Week 9 Lab: Employee Management System (Object-Oriented Programming and Security)

Introduction

This is Phase 4, the final phase of the course project. In this lab you transform your Phase 3 program into an object-oriented application with classes, user authentication, and role-based access control and create a professional HR system with security features and the ability to manage user accounts.

What you'll add in Phase 4:

- Employee class to represent employee objects with methods
- **HRUser** class to represent system users with login credentials
- User authentication system (login with username and password)
- Role-based access control (HR Manager vs HR Staff permissions)
- Menu-driven interface based on user role
- User account management (add new users, view users, save/load users)
- Two-file system (**employee_records.txt and hr_users.txt**)

Learning Objectives

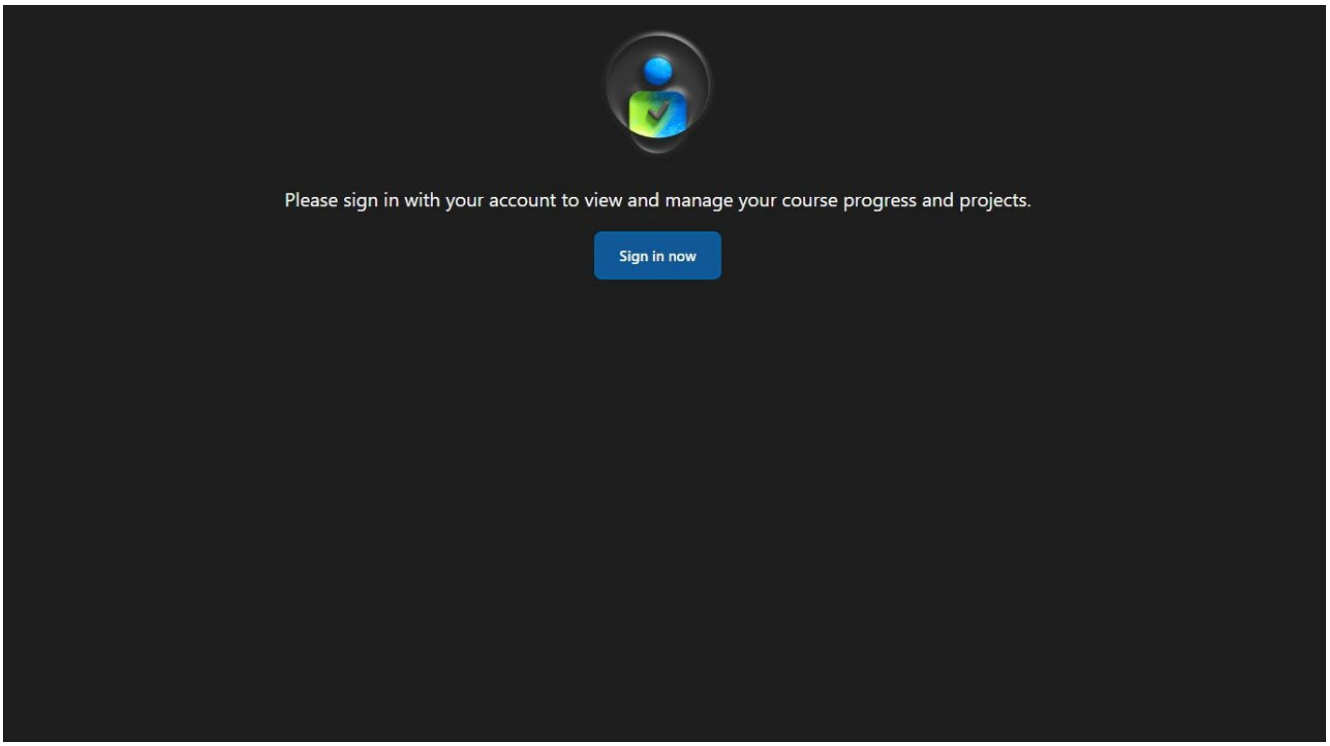
By completing this phase, you will demonstrate your ability to:

- Create classes with **init** constructors
- Define instance methods and attributes
- Use the self keyword correctly
- Create special methods like **str**()
- Instantiate objects from classes
- Implement user authentication systems
- Build role-based access control (RBAC)
- Create menu-driven interfaces
- Work with multiple related classes
- Manage multiple data files
- Add and manage user accounts
- Apply object-oriented programming principles

Lab Instructions

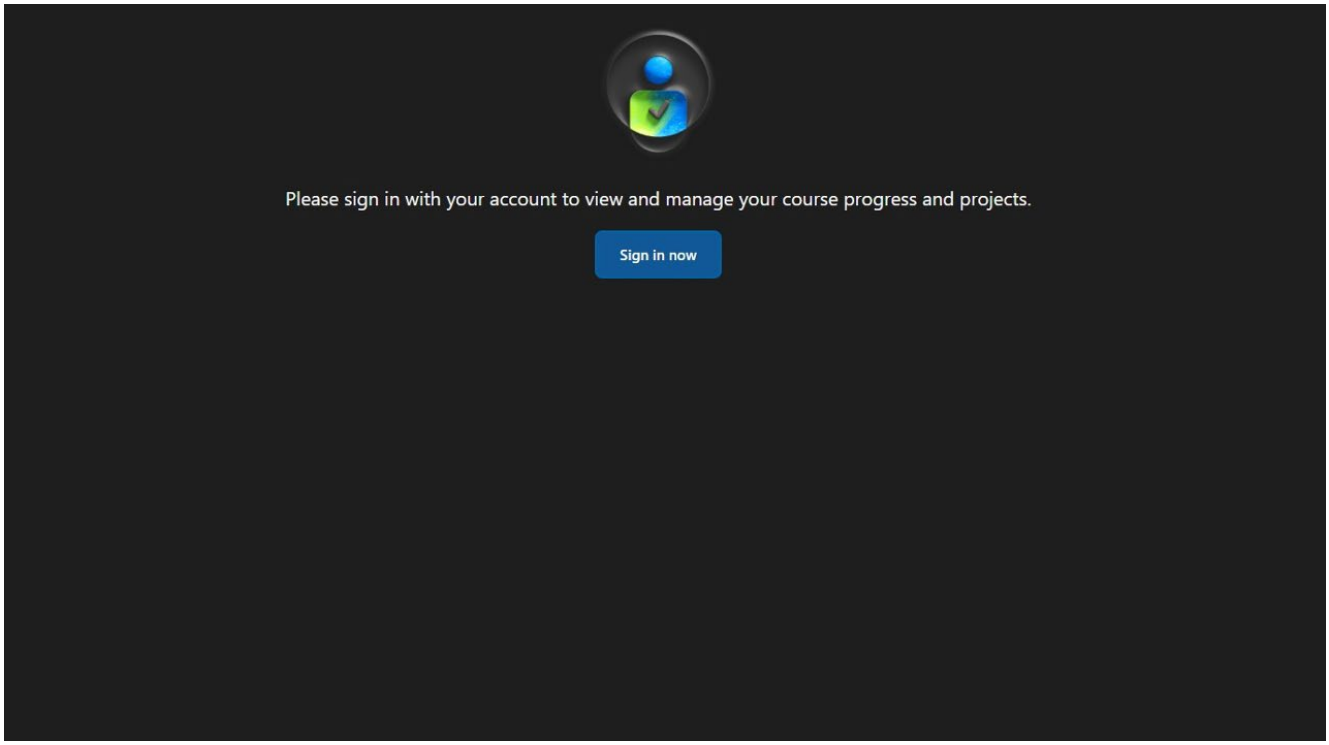
Step 1

Click on https://vscodeedu.com/my-work/projects to access Visual Studio Code for Education.

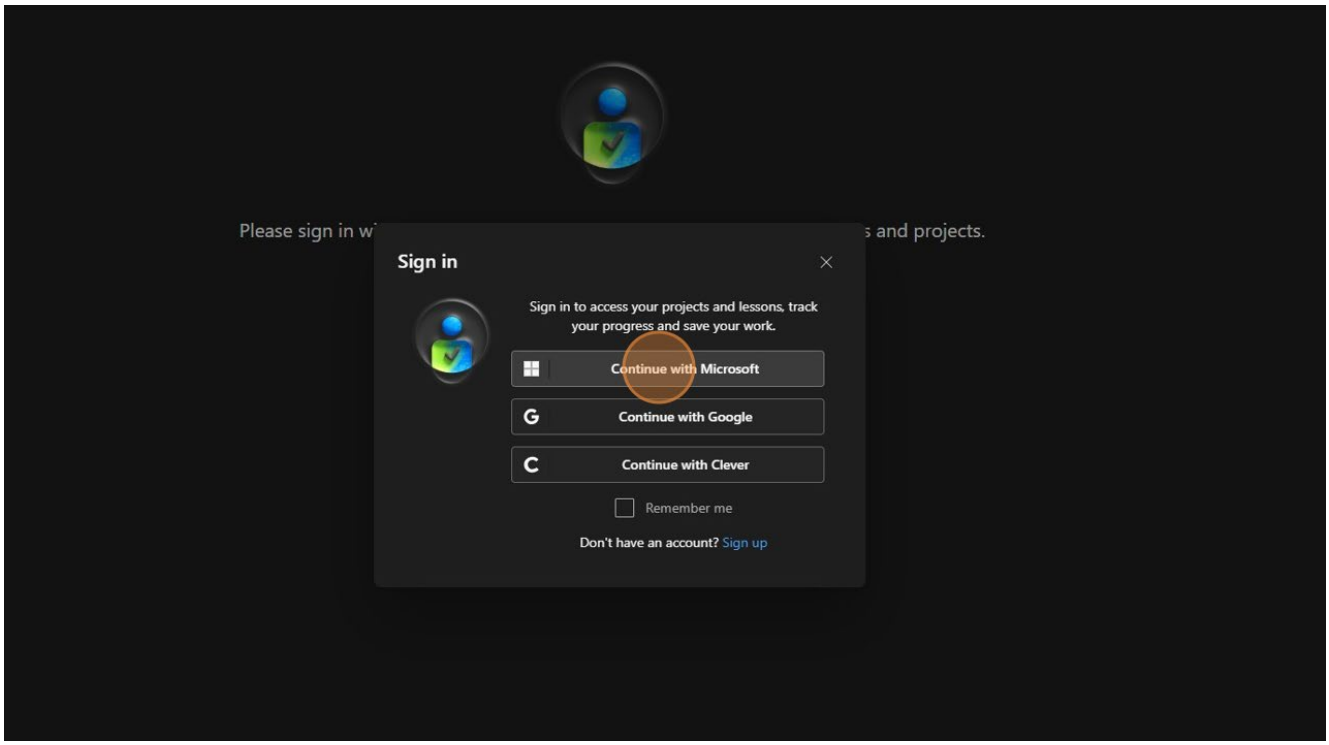


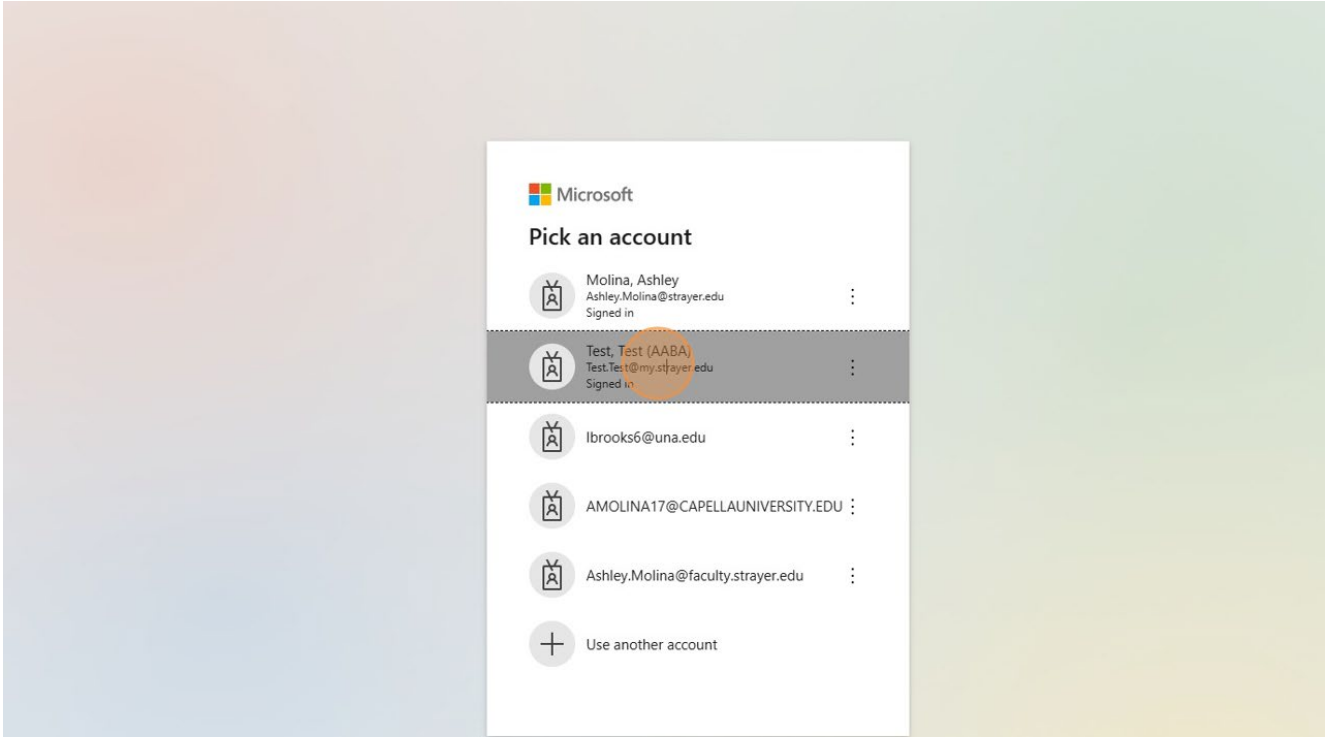
Step 2

You will then Click ‘Sign in now’.

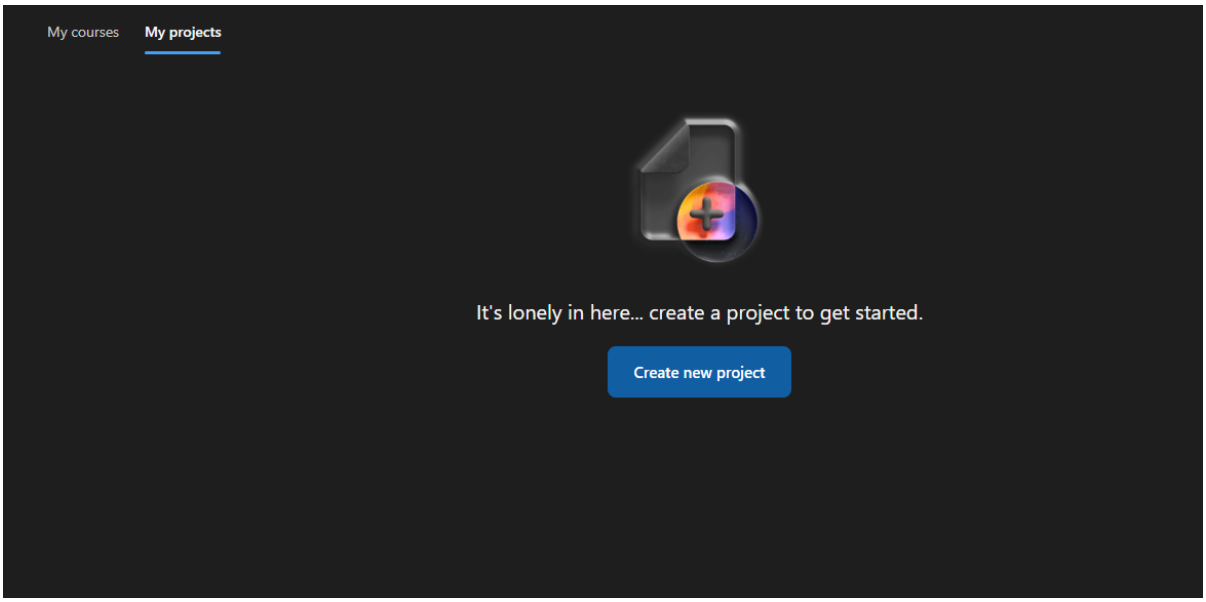


Then select **‘Continue with Microsoft’ or ‘Continue with Google’** to sign in with your already established email address. Using ‘Continue with Microsoft’ you will sign in with your Strayer email address that ends in @my.strayer.edu and your iCampus password. You will be prompted to log into the Strayer Systems as normal.


Please sign in w
s and projects.
Sign in
Sign in to access your projects and lessons, track your progress and save your work.
Continue with Microsoft
Continue with Google
Continue with Clever
Remember me
Don't have an account? Sign up

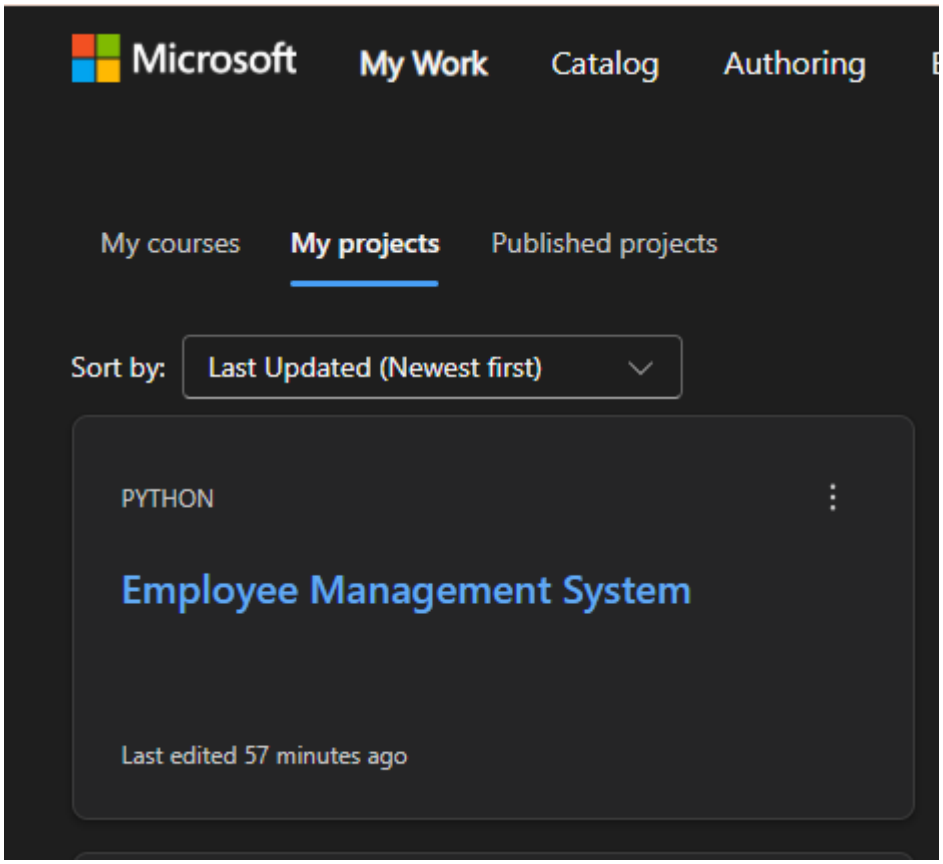

Microsoft
Pick an account
Molina, Ashley
Ashley.Molina@strayer.edu
Signed in
Test, Test (AABA)
Test.Test@my.strayer.edu
Signed in
lbrooks6@una.edu
AMOLINA17@CAPELLAUNIVERSITY.EDU
Ashley.Molina@faculty.strayer.edu
Use another account

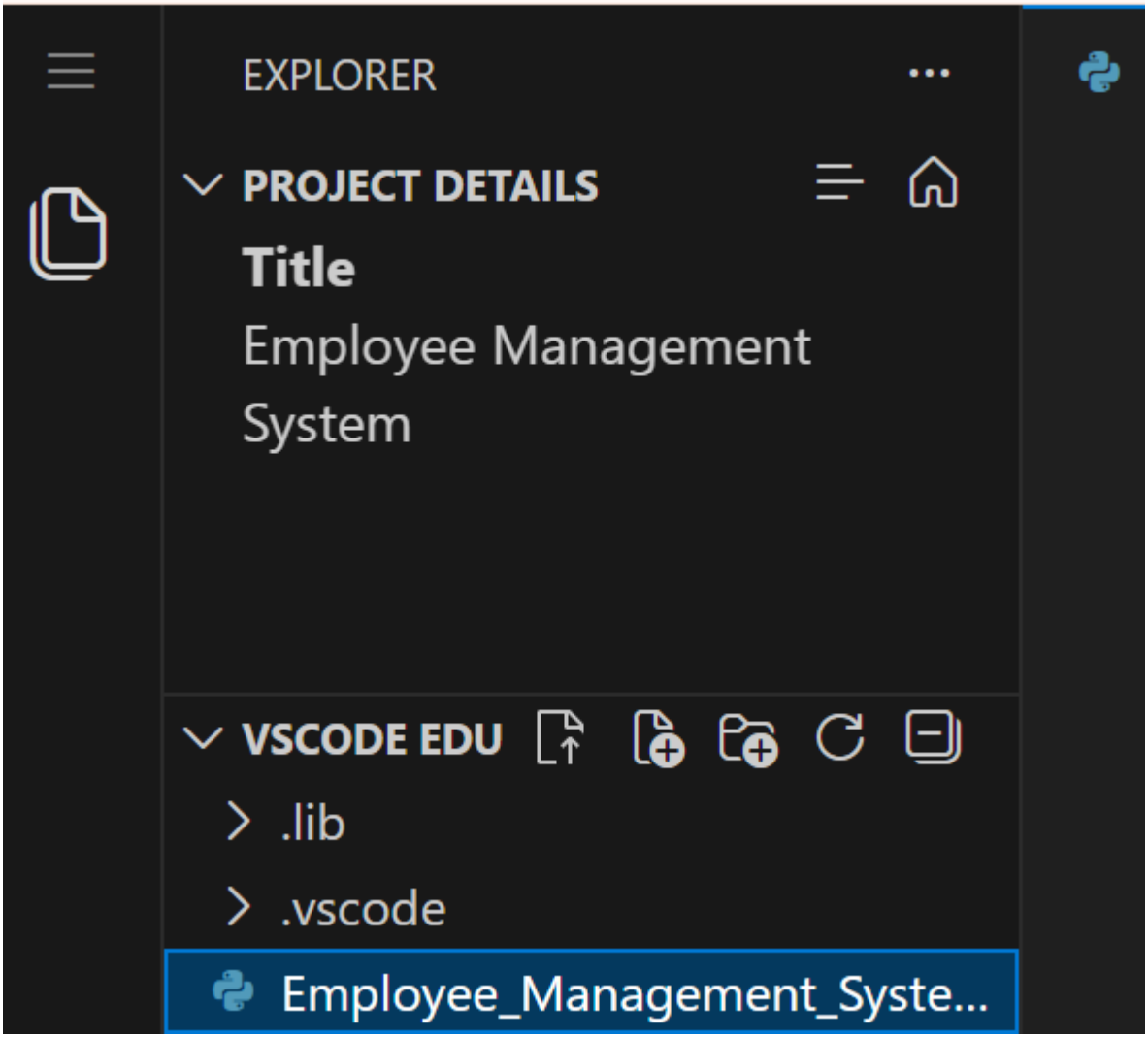
Step 3



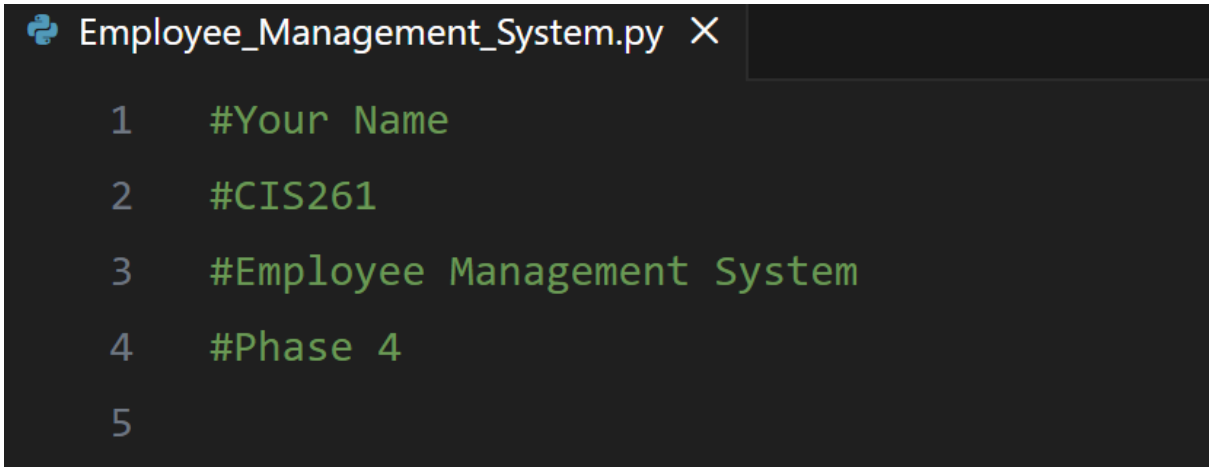
Step 4

Open your previous project named **Employee Management System** and click on the file name Employee_Management_System.py from the left-hand menu once it loads to bring up your previous code.





Step 5
Edit your line 4 comment to #Phase 4.



Step 6
Write a program to perform the following functionality.

Part 1

Create an **Employee** class with the following structure:

init constructor:

- Accept parameters: **first_name, last_name, employee_id, hours, rate, record_date_str=None**
- Use **self** to store all parameters as attributes
- Handle record_date: if None, use **datetime.now()**; if provided, convert string to datetime
- Initialize attributes: **gross_pay**, **deductions**, **net_pay** to 0.0
- Automatically call calculation methods: **self.calculate_gross_pay()**, **self.calculate_deductions(), self.calculate_net_pay()**

Calculation methods:

- **calculate_gross_pay()** - Calculates gross pay with overtime (1.5× for hours > 40), stores in **self.gross_pay**
- **calculate_deductions()** - Calculates 25% of **gross_pay**, stores in **self.deductions**
- **calculate_net_pay()** - Calculates gross_pay - deductions, stores in **self.net_pay**

Helper methods:

- **worked_overtime()** - **Returns True** if **self.hours > 40**, False otherwise
- **is_high_earner(minimum=1000)** - **Returns True** if **self.gross**_pay >= minimum, False otherwise

File and display methods:

- **to_file_string()** - Returns pipe-delimited string for file storage (9 fields)
- **str()** - Returns formatted string for display

Part 2

Create an **HRUser** class with the following structure:

init constructor:

- Accept parameters: username, password, role, full_name
- Use **self** to store all parameters as attributes
- Role should be either "HR Manager" or "HR Staff"

Part 3

Create the following **security functions**:

create_default_users()

- Creates and returns a list with 2 default HRUser objects:
  - HR Manager: **username="hrmanager"**, **password="hr2024", role="HR Manager", full_name="Admin User"**
  - HR Staff: **username="hrstaff", password="staff123", role="HR Staff", full_name="Staff User"**

display_login_screen()

- Displays login prompt with system title

authenticate_user(users, username, password)

- Loops through users list
- Compares entered username and password with each user
- Returns the HRUser object if match found, None if not found

display_access_granted(username, role)

- Displays success message with username and role
- Shows permissions based on role

display_access_denied()

- Displays login failure message

Part 4

Create the following **user management functions**:

add_new_user(users)

- Prompts for: username, password, role (HR Manager or HR Staff), full_name
- Creates new HRUser object
- Appends to users list
- Displays confirmation message

display_all_users(users)

- Displays all users WITHOUT showing passwords
- Shows: username, role, full_name for each user

save_users_to_file(users, filename)

- Writes users to file in pipe-delimited format: username|password|role|full_name
- Uses try/except for error handling

load_users_from_file(filename)

- Reads users from file
- Creates HRUser objects from parsed data
- Returns list of HRUser objects
- Handles **FileNotFoundError** (creates default users if file doesn't exist)

display_user_management_menu()

- Displays submenu: **1. Add New User, 2. View All Users, 3. Return to Main Menu**

manage_user_accounts(users, users_filename)

- Shows user management menu
- Routes to **add_new_user()** or **display_all_users()**
- Saves users to file after changes
- Returns to main menu when done

Part 5

Create **menu functions**:

display_menu(role)

- Shows different menu options based on role:
  - **HR Manager menu:** 1. Process Employee Records, 2. View Reports, 3. Manage User Accounts, 4. Exit
  - **HR Staff menu:** 1. View Reports, 2. Exit

get_menu_choice()

- Prompts for and returns menu choice

Part 6

Create **menu handler functions**:

process_employee_records(filename)

- Prompts for employee information
- Creates Employee objects (not dictionaries!)
- Appends to list
- Saves to file using object's **to_file_string()** method
- Only accessible to HR Manager

view_reports(filename)

- Loads employees from file (creates Employee objects)
- Calls **generate_report()** to display comprehensive report
- Accessible to both HR Manager and HR Staff

generate_report(employee_records)

- Displays all employee records using object attributes
- Calculates and displays statistics
- Shows count of overtime employees using **.worked_overtime()** method
- Shows count of high earners using **.is_high_earner()** method

Step 7

Modify all file I/O functions to work with Employee objects:

save_records_to_file(employee_records, filename)

- Loops through list of Employee objects
- Calls **employee.to_file_string()** to get file line
- Writes to file

load_records_from_file(filename)

- Reads from file
- Splits by pipe
- Creates Employee objects (not dictionaries!)
- Returns list of Employee objects

Step 8

Create the **main()** function with complete workflow:

1. Set filenames

- **employees_filename = "employee_records.txt"**
- **users_filename = "hr_users.txt"**

2. Load users

- Call **load_users_from_file(users_filename)**

3. Display login screen

- Call **display_login_screen()**
- Prompt for username and password

4. Authenticate

- Call **authenticate_user(users, username, password)**
- If None returned, call **display_access_denied()** and exit program

5. Display access granted

- Call **display_access_granted(username, role)**

6. Menu loop

- Display menu based on **current_user.role**
- Get menu choice
- Route to appropriate function:
    - **HR Manager Option 1: process_employee_records()**
    - **HR Manager Option 2: view_reports()**
    - **HR Manager Option 3: manage_user_accounts()**
    - **HR Manager Option 4: Exit**
    - **HR Staff Option 1: view_reports()**
    - **HR Staff Option 2: Exit**

7. Save users before exit

- Call **save_users_to_file(users, users_filename)**

8. Call **main()** to run program

NOTE: *HR Staff can ONLY view reports. HR Manager has full access including user management.*

Step 9

Testing Requirements

Test your program with a minimum of **2 employees** across **multiple runs** to demonstrate all features:

Test 1 - HR Manager Login:

- Login with username="hrmanager", password="hr2024"
- Verify full menu displays (4 options)
- Process 2 employee records
- View reports
- Manage user accounts (add at least 1 new user)
- Exit

Test 2 - HR Staff Login:

- Login with username="hrstaff", password="staff123"
- Verify limited menu displays (2 options only)
- View reports (should see data from Test 1)
- Verify CANNOT access Process Records or Manage Users
- Exit

Test 3 - Failed Login:

- Try logging in with incorrect username/password
- Verify access denied message appears
- Program exits

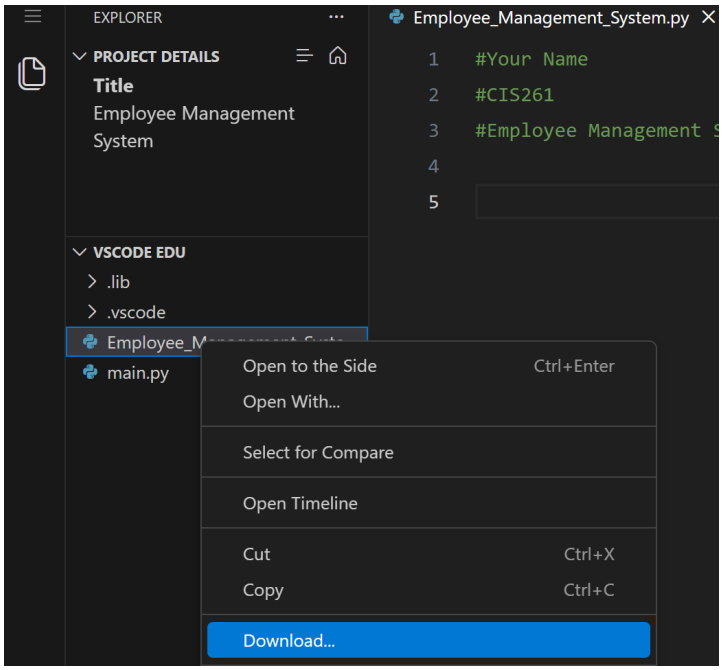
Test 4 - Data Persistence:

- Run program again
- Verify employee data persists
- Verify user accounts persist (including newly added user from Test 1)

Saving Your Work to Your Computer

Now, we are going to download our python file for submission!

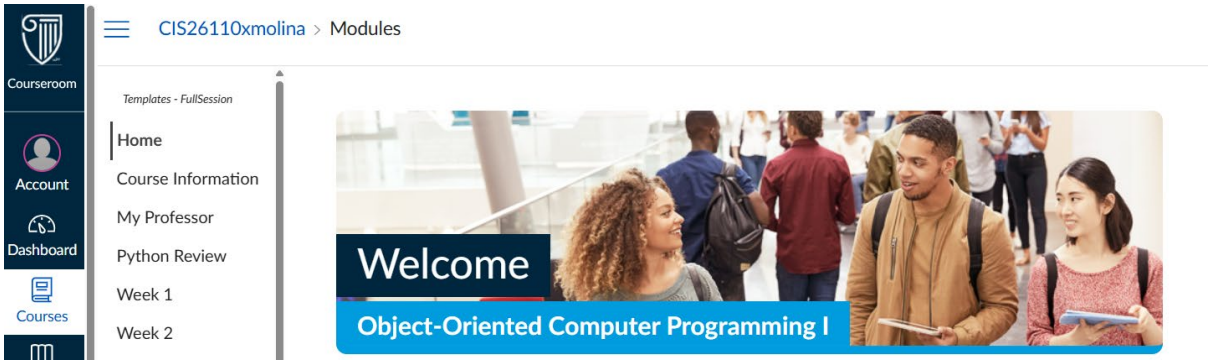
We need to download your **Employee_Management_System.py** file to your computer to prepare your submission. To do this right-click on **Employee_Management_System.py** and a pop-up box will appear. Select **Download** from the menu, to download the file to your downloads folder on your computer.



*If this is the first time in here you may get a pop-up that asks '**vscode.dev wants to See text and images copied to the clipboard Block or Allow**' click **Allow**.* Once you 'Allow' you will then see your menu to download your file.

Now your python file for this project is in your computer's downloads folder!

Go to https://canvas.strayer.edu/ in your web browser, sign-in, and select **CIS261 Object-Oriented Programming** from your tiles.



Step 3

Select Week 9 – Lab from the Week 9 menu options

For this lab you should submit your **CIS261_Employee_Management_System.py** directly to the **Week 9 – Lab** submission area.

Step 4

Select the **Upload** option and then select **Choose a file to upload** from the submission area and select **CIS261_Employee_Management_System.py** from your selected location in your File Explorer.

When you select the file you will click **Open** to add the file to Canvas.

Once your file shows a green check mark beside it click "***I agree to the tool's End-User License Agreement. This assignment submission is my own, original work"*** and click "**Submit Assignment"**

Review

Congratulations! You have completed all 4 phases of the Employee Management System! You've built a complete, professional HR application with:

- Object-oriented design
- User authentication
- Role-based security
- User account management
- Data persistence
- Menu-driven interface

This project demonstrates your competency in all major Python programming concepts!

Important Notes

Employee Class:

- Must have **init** constructor with all required parameters
- Must use **self** for all attributes
- Must automatically calculate pay in **init**
- Must have all required methods (calculate methods, helper methods, file/display methods)

HRUser Class:

- Must have **init** with username, password, role, full_name
- Must use **self** for all attributes

Security:

- Must have default users: **hrmanager** and **hrstaff**
- Must authenticate before allowing access
- Must deny access if login fails
- Must exit program if authentication fails

Role-Based Access:

- HR Manager: Full access (4 menu options)
- HR Staff: Read-only (2 menu options - can ONLY view reports)
- Menu must display different options based on role

User Management:

- Must be able to add new users
- Must display all users (without passwords)
- Must save users to **hr_users.txt**
- Must load users from hr_**users.txt**

- Must demonstrate adding at least 1 new user

Files:

- Must use TWO files: **employee_records.txt and hr_users.txt**
- Both files must use pipe-delimited format
- Must load users at program start
- Must save users before program exit

Objects:

- Must create Employee objects (not dictionaries)
- Must use object methods (**employee.worked_overtime(), employee.is_high_earner()**)
- Must use object attributes (**employee.first_name, employee.gross_pay**)
- Must call **employee.to_file_string()** for file output

END OF DOCUMENT