

Introduction:

Module 09 examines creating modules in Python to organize functions and classes. It demonstrates how to create a Python application using two or more script files with a "Main" module that runs at the start of the program. It shows how to use the Python system variable "__name__", that returns the string "__main__" when you execute a script directly, to verify if a script is running as the Main module. I discovered how to create modules with multiple classes and linking modules together, designing modules by separation of concerns to hold classes or functions to support data, processing, or presentation, how classes can "inherit" code from another class to form parent (Super or Base) class - child (Derived or Sub) class relationships with Python's "object" class being the ultimate parent class of all classes. I learned how to use a test harness script to assess modules as you create them and using the Unified Modeling Language (UML) as a standard way of modeling relationships between software components.

The objective of assignment 09 is to create the "Employee List" program, which consists of four modules and a data file, and add code to each script based on its purpose to complete the program. The program consists of four separations of concerns: Data, Processing, Presentation (Inputs-Outputs) and Main. There is also a test harness script to test the modules. This document is a breakdown of the logic I created to complete assignment 09 with the original **code** from the provided modules and scripts and the **code** I added or modified to complete the program.

Scripts and Data File(s):

Main.py
DataClasses.py
ProcessingClasses.py
IOClasses.py
TestHarness.py
EmployeeData.txt

The "Main" module (taken from Module 09 – Listing13.py) provided the following **TODO:** steps and **pseudocode** as scripted by the original programmer "RRoot".

Main.py:

TODO: Import Modules
TODO: Add Data Code to the Main body

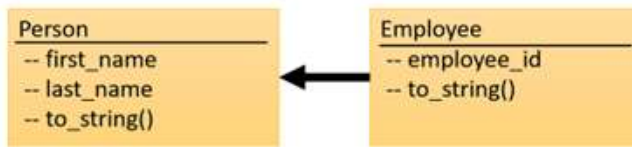
Pseudocode:
Load data from file into a list of employee objects when script starts
Show user a menu of options
Get user's menu option choice
Show user current data in the list of employee objects
Let user add data to the list of employee objects
let user save current data to file
Let user exit program

UML (Unified Modeling Language):

I have provided excerpts from Module 09 “Mod9PythonProgrammingNotes.docx” (RRoot, 2022, p. 12-14) that illustrate UML for Assignment 09.

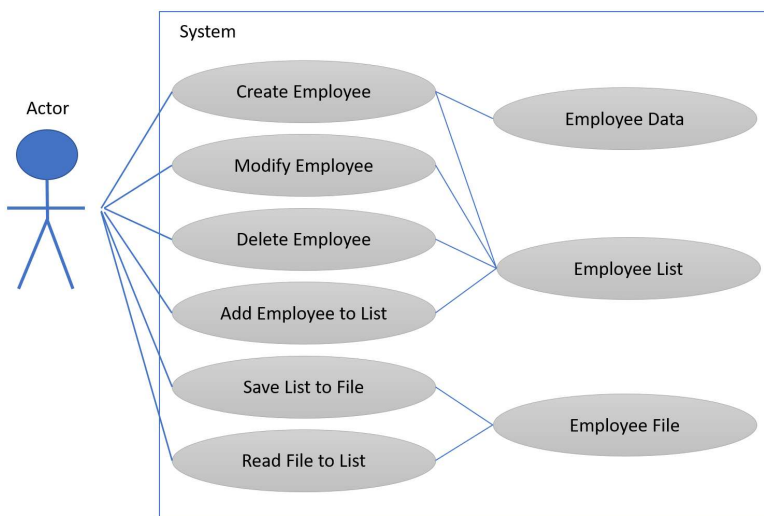
Class Diagram:

Relationship between classes:



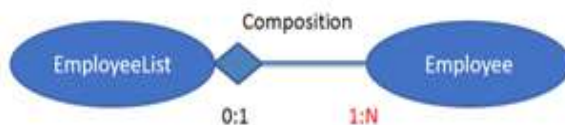
Use Case Diagram:

Components within the software and the actions they perform. My assignment 09 program does not include “Modify Employee”.



Composition Diagram:

Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). ... **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child).



The numbers used in the diagram indicate "Cardinality" or the expected number of objects in the relationship. For example, "0:1" near the EmployeeList object means that there will be either zero or one EmployeeList object connected to Employee objects. The "1:N" near the Employee object indicates that there must be at least one, but possibly many (of an undefined Number). Composition represents a strong bond, and in this case, the numbers indicate that without at least one Employee, the EmployeeList is not needed.

Main.py

In the Main module, I added my name and date to the changelog of the script header and code for all **TODO** and **pseudocode** steps.

```
# ----- #
# Title: Assignment 09 - Main.py
# Description: Working with Modules
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created script
# RRoot,1.1.2030,Added pseudo-code to start assignment 9
# ALarkin,6.14.2022,Modified code to complete assignment 9
# ----- #
```

```
strFileName = "EmployeeData.txt"
intEmployeeId = 0
strFirstName = ""
strLastName = ""
```

TODO: Import Modules

```
# print ( __name__ )
if __name__ == "__main__":
    from DataClasses import Employee as EMP
    from ProcessingClasses import FileProcessor as FP
    from IOClasses import EmployeeIO as EIO
    # print("This file is the starting point of my program!")
else:
    raise Exception("This file was not created to be imported")
```

```
# Main Body of Script ----- #
```

TODO: Add Data Code to the Main body

TODO: Load data from file into a list of employee objects when script starts

```
lstOfEmployeeObjects = FP.read_data_from_file(strFileName)
```

The first step in the main body sets the list variable "lstOfEmployeeObjects" by calling the ProcessingClasses module FileProcessor class read_data_from_file() method to load data from the EmployeeData.txt. file if it exists.

```
while (True):
```

TODO: Show user a menu of options

```
EIO.menu() # Shows menu
```

Executes the IOClasses module menu() method to print a display of menu options.

TODO: Get user's menu option choice

```
strChoice = EIO.choice()
```

Executes the IOClasses module EmployeeIO class choice() method to prompt the user to input a menu option choice.

TODO: Show user current data in the list of employee objects

```
if strChoice.strip() == '1':  
    EIO.employee_list(lstOfEmployeeObjects) # Show current data in the list/table  
    continue
```

Executes menu option one that calls the IOClasses module EmployeeIO class employee_list() method to list current employee data from the list table of rows lstOfEmployeeObjects.

TODO: Add data to the list of employee objects

```
elif strChoice.strip() == '2':  
    intEmployeeId, strFirstName, strLastName = EIO.input_employee_data()  
    if not (intEmployeeId == 0 or strFirstName == "" or strLastName == ""):  
        lstOfEmployeeObjects.append(EMP(employee_id=intEmployeeId,  
                                         first_name=strFirstName,  
                                         last_name=strLastName))  
    else:  
        continue
```

Executes menu option two that calls the IOClasses module EmployeeIO class input_employee_data() method to prompt the user to input an employee id, first name and last name and then appends that data to the list table of rows if not intEmployeeId == 0 or strFirstName == "" or strLastName == "". This ensure only valid data are appended to lstOfEmployeeObjects.

```
# Added option to remove data from the list of employee objects  
elif strChoice.strip() == '3':  
    EIO.remove_employee_data(lstOfEmployeeObjects)  
    continue
```

Executes menu option three that calls the IOClasses module EmployeeIO class remove_employee_data() method to remove a valid employee from the list table of rows lstOfEmployeeObjects.

TODO: Save current data to file

```
elif strChoice == '4':  
    FP.save_data_to_file(strFileName, lstOfEmployeeObjects)  
    continue
```

Executes menu option four that calls the ProcessingClasses module FileProcessor class save_data_to_file() method and passes in the strFileName and lstOfEmployeeObjects parameters to save the employee data to the EmployeeData.txt delimited text file.

```
# TODO: Exit program
```

```
elif strChoice == '5':  
    break
```

```
# Exit the program
```

```
input("\nPress the enter key to exit.")
```

Executes menu option five that breaks the loop and prompts the user with an input() function asking the user to press the "enter" key to exit and close the program.

```
# Main Body of Script ----- #
```

DataClasses.py

In the DataClasses module, I added my name and date to the changelog of the script header and docstrings. I added types for all attributes in both classes and strip(), lower() and title() methods to the attribute getter properties in the Person() class for the first_name and last_name attributes to ensure data consistency In Employee(Person) class, I added the "int" type to the "value" in the setter property for the employee_id attribute.

```
# ----- #
# Title: Assignment 09 - DataClasses.py
# Description: A module of data classes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ALarkin,6.14.2022,Modified code to complete assignment 9
# ----- #

if __name__ == "__main__":
    raise Exception("This file is not meant to run by itself")

class Person(object): # Inherits from object
    """Stores data about a person:

    properties:
        first_name: (string) person first name
        last_name: (string) person last name
    methods:
        to_string() returns comma separated employee data (alias for __str__())
        changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        ALarkin,6.14.2022,Modified class to complete assignment 9
    """

    # -- Constructor --
    def __init__(self, first_name: str, last_name: str):
        # -- Attributes --
        self.__first_name = first_name
        self.__last_name = last_name

    # -- Properties --
    @property
    def first_name(self):
        return str(self.__first_name).strip().lower().title()

    @first_name.setter
    def first_name(self, value):
        self.__first_name = value

    @property
    def last_name(self):
        return str(self.__last_name).strip().lower().title()

    @last_name.setter
    def last_name(self, value):
        self.__last_name = value
```

```

# -- Methods --
def to_string(self):
    """ Explicitly returns a string with this object's data """
    return self.__str__()

def __str__(self):
    """ Implicitly returns a string with this object's data """
    return self.first_name + ',' + self.last_name

class Employee(Person): # Inherits from Person
    """Stores data about an employee:

    properties:
        employee_id: (int) employee ID

        first_name: (string) employee first name

        last_name: (string) employee last name
    methods:
        to_string() returns comma separated employee data (alias for __str__())
    changelog: (When, Who, What)
        RRoot, 1.1.2030, Created Class
        ALarkin, 6.14.2022, Modified class to complete assignment 9
    """

    def __init__(self, employee_id: int, first_name: str, last_name: str):
        # Attributes
        self.__employee_id = employee_id
        self.first_name = first_name
        self.last_name = last_name

    # --Properties--
    @property
    def employee_id(self):
        return self.__employee_id

    @employee_id.setter
    def employee_id(self, value):
        self.__last_name = int(value)

    # --Methods--
    def to_string(self): # Overrides the original method (polymorphic)
        """ Explicitly returns a string with this object's data """
        # Linking to self.__str__() does not work with inheritance
        data = super().__str__() # get data from parent(super) class
        return str(self.employee_id) + ',' + data

    def __str__(self): # Overrides the original method (polymorphic)
        """ Implicitly returns field data """
        data = super().__str__() # get data from parent(super) class
        return str(self.employee_id) + ',' + data
# --End of Class --

```

ProcessingClasses.py

In the ProcessingClasses module, I added my name and date to the changelog of the script header and docstrings. I added an import for the DataClasses module. I replaced the read_data_from_file() and save_data_to_file() methods with my own versions.

```
# ----- #
# Title: Assignment 09 - ProcessingClasses.py
# Description: A module of multiple processing classes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ALarkin,6.14.2022,Modified code to complete assignment 9
# ----- #

if __name__ == "__main__":
    raise Exception("This file is not meant to run by itself")
else:
    import DataClasses as DC

class FileProcessor:
    """Processes data to and from a file and a list of objects:

    methods:
        save_data_to_file(file_name,list_of_objects):

        read_data_from_file(file_name): -> (a list of objects)

    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        ALarkin,6.14.2022,Modified class to complete assignment 9
    """

    @staticmethod
    def read_data_from_file(file_name: str):
        """Reads data from a file into a list of dictionary rows:

        :param file_name: (string) with name of file:
        :return: (list) of object rows
        """
        list_of_rows = []
        try:
            import os.path
            isfile_bln = (os.path.isfile(file_name))
            if (isfile_bln == True):
                file = open(file_name, "r")
                for line in file:
                    data = line.split(",")
                    row = DC.Employee(int(data[0]), (data[1]), (data[2]))
                    list_of_rows.append(row)
                file.close()
            except FileNotFoundError as e:
                print("Error file not found:", e, sep='\n')
            except Exception as e:
                print()
```



```

        print("Error reading data from file:", e, sep='\n')
    return list_of_rows

```

Replaced existing code to read data from a file into a list of dictionary rows.

The `read_data_from_file` method is executed when the program starts and includes a try/except block that contains the variable `isfile_bln` that performs a validation on the `file_name` parameter passed to the function `"os.path.isfile"` and returns a Boolean value (True or False). This function also calls the `os.path` module. It is useful when processing files from different places in the system and for different purposes such as for merging, normalizing, and retrieving path names in python. If `isfile_bln` evaluates to "True," `file_name` is determined to be a valid file and I then load data from `EmployeeData.txt` into the file object and unpack it into the list `table list_of_rows`. I added two exceptions for the try/except block, `"FileNotFoundError"` and an exception for general errors.

```

@staticmethod
def save_data_to_file(file_name: str, list_of_objects: list):
    """Writes data from a list of dictionary rows to a file:

    :param file_name: (string) with name of file:
    :param list_of_objects: (list) you want filled with file data:
    :return: status_bln (boolean) return status
    """
    save_bln = False
    try:
        strOverwrite = str(input("Overwrite: " + file_name + "?" + " [y/n] ").strip().lower())
        if (strOverwrite == 'y'):
            file = open(file_name, "w")
            for row in list_of_objects:
                file.write(row.to_string() + "\n")
                # file.write(row.__str__() + "\n")
            file.close()
            save_bln = True
            print() # Add an extra line for looks
            print("*****")
            print("Data Saved")
            print("*****")
        else:
            print("Overwrite = No | File not overwritten")
    except Exception as e:
        print() # adding a new line for looks
        print("Error saving data:", e, sep='\n')
    return save_bln

```

Replaced existing code to write data from a list of dictionary rows to a file.

The `"save_data_to_file"` method executes menu option four to save data. I added a try/except block that begins with the `strOverwrite` variable set to an `input()` statement asking the user for a "y/n" (yes or no) to overwrite the data file. If "no" it executes the `else:` block and prints `"Overwrite = No | File not overwritten"`. If "yes," I call the `open()` function to open the data file and write the `list_of_rows` data to `EmployeeData.txt`. A `print()` statement prints `"Data Saved"` on completion. The exception block catches all general errors.

IOClasses.py

In the IOClasses module, I added my name and date to the changelog of the script header and docstrings. I removed the import for the DataClasses module. I replaced all existing methods with my own versions and added a method to remove employee data.

```
# ----- #
# Title: Assignment 09 - IOClasses.py
# Description: A module of IO classes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ALarkin,6.14.2022,Modified code to complete assignment 9
# ----- #

if __name__ == "__main__":
    raise Exception("This file is not meant to run by itself")

class EmployeeIO:
    """ A class for performing Employee Input and Output

    methods:
        menu():

        choice():

        employee_list(list_of_rows):

        input_employee_data():

        remove_employee_data(lstOfEmployeeObjects):

    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        ALarkin,6.14.2022,Modified class to complete assignment 9
    """

    @staticmethod
    def menu():
        """ Display a menu of options to the user

        :return: nothing
        """
        print("""
        *****
        Employee Data - Option Menu
        *****
        1) Show current employees
        2) Add a new employee
        3) Remove an employee
        4) Save employee data to file
        5) Exit program
        """)
```

```
*****
    ")
    print() # Add an extra line for looks
```

Replaced existing code to display a menu of options to the user.

The menu() method prints a display menu of options to the user that includes five menu options for user selection.

```
@staticmethod
def choice():
    """ Get users menu choice
```

```
:return: (str) choice
    """
    choice = str(input("Choose an option? [1 to 5] - ")).strip()
    print() # Add an extra line for looks
    if str(choice) not in ("1", "2", "3", "4", "5"):
        print("*****")
        print(choice + " is not a valid option")
        print("*****")
        print() # Add an extra line for looks
    return choice
```

Replaced existing code to get user's menu choice.

The choice() method prompts the user via an input() function to choose from menu options 1 to 5 and returns a string variable "choice". An "if" statement validates the user's choice and prints the choice is not a valid option if the input choice is not 1 to 5.

```
@staticmethod
def employee_list(list_of_rows: list):
    """ Print current employees in the list of rows
```

```
:param list_of_rows: (list) of rows you want to display
:return: nothing
    """
    print("***** Current Employee List *****")
    for row in list_of_rows:
        print(str(row.employee_id) + "," + row.first_name + "," + row.last_name)
    print("*****")
    print() # Add an extra line for looks
    return
```

Replaced existing code to print current employees in the list of object rows.

The employee_list() method executes menu option one to show current data from the list table list_of_employee_objects rows.

```

@staticmethod
def input_employee_data():
    """ Get users input data for an employee object

    :return: (int) employee_id, (str) first_name, (str) last_name
    """
    try:
        employee_id = str(input("What is the employee Id? - ").strip())
        first_name = str(input("What is the employee First Name? - ").strip())
        last_name = str(input("What is the employee Last Name? - ").strip())
        print() # Add an extra line for looks
        if not str(employee_id).isnumeric():
            employee_id = 0
            print("*****")
            print("Employee ID must be a number!")
        if str(first_name).isnumeric():
            first_name = ""
            print("First Name must not be a number!")
        if str(last_name).isnumeric():
            last_name = ""
            print("Last Name must not be a number!")
            print("*****")
        return int(employee_id), first_name, last_name
    except Exception as e:
        print("Invalid data entered:", e, sep='\n')
    return

```

Replaced existing code to get users input data for an employee object.

The input_employee_data() method executes menu option two to add data. It has a try/except block that contains three input() statements for the user to enter an employee id, first name and last name. I used the strip() method to remove whitespace and characters. Three “if” statements validate the user (str) inputs. If the user inputs a non-numeric employee id, it sets the employee_id = 0 and prints “Employee ID must be a number!”. If the user inputs a numeric first name, it sets first_name = “” and prints “First Name must not be a number!”. If the user inputs a numeric last name, it sets last_name = “” and prints “Last Name must not be a number!”. It returns int(employee_id), first_name and last_name. The exception block catches all general errors.

```
@staticmethod
```

```
def remove_employee_data(lstOfEmployeeObjects: list):
```

```
    """ Remove data for an employee object
```

```
    :param lstOfEmployeeObjects
```

```
    :return: nothing
```

```
    """
```

```
    try:
```

```
        employee_id = int(input("Enter an employee id to remove: ").strip())
```

```
        first_name = ""
```

```
        last_name = ""
```

```
        remove_bln = False # verify that the data was found
```

```
        for row in lstOfEmployeeObjects:
```

```
            if row.employee_id == employee_id:
```

```
                first_name = row.first_name
```

```
                last_name = row.last_name
```

```
                lstOfEmployeeObjects.remove(row)
```

```
                remove_bln = True
```

```
        if remove_bln:
```

```
            print() # Add an extra line for looks
```

```
            print("*****")
```

```
            print(str(first_name + " " + last_name) + " removed from Employee List. ")
```

```
            print("*****")
```

```
        else:
```

```
            print() # Add an extra line for looks
```

```
            print("*****")
```

```
            print(str(employee_id) + " is not a valid Employee ID. ")
```

```
            print("*****")
```

```
    except Exception as e:
```

```
        print() # adding a new line for looks
```

```
        print("Error Removing Data:", e, sep='\n')
```

Added new code to remove data for an employee object.

The `remove_employee_data()` method executes menu option three to remove data. It has a `try/except` block that contains an `input()` statement for the user to enter an employee id to remove. I used the `strip()` method to remove whitespace and characters. I set a boolean value `remove_bln = False` then run a “for” loop to remove the employee from the list table `lstOfEmployeeObjects`. If a valid row is removed, it sets `remove_bln = True` and then executes an `if` statement for `remove_bln = True` and prints the employee’s name removed from the employee list. If `remove_bln = False`, the `else:` block prints the employee’s name is not in employee list. The exception block catches all general errors.

TestHarness.py

In the TestHarness.py script, I added my name and date to the changelog of the script. I modified or added code to assess classes and methods in the DataClasses, ProcessingClasses and IOClasses modules.

```
# ----- #
# Title: Assignment 09 - TestHarness.py
# Description: A main module for testing
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ALarkin,6.14.2022,Update script
# ----- #

if __name__ == "__main__":
    from DataClasses import Employee as EMP
    from ProcessingClasses import FileProcessor as FP
    from IOClasses import EmployeeIO as EIO
else:
    raise Exception("This file was not created to be imported")

# Test data module: [DataClasses.py]
print()
print("Test data module: [DataClasses.py]")
print("*****")
print()
p1 = EMP(1, "Haruka", "Nanase")
p2 = EMP(2, "Makoto", "Tachibana")
p3 = EMP(3, "Nagisa", "Hazuki")
p4 = EMP(4, "Rin", "Matsuoka")
lstTable = [p1, p2, p3, p4]
for row in lstTable:
    print(row.to_string(), type(row))
print()

# Test processing module: [ProcessingClasses.py]
print("Test processing module: [ProcessingClasses.py]")
print("*****")
print()
print("[save_data_to_file:]")
print()
FP.save_data_to_file("EmployeeData.txt", lstTable)
print()
print("[read_data_from_file:]")
print()
lstFileData = FP.read_data_from_file("EmployeeData.txt")
lstTable.clear()
for row in lstFileData:
    lstTable.append(row)
for row in lstTable:
    print(row.to_string(), type(row))
print()
```

```

# Test IO module: [IOClasses.py]
print("Test IO module: [IOClasses.py]")
print("*****")
print()
print("[menu:]")
print()
EIO.menu()
print()
print("[employee_list:]")
print()
EIO.employee_list(lstTable)
print()
print("[choice:]")
print()
print(EIO.choice())
print()
print("[input_employee_data:]")
print()
print(EIO.input_employee_data()) # (9, 'TestFirst', 'TestLast')
print()

```

Run the script from PyCharm.

Ashton > UW > Foundations of Python > _PythonClass > Module09 > Assignment09					
Name	Status	Date modified	Type	Size	
__pycache__	✓	6/13/2022 3:37 PM	File folder		
Assignment09.docx	🔄	6/14/2022 11:49 AM	Microsoft Word Document	294 KB	
DataClasses.py	✓	6/13/2022 3:29 PM	Python File	4 KB	
IOClasses.py	✓	6/14/2022 11:39 AM	Python File	5 KB	
Main.py	✓	6/13/2022 3:34 PM	Python File	3 KB	
ProcessingClasses.py	✓	6/13/2022 3:36 PM	Python File	3 KB	
TestHarness.py	✓	6/13/2022 3:41 PM	Python File	2 KB	

EmployeeData.txt file does not exist on initial program execution.

Execute TestHarness.py to assess the modules and create EmployeeData.txt.

Execute the Employee List program from module Main.py.

1. Choose option one to “Show current employees” to read data from EmployeeData.txt and append the data into a list of objects.

2. Choose option two to add a new employee with invalid and valid data inputs.

3. Choose option four to save the data to file with overwrite “no and then “yes” and show the data added in EmployeeData.txt.

4. Choose option three with an invalid Employee ID and a valid Employee ID to remove the data added with option two and then option four to save the data.

5. Choose option two to exit the program.

Test data module: [DataClasses.py]

```
1,Haruka,Nanase <class 'DataClasses.Employee'>
2,Makoto,Tachibana <class 'DataClasses.Employee'>
3,Nagisa,Hazuki <class 'DataClasses.Employee'>
4,Rin,Matsuoka <class 'DataClasses.Employee'>
```

Test processing module: [ProcessingClasses.py]

[save_data_to_file:]

Overwrite: EmployeeData.txt? [y/n] *y*

Data Saved

[read_data_from_file:]

```
1,Haruka,Nanase <class 'DataClasses.Employee'>
2,Makoto,Tachibana <class 'DataClasses.Employee'>
3,Nagisa,Hazuki <class 'DataClasses.Employee'>
4,Rin,Matsuoka <class 'DataClasses.Employee'>
```

Test IO module: [IOClasses.py]

[menu:]

Employee Data - Option Menu

- 1) Show current employees
- 2) Add a new employee
- 3) Remove an employee
- 4) Save employee data to file
- 5) Exit program

[employee_list:]

***** Current Employee List *****

- 1,Haruka,Nanase
- 2,Makoto,Tachibana
- 3,Nagisa,Hazuki
- 4,Rin,Matsuoka

```
[choice:]
```

```
Choose an option? [1 to 5] - 1
```

```
1
```

```
[input_employee_data:]
```

```
What is the employee Id? - 9
```

```
What is the employee First Name? - TestFirstName
```

```
What is the employee Last Name? - TestLastName
```

```
(9, 'TestFirstName', 'TestLastName')
```

```
[remove_data:]
```

```
Enter an employee id to remove: 9
```

```
*****
```

```
9 is not a valid Employee ID.
```

















```
*****
```

```
None
```

```
Process finished with exit code 0
```

```
|
```

EmployeeData.txt created by TestHarness.py execution.

Ashton > UW > Foundations of Python > _PythonClass > Module09 > Assignment09					
Name	Status	Date modified	Type	Size	
 __pycache__		6/14/2022 11:53 AM	File folder		
 Assignment09.docx		6/14/2022 11:57 AM	Microsoft Word Document	539 KB	
 DataClasses.py		6/13/2022 3:29 PM	Python File	4 KB	
 EmployeeData.txt		6/14/2022 11:53 AM	Text Document	1 KB	
 IOClasses.py		6/14/2022 11:39 AM	Python File	5 KB	
 Main.py		6/13/2022 3:34 PM	Python File	3 KB	
 ProcessingClasses.py		6/13/2022 3:36 PM	Python File	3 KB	
 TestHarness.py		6/13/2022 3:41 PM	Python File	2 KB	

```
*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

Choose an option? [1 to 5] - 1

```
***** Current Employee List *****
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
*****
```

```
*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

Choose an option? [1 to 5] -

```
*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

Choose an option? [1 to 5] - 2

What is the employee Id? - A

What is the employee First Name? - 12345

What is the employee Last Name? - 67890

```
*****
Employee ID must be a number!
First Name must not be a number!
Last Name must not be a number!
*****
```

```
*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

Choose an option? [1 to 5] -

Choose an option? [1 to 5] - 1

***** Current Employee List *****

1,Haruka,Nanase

2,Makoto,Tachibana

3,Nagisa,Hazuki

4,Rin,Matsuoka

Employee Data - Option Menu

1) Show current employees

2) Add a new employee

3) Remove an employee

4) Save employee data to file

5) Exit program

Choose an option? [1 to 5] - 2

What is the employee Id? - 5

What is the employee First Name? - Rei

What is the employee Last Name? - Ryugazaki

Employee Data - Option Menu

1) Show current employees

2) Add a new employee

3) Remove an employee

4) Save employee data to file

5) Exit program

Choose an option? [1 to 5] - 1

***** Current Employee List *****

1,Haruka,Nanase

2,Makoto,Tachibana

3,Nagisa,Hazuki

4,Rin,Matsuoka

5,Rei,Ryugazaki

Employee Data - Option Menu

1) Show current employees

2) Add a new employee

3) Remove an employee

4) Save employee data to file

5) Exit program

Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] n

Overwrite = No | File not overwritten

Employee Data - Option Menu

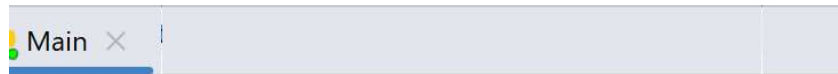
1) Show current employees

2) Add a new employee

3) Remove an employee

4) Save employee data to file

5) Exit program



Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] y

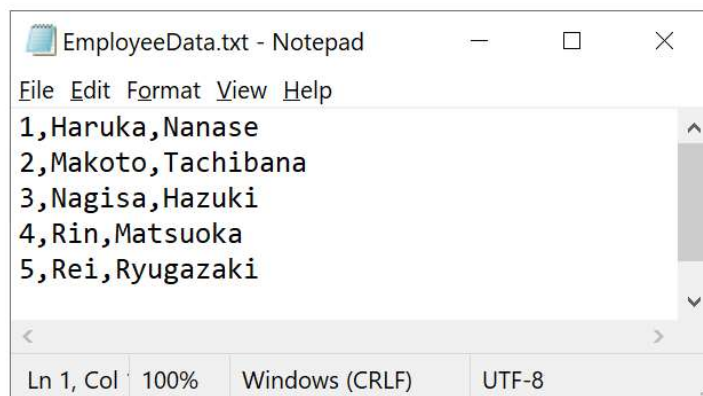
Data Saved

```
*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

Choose an option? [1 to 5] - 1

***** Current Employee List *****

```
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
5,Rei,Ryugazaki
```



Choose an option? [1 to 5] - 3

Enter an employee id to remove: 9

9 is not a valid Employee ID.

Employee Data - Option Menu

- 1) Show current employees
- 2) Add a new employee
- 3) Remove an employee
- 4) Save employee data to file
- 5) Exit program

Choose an option? [1 to 5] - 3

Enter an employee id to remove: 5

Rei Ryugazaki removed from Employee List.

Employee Data - Option Menu

- 1) Show current employees
- 2) Add a new employee
- 3) Remove an employee
- 4) Save employee data to file
- 5) Exit program

Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] y

Data Saved

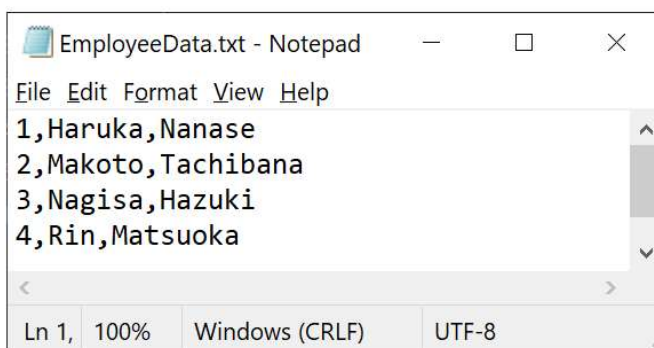
Employee Data - Option Menu

- 1) Show current employees
- 2) Add a new employee
- 3) Remove an employee
- 4) Save employee data to file
- 5) Exit program

Choose an option? [1 to 5] - 1

***** Current Employee List *****

- 1,Haruka,Nanase
- 2,Makoto,Tachibana
- 3,Nagisa,Hazuki
- 4,Rin,Matsuoka



Choose an option? [1 to 5] - 1

***** Current Employee List *****

1,Haruka,Nanase

2,Makoto,Tachibana

3,Nagisa,Hazuki

4,Rin,Matsuoka

Employee Data - Option Menu

1) Show current employees

2) Add a new employee

3) Remove an employee

4) Save employee data to file

5) Exit program

Choose an option? [1 to 5] - 5

Press the enter key to exit.

Process finished with exit code 0

Run the Python Script from the Windows OS Command Shell using the same execution steps used to run the script from PyCharm.

```
C:\WINDOWS\py.exe

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 1

***** Current Employee List *****
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
*****

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 2

What is the employee Id? - A
What is the employee First Name? - 12345
What is the employee Last Name? - 67890

*****
Employee ID must be a number!
First Name must not be a number!
Last Name must not be a number!
*****

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

```
C:\WINDOWS\py.exe

Choose an option? [1 to 5] - 2

What is the employee Id? - 5
What is the employee First Name? - Ikuya
What is the employee Last Name? - Kirishima

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 1

***** Current Employee List *****
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
5,Ikuya,Kirishima
*****

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] n
Overwrite = No | File not overwritten

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

```
C:\WINDOWS\py.exe

Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] y

*****
Data Saved
*****

*****
Employee Data - Option Menu
*****

1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 1

***** Current Employee List *****
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
5,Ikuya,Kirishima
*****

*****
Employee Data - Option Menu
*****

1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

```
EmployeeData.txt - Notepad
File Edit Format View Help
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
5,Ikuya,Kirishima
Ln 1, Col 100% Windows (CRLF) UTF-8
```



```
C:\WINDOWS\py.exe

Choose an option? [1 to 5] - 3

Enter an employee id to remove: 9

*****
9 is not a valid Employee ID.
*****

*****
Employee Data - Option Menu
*****

1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 3

Enter an employee id to remove: 5

*****
Ikuya Kirishima removed from Employee List.
*****

*****
Employee Data - Option Menu
*****

1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****
```

```
C:\WINDOWS\py.exe
Choose an option? [1 to 5] - 4

Overwrite: EmployeeData.txt? [y/n] y

*****
Data Saved
*****

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

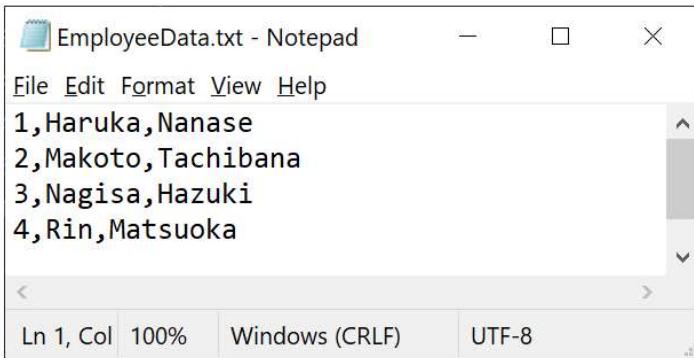
Choose an option? [1 to 5] - 1

***** Current Employee List *****
1,Haruka,Nanase
2,Makoto,Tachibana
3,Nagisa,Hazuki
4,Rin,Matsuoka
*****

*****
Employee Data - Option Menu
*****
1) Show current employees
2) Add a new employee
3) Remove an employee
4) Save employee data to file
5) Exit program
*****

Choose an option? [1 to 5] - 5

Press the enter key to exit.
```

Summary:

Module 08 introduced me to the methodology of Object-oriented programming (OOP) and its basic building block, the software object using a single object. In Module 09, I saw how to expand on that by combining multiple objects together that interact and send messages between them. I saw how to create new classes (Derived/Sub class) based on existing ones (Base/Super class) through the concept of inheritance to define parent-child relationships, extend a derived class by adding new methods and override inherited methods. I learned to write and import my own modules including the “Main” module that runs at the start of the program and how to use UML (Unified Modeling Language) to diagram classes before beginning a project. Assignment 09 demonstrates these concepts in the “Employee List” program. It displays a menu of options to the user that prompts the user to enter an Employee Id, First Name and Last Name. It appends that data to a list table of object rows and reads the data from and writes the data to a delimited text file “EmployeeData.txt”. To complete assignment 09, I added code to each step in the Main.py script at the “TODO” comments and pseudo-code steps, created new or replaced existing code in the DataClasses, ProcessingClasses and IOClasses modules, added error handling to the modules and modified the TestHarness.py script to evaluate the modules.