Finishing the Employee Class Program

by Sophia

WHAT'S COVERED

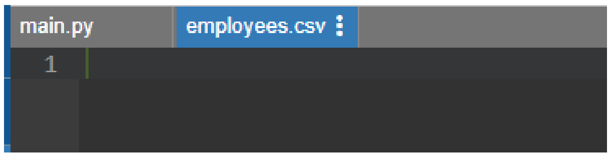
In this lesson, we'll be looking at creating a program that makes use of files and modules. Specifically, this lesson covers:

1. Storing Employee Information to a File

Giving the employee class program some final touches!

For this lesson, we are going to create a program that stores the employee ID and salary for our employees into a file. Rather than having it entered every single time, we will write that information to a file so that we can easily recall that information. We could extend this program to include additional details of the employee but instead, we will focus on the key elements (namely employee ID and salary). Let's first start by creating an `employees.csv` file in the IDE by clicking on the "New File" icon in the top menu.

We will name it `employees.csv`.



TRY IT

Directions: Go ahead and create a new file called `employees.csv`.

DID YOU KNOW

This new file is a file that has an extension of .csv. What is that? Comma-separated value (or .csv) is a file format where each of the data elements for a row of data is split (divided) by a comma. This is very common to spreadsheet applications like Microsoft Excel. In a .csv file we would see data that is saved in a format like this example:

Doe,John,November,15,1980,Johnson,Mary,June,23,1975

This would be a file that contains first name, last name, birthday month, birthday day, and birthday year. As more information (individuals) are added, the data separation would continue.

We will take a look back at the `employees.csv` file later to see what it looks like as data is added to it. For now, the file should be completely empty.

As this is meant to be a larger program, we'll first start by defining some modules that we will import. In particular, we'll `import` the `csv` and `sys` modules. The `csv` module will allow us to work with .csv files without having to figure out how to split up the data elements. Rather, the commas will automatically be added when we write the values and input data will be automatically split up when we read the values. The `sys` module will allow us to use the `sys.exit()` function to exit the program, much like we did in the prior lesson.

We will also define our `employees.csv` file name early as the variable `FILENAME` so that we can make use of it throughout our program.

BIG IDEA

Note: We are using all caps because we will not change this variable. Python does not support constant values as other programming languages do. This is just a visual indicator that this variable is not meant to be changed. We will set the string `employees.csv` to the `FILENAME` variable so that if we do need to change the filename later, it can be done in one place. This is another case of creating code to be as manageable as possible. If we had used the actual filename everywhere in the program and then the filename needed to be changed, we would need to identify all calls to that filename to make the change. Setting it to a global variable allows us to make a change quickly (once) if needed.

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"
```

TRY IT

Directions: On your `main.py` file, go ahead and start adding the code above.

Next, we will create a function called `read_employees()`. The purpose of this function is to read in from the `employees.csv` file (now our variable `FILENAME`) and use the `.append()` method to append each line into a list and then return the list of employees.

EXAMPLE

```
#read the employees from the file
```

```
def read_employees():
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
```

As part of the code statement of the `read_employee()` function, we create an empty list called `employees`. Next, we use the `with` statement along with the `open()` function to open a file.

KEY CONCEPT

The **with** statement (a reserved keyword) is used with the `open()` function and simplifies exception handling as it incorporates all of the common file tasks automatically. This includes errors in finding the files, the process of closing the file, and other common issues. Using the `with` statement is more beneficial than just using the `open()` function, since there is no need to utilize the `try` and `except` statements to handle exceptions like we did in the previous lesson. Also, using the `with` statement automatically closes out the file after we are complete.

Then, in the same line, we open up `FILENAME` using the newline parameter, which takes the input of the file and converts the *\n* (newline) to be changed to *""* (blank) which removes the newline character at the end of the string. At the end of the statement, we see the file handler using the `as` keyword to set the variable `file` as the alias.

We then create a variable called `reader` that passes `file` to the `csv.reader()` function. The **csv.reader() function**, which is part of the `csv` module, takes a line from the file and splits the data into individual data elements if the data is separated by a comma. In our .csv file, what we plan is having it contain the employee ID and the salary. Then we have a `for` loop that moves through the rows of the variable `reader` and uses the `.append()` method to add the data separated by commas to the employees list. Finally, we return the `employees` list.

TRY IT

Directions: Next, add the function `read_employees()` to your program.

Using just employee ID and salary, an example of what one line of data may look like is the following:

1,50000

This logically would map the employee ID of 1 to have a salary of $50000. Each line in the file would correspond to another employee. However, there are some potential errors and problems that could exist. Let's give it a try by switching the name slightly for the file to `wrong.csv` file instead of `employees.csv` file.

EXAMPLE

```
import csv
import sys

FILENAME = "wrong.csv"

#read the employees from the file
def read_employees():
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees

read_employees()
```

On the last line, we make a call to the `read_employees()` function. Let's see what happens.

TRY IT

Directions: Try switching out the filename for `wrong.csv` and running the program. Do you see the error below?

```
Traceback (most recent call last):
  File "/home/main.py", line 15, in <module>
    read_employees()
  File "/home/main.py", line 9, in read_employees
    with open(FILENAME, newline="") as file:
FileNotFoundError: [Errno 2] No such file or directory: 'wrong.csv'
```

We have an error as expected, as there is no `wrong.csv` file. However, even if the file does not exist, we want to handle this error gracefully and exit the program. Let's define a function called `exit_program()` that will be used to exit the program we've created using the `sys.exit()` function call from the `sys` module. This function will exit out of the current program. Before we do so, we'll make sure that we print a message to let the user know that the program is ending.

EXAMPLE

```
#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()
```

TRY IT

Directions: Now add this existing function to your program.

The `exit_program` function that we just created will only be used if there is a serious issue with the program that we cannot recover from. Otherwise, we will not call this function. The reason we created this function is to provide feedback to the user when we exit the program.

Before we even attempt to open the file, we'll set up a `try` and `except` statement. Specifically, we will catch the FileNotFoundError that we see in the error message above. If the file is not found, we'll output "the file is not found" and call the `exit_program()` function. We will also have a catchall type of exception handling that will output the exception. Right now we have seen the FileNotFoundError but this is only one of the possibilities of problems that can occur. Any other unknown errors that may occur would be caught here and we can continue the program execution rather than exit the program.

EXAMPLE

```
#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()
```

We are adding the `try` and `except` statements to our `read_employee()` function. The `try` statement will encompass our opening and appending as normal, considering the file was found. The first `except` statement will catch the FileNotFoundError and let the user know that the program could not find the `FILENAME`. The second `except` statement is the catchall exception handler that will display the type of error and the error itself.

Let's see what happens now if we run the code up to this point with the `wrong.csv` file:

```
Could not find wrong.csv file.
Traceback (most recent call last):
  File "/home/main.py", line 10, in read_employees
    with open(FILENAME, newline="") as file:
FileNotFoundError: [Errno 2] No such file or directory: 'wrong.csv'
```

This is much better now. Although we see a separate error in the output, this may not show up in other systems that we run the code in. Let's see what happens when we correct the program and now run it as-is:

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"

#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()

#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()

read_employees()
```

TRY IT

Directions: Go ahead and add the expectation handling using the `try` and `except` statements to our `read_employee()` function. Notice that the call to the `read_employees()` function needs to be at the bottom.

There is no output as there were no errors. The program up to this point shouldn't have any errors to begin with. Now we can move onto the next step, writing to the file using the `csv.writer()` function.

EXAMPLE

```
#write employees to files
def write_employees(employees):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(employees)
    except Exception as e:
        print(type(e), e)
        exit_program()
```
In this function called `write_employees()`, we have the parameters set to take the employees list. Similar to the `read_employees()` function, we will also incorporate the `try` and `except` statements since we are setting this function up for error handling as well. Within the `try` statement, we are opening up the filename employees.csv file that we have with the parameters of write ('w') and newline equal to blanks on the `open()` function. Then, using the `csv.writer()` function, we assign the `writer` variable to output to the file in a .csv format. The **csv.writer()** function, which is part of the `csv` module, outputs data and automatically adds in commas and quotes to separate out those data elements into .csv file format.

Writing to the file is as simple as passing in the `employees` list to the `.writerows()` function of the `csv` module. The **writerows()** function, part of the `csv` module, takes the input from the data lists and adds in the commas and quotes one line at a time. Notice that we do not have a specific catch on the FileNotFoundError since we are using the write ('w') mode of the `open()` function. If the file does not exist, it will simply create the file so the FileNotFoundError will not be raised as an issue. This function on its own isn't useful yet as we need to have a means to add to the file.

TRY IT

Directions: Next, add this `write_employees()` function to your program.

Let's take the next step to create a function that prompts the user for the employee ID and the salary. We will name this function `add_employee()` and pass in the `employees` list.

EXAMPLE

```
#add employee to the list
def add_employee(employees):
    empid = input("Enter the employee ID: ")
    sal = input("Enter the salary of the employee: ")
    employee = [empid,sal]
    employees.append(employee)
    write_employees(employees)
    print(f"Employee {empid}: {sal} was added.\n")
```
After reading in the `empid` (employee ID) and `sal` (salary) from the user, we create an variable called `employee` as a list containing the `empid` and `sal` values.

BIG IDEA

You may have noticed that we have been creating variables that are very similar—employee and employees—inside our functions. As long as these variables are within local scope, we will have no issues. Remember, if we were to pull one or two of these out from the function scope, we could potentially have scope errors like naming collisions. Always keep in mind that when creating variables, know what they are called and where they exist.

This newly created variable `employee` is appended to the `employees` list that was passed into the function. After this is done, we call the `write_employees()` function passing in the employees list. After that has been returned, we output to the screen that the employee was added.

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"

#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()

#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()

#write employees to files
def write_employees(employees):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(employees)
    except Exception as e:
        print(type(e), e)
        exit_program()


#add employee to the list
def add_employee(employees):
    empid = input("Enter the employee ID: ")
    sal = input("Enter the salary of the employee: ")
    employee = [empid,sal]
    employees.append(employee)
    write_employees(employees)
    print(f"Employee {empid}: {sal} was added.\n")

employees = read_employees()
add_employee(employees)
```

Let's put this together with what we have so far by first returning the `employees` list from the `read_employees()` function and then we will call the `add_employee()` function.

TRY IT

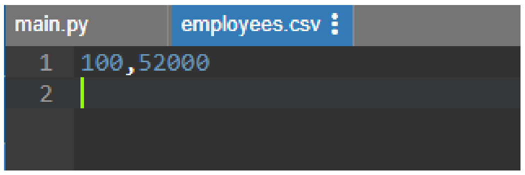
Directions: Ok, let's add the `add_employee()` function to your program. Remember to also add the calls to the `read_employees()` and `add_employee()` functions at the bottom. When finished, run the program and give input for the employee ID and salary of your first employee. We are using 100 as ID and 52,000 as salary.

```
Enter the employee ID: 100
Enter the salary of the employee: 52000
Employee 100: 52000 was added.
```

On-screen, it looks correct so far. Let's take a look at what we have in the `employees.csv` file.

TRY IT

Directions: See if your `employee.csv` file is showing the correct data too.



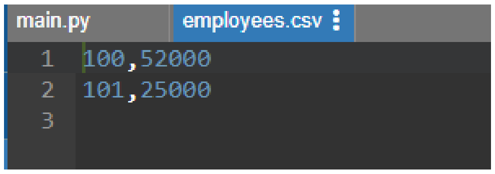
```
100,52000

```

This looks good as well, as the data was correctly saved. Let's run the program again now.

```
Enter the employee ID: 101
Enter the salary of the employee: 25000
Employee 101: 25000 was added.
```

Looking back again at the `employees.csv` file:


main.py employees.csv

```
100,52000
101,25000
```

This is quite useful but it's probably not always ideal to have to keep looking at the `employees.csv` file. Rather, we can create a function to output the list of employees. Remember, back when we first learned about loops, we discussed that an iterable is an object that can return its members one at a time. The `enumerate()` function from Python allows us to loop in the same way with the data. The **enumerate() function** returns the current count of the current iteration to use if we need it and the value of the item at the current iteration. In our case, this is the employee listing of the employee ID and salary within the `employees` list.

EXAMPLE

```
#display the list of employees
def list_employees(employees):
    for i, employee in enumerate(employees, start=1):
        print(f"{i} Employee ID: {employee[0]} (${employee[1]})")
    print()
```

Let's break down this function that we are calling `list_employees()`. Here we are again passing the `employees` list to this function. In a `for` loop using i as an iterative variable, we are using the `enumerate()` function on the employees list with a parameter to start the count at 1 rather than the default of 0. However, we are only using that as part of the output. The print line formats the output so that it is simple to read. Remember that the {} (curly brackets) within the `print()` function indicate specific variables that we are replacing in the output string. So, the formatting will start with the iterative variable (`i`) in curly brackets, which will render as a number, then the string "Employee ID:, then another section of curly brackets that output the first element of the `employees` list (employee ID positioned at 0) and the salary from the `employees` list (positioned at 1). Notice that we are using the dollar sign ($) for currency. We could have used any currency symbol that is local to us.

Let's see what happens when we run this function with the rest of our program.

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"

#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()

#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()

#write employees to files
def write_employees(employees):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(employees)
    except Exception as e:
        print(type(e), e)
        exit_program()

#add employee to the list
def add_employee(employees):
    empid = input("Enter the employee ID: ")
    sal = input("Enter the salary of the employee: ")
    employee = [empid,sal]
    employees.append(employee)
    write_employees(employees)
    print(f"Employee {empid}: {sal} was added.\n")

#display the list of employees
def list_employees(employees):
    for i, employee in enumerate(employees, start=1):
        print(f"{i}. Employee ID: {employee[0]} (${employee[1]})")
    print()

employees = read_employees()
list_employees(employees)
```

TRY IT

Directions: Now go ahead and add the `list_employees()` function to your program. Notice that we replaced the call to the `add_employee()` function with the new `list_employees()` function. This time the program will not have input, only listing the contents of the `employee.csv`. Remember to keep the `read_employees()` and `list_employees()` functions at the bottom. When finished, run the program.

```
1. Employee ID: 100 ($52000)
2. Employee ID: 101 ($25000)
```

The output is much cleaner now as we have a centralized place to see the list of employees with the employee ID and the salary for each.

TERMS TO KNOW

with
The `with` statement (a reserved keyword) is used with the `open()` function and simplifies exception handling as it incorporates all of the common file tasks automatically. This includes errors in finding the files, the process of closing the file, and other common issues.

csv.reader()
The `csv.reader()` function, which is part of the `csv` module, takes a line from the file and splits the data into individual data elements if the data is separated by a comma.

csv.writer()
The `csv.writer()`, which is part of the `csv` module, outputs data and automatically adds in commas and quotes to separate out those data elements into .csv file format.

writerows()
The `writerows()` function, part of the `csv` module, takes the input from the data lists and adds in the commas and quotes one line at a time.

enumerate()
The `enumerate()` function returns back the current count of the current iteration to use if we need it and the value of the item at the current iteration.

2. Delete From a File

Next, let's create a function that will remove an employee from the file based on the employee ID.

EXAMPLE

```
#delete an employee based on ID
def delete_employee(employees):
    found = False
    number = input("Enter in the employee ID: ")
    for i, employee in enumerate(employees, start=0):
      if (employee[0] == number):
        print(f"Employee was deleted.\n")
        employee = employees.pop(i)
        found = True

    if (found == False):
      print("Employee was not found.\n")
    else:
      write_employees(employees)
```

In this function, we are calling `delete_employee()`. We again passed in the `employees` list. We then set a variable called `found` to False initially (only if an employee ID was found later would this variable be changed to True). We will prompt the user for an employee ID, placing the input in a variable `number`. In the `for` loop again, using `i` as an iterative variable (remember we are using this variable locally within the function so using `i` again as the variable name is fine), we are using the `enumerate()` function on the `employees` list and using the start parameter to start the default at 0. Using an `if` statement on each iteration of the loop, it will compare the user's input (`number`) with employee IDs of the list. If found, it will print "Employee was deleted" to the screen and use the `.pop()` method to remove the element from the list. Then, the `found` variable is set to True. Once the loop has moved through all elements of the list and completed, the final `if` statement will check if an employee ID was found or not. If it was not found (`found` still set to False) it will output that "Employee was not found"; otherwise, it will call the `write_employees()` function that will write an updated list with the element removed.

Let's give it a try now.

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"

#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()

#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()

#write employees to files
def write_employees(employees):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(employees)
    except Exception as e:
        print(type(e), e)
        exit_program()

#add employee to the list
def add_employee(employees):
    empid = input("Enter the employee ID: ")
    sal = input("Enter the salary of the employee: ")
    employee = [empid,sal]
    employees.append(employee)
    write_employees(employees)
    print(f"Employee {empid}: {sal} was added.\n")

#display the list of employees
def list_employees(employees):
    for i, employee in enumerate(employees, start=1):
        print(f"{i}. Employee ID: {employee[0]} (${employee[1]})")
    print()

#delete an employee based on ID
def delete_employee(employees):
    found = False
    number = input("Enter in the employee ID: ")
    for i, employee in enumerate(employees, start=0):
      if (employee[0] == number):
        print(f"Employee was deleted.\n")
        employee = employees.pop(i)
        found = True

    if (found == False):
      print("Employee was not found.\n")
    else:
      write_employees(employees)

employees = read_employees()
list_employees(employees)
delete_employee(employees)
```

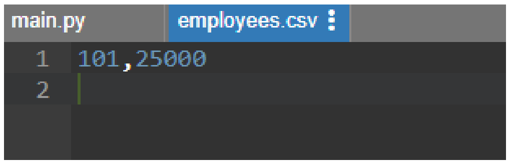
TRY IT

Directions: Now go ahead and add the `delete_employees()` function to your program. Remember to keep all the function calls at the bottom. When finished, run the program. Input an employee ID that is in the `employee.csv` file (we are using 100 in the example). You should see the following.

```
1. Employee ID: 100 ($52000)
2. Employee ID: 101 ($25000)

Enter in the employee ID: 100
Employee was deleted.
```

Looking at the `employees.csv` file, the result is correct; now only employee ID 101 is left.



```
101,25000
```

If we run the program again and try to enter in 100, it should prompt us that the employee was not found.

```
1. Employee ID: 101 ($25000)

Enter in the employee ID: 100
Employee was not found.
```

3. Adding a Main Menu

Now that we have this program, let's create a function to truly make this a program. We'll set up a basic menu that will allow the user to know how to run the program to call each of these functions with a basic menu program.

⮫ EXAMPLE

```
def display_menu():
    print("The Employee Salary List program")
    print()
    print("LIST OF COMMANDS")
    print("list - List all employees")
    print("add -  Add an employee")
    print("del -  Delete an employee")
    print("exit - Exit program")
    print()
```

Here we have a function called `display_menu()`. Right now it is just an output of what is available in our program. We will give the user the list of commands that are expected to be used to access different functions of the program.

And now for the last part.

⮫ EXAMPLE

```
display_menu()
employees = read_employees()
while True:
    command = input("Command: ")
    if command.lower() == "list":
        list_employees(employees)
    elif command.lower() == "add":
        add_employee(employees)
    elif command.lower() == "del":
        delete_employee(employees)
    elif command.lower() == "exit":
        break
    else:
        print("Not a valid command. Please try again.\n")
print("Ending Salary Program")
```

We are going to remove all past calls at the bottom of the program and replace those with this first call to the `display_menu()` function. Calling the `display_menu()` function gives the user the list of commands on the screen. We then read and build the `employees` list from the `employee.csv` file. After the list is set up, a `while` loop is set to indefinitely iterate by passing in True. This allows the user to perform actions on our `employees` list using the functions we have created. The `while` loop will continue to iterate until the exit command is entered. Once exit is entered, the `break` statement will terminate the `while` loop and allow the final `print()` function to write "Ending Salary Program" to the screen. With the added conditionals inside the `while` loop, if any command is not entered correctly, it will fall to the `else` statement, producing an output to the user that their input was not a valid command and to please try again.

The completed program will look like the following:

EXAMPLE

```
import csv
import sys

FILENAME = "employees.csv"

#exiting the program
def exit_program():
    print("Terminating program.")
    sys.exit()

#read the employees from the file
def read_employees():
    try:
        employees = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                employees.append(row)
        return employees
    except FileNotFoundError as error:
        print(f"Could not find {FILENAME} file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()

#write employees to files
def write_employees(employees):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(employees)
    except Exception as e:
        print(type(e), e)
        exit_program()
```

```
#add employee to the list
def add_employee(employees):
    empid = input("Enter the employee ID: ")
    sal = input("Enter the salary of the employee: ")
    employee = [empid,sal]
    employees.append(employee)
    write_employees(employees)
    print(f"Employee {empid}: {sal} was added.\n")

#display the list of employees
def list_employees(employees):
    for i, employee in enumerate(employees, start=1):
        print(f"{i}. Employee ID: {employee[0]} (${employee[1]})")
    print()

#delete an employee based on ID
def delete_employee(employees):
    found = False
    number = input("Enter in the employee ID: ")
    for i, employee in enumerate(employees, start=0):
      if (employee[0] == number):
        print(f"Employee was deleted.\n")
        employee = employees.pop(i)
        found = True

    if (found == False):
      print("Employee was not found.\n")
    else:
      write_employees(employees)

def display_menu():
    print("The Employee Salary List program")
    print()
    print("LIST OF COMMANDS")
    print("list - List all employees")
    print("add -  Add an employee")
    print("del -  Delete an employee")
    print("exit - Exit program")
    print()

display_menu()
employees = read_employees()
```

```
while True:
    command = input("Command: ")
    if command.lower() == "list":
        list_employees(employees)
    elif command.lower() == "add":
        add_employee(employees)
    elif command.lower() == "del":
        delete_employee(employees)
    elif command.lower() == "exit":
        break
    else:
        print("Not a valid command. Please try again.\n")
print("Ending Salary Program")
```

TRY IT

Directions: Now go ahead and add the `display_menu()` function to your program. Then, add the call to that function, the read and build of the `employees` list, the `while` loop of all commands, and finally, the last `print()` function indicating the end of the program. All of this goes at the bottom of the program like shown above.

Now we can give this program a try and see if the results from the employee class program are as expected.

```
The Employee Salary List program

LIST OF COMMANDS
list - List all employees
add -  Add an employee
del -  Delete an employee
exit - Exit program

Command: list
1. Employee ID: 101 ($25000)

Command: add
Enter the employee ID: 100
Enter the salary of the employee: 80000
Employee 100: 80000 was added.

Command: add
Enter the employee ID: 300
Enter the salary of the employee: 40000
Employee 300: 40000 was added.
```

```
Command: list
1. Employee ID: 101 ($25000)
2. Employee ID: 100 ($80000)
3. Employee ID: 300 ($40000)

Command: del
Enter in the employee ID: 100
Employee was deleted.

Command: list
1. Employee ID: 101 ($25000)
2. Employee ID: 300 ($40000)

Command: exit
Ending Salary Program
```

Everything looks accurate. Now the next time we launch the program, it should work exactly as expected with the `employees` list available to start with.

To see the final version of this program visit Sophia's Python code page

SUMMARY

In this lesson, we continued with our Employee Class Program, this time **storing employee information to a file**. This file was a .csv file (comma separated file) instead of the typical text files in recent lesson examples. We created functions that allowed us to manipulate the data read from this file. Finally, using a selection of functions we built, we **added a main menu** program that allows a user to list out all employees in the file, add employee IDs and salaries to the file, **delete employees from the file**, and exit the program altogether.

Best of luck in your learning!

TERMS TO KNOW

csv.reader()

The `csv.reader()` function, which is part of the `csv` module, takes a line from the file and splits the data into individual data elements if the data is separated by a comma.

csv.writer()

The `csv.writer(),` which is part of the `csv` module, outputs data and automatically adds in commas and quotes to separate out those data elements into .csv file format.

enumerate()

The `enumerate()` function returns back the current count of the current iteration to use if we need it and the value of the item at the current iteration.

with

The `with` statement (a reserved keyword) is used with the `open()` function and simplifies exception handling as it incorporates all of the common file tasks automatically. This includes errors in finding the files, the process of closing the file, and other common issues.

writerows()

The `writerows()` function, part of the `csv` module, takes the input from the data lists and adds in the commas and quotes one line at a time.