.



```
def main():
    # Create a variable to control the loop.
    another = 'y'

    # Open the coffee.txt file in append mode.
    coffee_file = open('coffee.txt', 'a')

    # Add records to the file.
    while another == 'y' or another == 'Y':
        # Get the coffee record data.
        print('Enter the following coffee data:')
        descr = input('Description: ')
        qty = int(input('Quantity (in pounds): '))

        # Append the data to the file.
        coffee_file.write(descr + '\n')
        coffee_file.write(str(qty) + '\n')

        # Determine whether the user wants to add
        # another record to the file.
        print('Do you want to add another record?')
        another = input('Y = yes, anything else = no: ')

    # Close the file.
    coffee_file.close()
    print('Data appended to coffee.txt.')

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

Program Output (with input shown in bold)

```
Enter the following coffee data:
Description: Brazilian Dark Roast 
Quantity (in pounds): 18 
Do you want to enter another record?
Y = yes, anything else = no: y 
Description: Sumatra Medium Roast 
Quantity (in pounds): 25 
Do you want to enter another record?
Y = yes, anything else = no: n 
Data appended to coffee.txt.
```



■ Your next job is to write a program that displays all of the records in the inventory file.

```
def main():
    # Open the coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Read the first record's description field.
    descr = coffee_file.readline()

    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

        # Display the record.
        print('Description:', descr)
        print('Quantity:', qty)

        # Read the next description.
        descr = coffee_file.readline()

    # Close the file.
    coffee_file.close()

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

Program Output

```
Description: Brazilian Dark Roast
Quantity: 18.0
Description: Sumatra Medium Roast
Quantity: 25.0
```



Searching For a Record

- Julie has been using the first two programs that you wrote for her. She now has several records stored in the `coffee.txt` file and has asked you to write another program that she can use to search for records. She wants to be able to enter a description and see a list of all the records matching that description.



```

def main():
    # Create a bool variable to use as a flag.
    found = False

    # Get the search value.
    search = input('Enter a description to search for: ')

    # Open the coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Read the first record's description field.
    descr = coffee_file.readline()

    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

        # Determine whether this record matches
        # the search value.
        if descr == search:
            # Display the record.
            print('Description:', descr)
            print('Quantity:', qty)
            print()
            # Set the found flag to True.
            found = True

        # Read the next description.
        descr = coffee_file.readline()

    # Close the file.
    coffee_file.close()

    # If the search value was not found in the file
    # display a message.
    if not found:
        print('That item was not found in the file.')

```

Program Output (with input shown in bold)

```

Enter a description to search for: Sumatra Medium Roast 
Description: Sumatra Medium Roast
Quantity: 25.0

```

Program Output (with input shown in bold)

```

Enter a description to search for: Mexican Altura 
That item was not found in the file.

```



Aletheia University
資訊工程學系

Modifying Records

- Julie is very happy with the programs that you have written so far. Your next job is to write a program that she can use to modify the quantity field in an existing record. This will allow her to keep the records up to date as coffee is sold or more coffee of an existing type is added to inventory.
- To modify a record in a sequential file, you must create a second temporary file. You copy all of the original file's records to the temporary file, but when you get to the record that is to be modified, you do not write its old contents to the temporary file. Instead, you write its new modified values to the temporary file. Then, you finish copying any remaining records from the original file to the temporary file.



- The temporary file then takes the place of the original file. You delete the original file and rename the temporary file, giving it the name that the original file had on the computer's disk. Here is the general algorithm for your program.

Open the original file for input and create a temporary file for output.

Get the description of the record to be modified and the new value for the quantity.

Read the first description field from the original file.

While the description field is not empty:

Read the quantity field.

If this record's description field matches the description entered:

Write the new data to the temporary file.

Else:

Write the existing record to the temporary file.

Read the next description field.

Close the original file and the temporary file.

Delete the original file.

Rename the temporary file, giving it the name of the original file.



■ Notice at the end of the algorithm you delete the original file then rename the temporary file. The Python standard library's `os` module provides a function named `remove`, that deletes a file on the disk. You simply pass the name of the file as an argument to the function.

■ Here is an example of how you would delete a file named `coffee.txt`:

```
remove('coffee.txt')
```

■ The `os` module also provides a function named `rename`, that renames a file. Here is an example of how you would use it to rename the file `temp.txt` to `coffee.txt`:

```
rename('temp.txt', 'coffee.txt')
```

```

import os # Needed for the remove and rename functions

def main():
    # Create a bool variable to use as a flag.
    found = False

    # Get the search value and the new quantity.
    search = input('Enter a description to search for: ')
    new_qty = int(input('Enter the new quantity: '))

    # Open the original coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Open the temporary file.
    temp_file = open('temp.txt', 'w')

    # Read the first record's description field.
    descr = coffee_file.readline()

    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

        # Write either this record to the temporary file,
        # or the new record if this is the one that is
        # to be modified.

        if descr == search:
            # Write the modified record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(new_qty) + '\n')

            # Set the found flag to True.
            found = True
        else:
            # Write the original record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(qty) + '\n')

    # Read the next description.
    descr = coffee_file.readline()

    # Close the coffee file and the temporary file.
    coffee_file.close()
    temp_file.close()

    # Delete the original coffee.txt file.
    os.remove('coffee.txt')

    # Rename the temporary file.
    os.rename('temp.txt', 'coffee.txt')

    # If the search value was not found in the file
    # display a message.
    if found:
        print('The file has been updated.')
    else:
        print('That item was not found in the file.')

```

Program Output (with input shown in bold)

```

Enter a description to search for: Brazilian Dark Roast Enter
Enter the new quantity: 10 Enter
The file has been updated.

```



Deleting Records

- Your last task is to write a program that Julie can use to delete records from the coffee.txt file. Like the process of modifying a record, the process of deleting a record from a sequential access file requires that you create a second temporary file. You copy all of the original file's records to the temporary file, except for the record that is to be deleted. The temporary file then takes the place of the original file. You delete the original file and rename the temporary file, giving it the name that the original file had on the computer's disk.



Open the original file for input and create a temporary file for output.

Get the description of the record to be deleted.

Read the description field of the first record in the original file.

While the description is not empty:

Read the quantity field.

If this record's description field does not match the description entered:

Write the record to the temporary file.

Read the next description field.

Close the original file and the temporary file.

Delete the original file.

Rename the temporary file, giving it the name of the original 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 coffee to delete.
    search = input('Which coffee do you want to delete? ')
```

```
    # Open the original coffee.txt file.
    coffee_file = open('coffee.txt', 'r')
```

```
    # Open the temporary file.
    temp_file = open('temp.txt', 'w')
```

```
    # Read the first record's description field.
    descr = coffee_file.readline()
```

```
    # Read the rest of the file.
```

```
    while descr != '':
```

```
        # Read the quantity field.
        qty = float(coffee_file.readline())
```

```
        # Strip the \n from the description.
        descr = descr.rstrip('\n')
```

```
        # If this is not the record to delete, then
        # write it to the temporary file.
        if descr != search:
```

```
            # Write the record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(qty) + '\n')
```

```
        else:
            # Set the found flag to True.
            found = True
```

```
        # Read the next description.
        descr = coffee_file.readline()
```

```
    # Close the coffee file and the temporary file.
    coffee_file.close()
    temp_file.close()
```

```
    # Delete the original coffee.txt file.
    os.remove('coffee.txt')
```

```
    # Rename the temporary file.
    os.rename('temp.txt', 'coffee.txt')
```

```
    # If the search value was not found in the file
    # display a message.
    if found:
```

```
        print('The file has been updated.')
```

```
    else:
        print('That item was not found in the file.')
```

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

Program Output (with input shown in bold)

```
Which coffee do you want to delete? Brazilian Dark Roast Enter
The file has been updated.
```



Aletheia University
資訊工程學系

Exceptions

- An exception is an error that occurs while a program is running, causing the program to abruptly halt.
- You can use the try/except statement to gracefully handle exceptions.



```
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()
```

Program Output (with input shown in bold)

Enter a number: **10**

Enter another number: **0**

Traceback (most recent call last):

File "C:\Python\division.py," line 13, in <module>
 main()

File "C:\Python\division.py," line 9, in main
 result = num1 / num2

ZeroDivisionError: integer division or modulo by zero



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

    # If num2 is not 0, 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()
```

Program Output (with input shown in bold)

```
Enter a number: 10 
Enter another number: 0 
Cannot divide by zero.
```



```
def main():
    # Get the number of hours worked.
    hours = int(input('How many hours did you work? '))

    # Get the hourly pay rate.
    pay_rate = float(input('Enter your hourly pay rate: '))

    # Calculate the gross pay.
    gross_pay = hours * pay_rate

    # Display the gross pay.
    print('Gross pay: $', format(gross_pay, ',.2f'), sep='')

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

Program Output (with input shown in bold)

```
How many hours did you work? forty 
Traceback (most recent call last):
  File "C:\Users\Tony\Documents\Python\Source
Code\Chapter 06\gross_pay1.py", line 17, in <module>
    main()
  File "C:\Users\Tony\Documents\Python\Source
Code\Chapter 06\gross_pay1.py", line 5, in main
    hours = int(input('How many hours did you work? '))
ValueError: invalid literal for int() with base 10: 'forty'
```



- Python, like most modern programming languages, allows you to write code that responds to exceptions when they are raised, and prevents the program from abruptly crashing.
- Such code is called an **exception handler** and is written with the try/except statement.
- There are several ways to write a try/except statement, but the following general format shows the simplest variation:

try:

statement

statement

etc.

except *ExceptionName*:

statement

statement

etc.



```
def main():
    try:
        # Get the number of hours worked.
        hours = int(input('How many hours did you work? '))

        # Get the hourly pay rate.
        pay_rate = float(input('Enter your hourly pay rate: '))

        # Calculate the gross pay.
        gross_pay = hours * pay_rate

        # Display the gross pay.
        print('Gross pay: $', format(gross_pay, ',.2f'), sep='')
    except ValueError:
        print('ERROR: Hours worked and hourly pay rate must')
        print('be valid numbers.')
```

Program Output (with input shown in bold)

How many hours did you work? **forty**
ERROR: Hours worked and hourly pay rate must
be valid numbers.



```
# This program calculates gross pay.

def main():
    try:
        # Get the number of hours worked.
        hours = int(input('How many hours did you work? '))

        # Get the hourly pay rate.
        pay_rate = float(input('Enter your hourly pay rate: '))

        # Calculate the gross pay.
        gross_pay = hours * pay_rate

        # Display the gross pay.
        print('Gross pay: $', format(gross_pay, ',.2f'), sep='')
    except ValueError:
        print('ERROR: Hours worked and hourly pay rate must')
        print('be valid integers.')

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

If this statement raises a `ValueError` exception...

The program jumps to the `except ValueError` clause and executes its handler.




```
def main():
    # Get the name of a file.
    filename = input('Enter a filename: ')

    # Open the file.
    infile = open(filename, '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()
```

Program Output (with input shown in bold)

```
Enter a filename: bad_file.txt 
Traceback (most recent call last):
File "C:\Python\display_file.py," line 21, in <module>
main()
File "C:\Python\display_file.py," line 9, in main
infile = open(filename, 'r')
IOError: [Errno 2] No such file or directory: 'bad_file.txt'
```



```
def main():
    # Get the name of a file.
    filename = input('Enter a filename: ')
    try:
        # Open the file.
        infile = open(filename, 'r')

        # Read the file's contents.
        contents = infile.read()

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

        # Close the file.
        infile.close()
    except IOError:
        print('An error occurred trying to read')
        print('the file', filename)

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

Program Output (with input shown in bold)

```
Enter a filename: bad_file.txt Enter
An error occurred trying to read the file bad_file.txt
```



Handling Multiple Exceptions

- In many cases, the code in a try suite will be capable of throwing more than one type of exception.
- In such a case, you need to write an except clause for each type of exception that you want to handle.



```
def main():
    # Initialize an accumulator.
    total = 0.0

    try:
        # Open the sales_data.txt file.
        infile = open('sales_data.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()

        # Print the total.
        print(format(total, ',.2f'))

    except IOError:
        print('An error occurred trying to read the file.')

    except ValueError:
        print('Non-numeric data found in the file.')

    except:
        print('An error occurred.')

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



Using One **except** Clause to Catch All Exceptions

- The previous example demonstrated how multiple types of exceptions can be handled individually in a try/except statement.
- Sometimes you might want to write a try/except statement that simply catches any exception that is raised in the try suite and, regardless of the exception's type, responds the same way.



```
def main():
    # Initialize an accumulator.
    total = 0.0

    try:
        # Open the sales_data.txt file.
        infile = open('sales_data.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()

        # Print the total.
        print(format(total, ',.2f'))

    except:
        print('An error occurred.')

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



Displaying an Exception's Default Error Message

- When an exception is thrown, an object known as an **exception object** is created in memory.
- The exception object usually contains a default error message pertaining to the exception.
- When you write an except clause, you can optionally assign the exception object to a variable, as shown here:

 except ValueError as err:
- This except clause catches ValueError exceptions. The expression that appears after the except clause specifies that we are assigning the exception object to the variable err.

```
def main():
    try:
        # Get the number of hours worked.
        hours = int(input('How many hours did you work? '))

        # Get the hourly pay rate.
        pay_rate = float(input('Enter your hourly pay rate: '))

        # Calculate the gross pay.
        gross_pay = hours * pay_rate

        # Display the gross pay.
        print('Gross pay: $', format(gross_pay, ',.2f'), sep='')
    except ValueError as err:
        print(err)
```

Program Output (with input shown in bold)

```
How many hours did you work? forty 
invalid literal for int() with base 10: 'forty'
```



- If you want to have just one except clause to catch all the exceptions that are raised in a try suite, you can specify **Exception** as the type.

```
def main():
    # Initialize an accumulator.
    total = 0.0

    try:
        # Open the sales_data.txt file.
        infile = open('sales_data.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()

        # Print the total.
        print(format(total, ',.2f'))
    except Exception as err:
        print(err)

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



The **else** Clause

- The try/except statement may have an optional else clause, which appears after all the except clauses.
- The statements in the else suite are executed after the statements in the try suite, only if no exceptions were raised.

try:

statement

statement

etc.

except *ExceptionName*:

statement

statement

etc.

else:

statement

statement

etc.



```
def main():
    # Initialize an accumulator.
    total = 0.0

    try:
        # Open the sales_data.txt file.
        infile = open('sales_data.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)
    else:
        # Print the total.
        print(format(total, ',.2f'))
```



The **finally** Clause

- The try/except statement may have an optional finally clause, which must appear after all the except clauses.
- The statements in the finally suite are always executed after the try suite has executed, and after any exception handlers have executed.

try:

statement

statement

etc.

except *ExceptionName*:

statement

statement

etc.

finally:

statement

statement

etc.



```
num_steps = int(input("總共輸入幾筆資料:"))
emp_file = open('employees.txt', 'w')
for count in range(1, num_steps+1):
    print('這是第', count, '筆資料')
    name = input('輸入名字:')
    id_num = input('ID編號:')
    dep = input('部門:')

    emp_file.write(name + '\n')
    emp_file.write(id_num + '\n')
    emp_file.write(dep + '\n')

    print()

emp_file.close()
print('職員資料寫入到檔案中...')
```

