

Employee Directory

Esther Faith Dennis _ T1A3

Using the Feature

Employee Directory - Four Features

- Create Employee
- Search Employee
- Update Employee
- Delete Employee

```
Welcome to the Employee Directory
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

Create New Employee

- Enter 1 to create a new employee
- It will print a message to show the user the format to enter employee data.
- User can enter the employee data
- Press enter to submit
- The program will print a success or fail message to the terminal

```
Enter your command: 1
```

```
Enter the new employee data in this format
```

```
<employee-id> <first-name> <last-name> <phone-number> <job-title> <salary> <start-date>
```

```
Enter the new employee data: █
```

```
Enter the new employee data: 1001 Esther Dennis 04000500 chef 80000 01/01/20
```

```
New employee created: 1001 Esther Dennis 04000500 chef 80000 01/01/20
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

Create New Employee

- If the employee already exists, it will print an error message.
- The create command also validates all the inputs, if one field is wrong, it will print an error message.
- The program will keep asking the user for a new command until they exit (press 5)

```
Enter your command: 1

Enter the new employee data in this format
    <employee-id> <first-name> <last-name> <phone-number> <job-title> <salary> <start-
Enter the new employee data: 1001 Emma Jane 04000045 manager 90000 1/3/21

[ERROR]: The employee id already exist, please input a valid employee id.

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

```
Enter your command: 1

Enter the new employee data in this format
    <employee-id> <first-name> <last-name> <phone-number> <job-title> <salary> <start-date>
Enter the new employee data: 1004 2 Dennis 04004443 barista 65000 01/01/20

[ERROR]: The first name is invalid, please input a valid name.

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

Search Employee

- To search for an employee enter the command 2.
- It will have some instruction to tell you which fields do you want to search the employee for.
- For example --first-name Esther
- It will print all the employees with first name, Esther.

```
Enter your command: 2
```

```
Enter the search fields exactly like left column and space follow by the contents you want to find, leave blank if you want to see all users
```

```
--employee-id    <employee-id>
--first-name     <first-name>
--last-name      <last-name>
--phone-number   <phone-number>
--job-title      <job-title>
--max-salary     <max-salary>
--min-salary     <min-salary>
--max-start-date <max-start-date>
--min-start-date <min-start-date>
```

```
Enter the employee search fields: 
```

```
Enter the employee search fields: --first-name Esther
```

```
1001 Esther Dennis 04000500 chef 80000 01/01/20
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: 
```

Search Employee

- If no employees match the searched fields. It will print a message saying no employees match your search.
- If more than one employee meet the search fields, it will print all employees that match the search
- In this case 2 employees have the same last name.

```
Enter the employee search fields: --employee-id 1234
```

```
No employees match your search
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

```
Enter the employee search fields: --last-name Jones
```

```
1002 Emma Jones 05300030 waitress 65000 03/5/22
```

```
1003 Charlie Jones 03003004 manager 89000 1/2/20
```

Search Employee

- The user can also use max salary field to search the maximum salary. It will print all the employee salary under or equal the maximum salary.
- If the the user use the min salary, it will print all the employee salary greater or equal the minimum salary.
- Also user can use the max or min start date to search the employee.

```
Enter the employee search fields: --max-salary 89000

1001 Esther Dennis 04000500 chef 85000 01/02/19
1003 Charlie Jones 03003004 manager 89000 1/2/20

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

```
Enter the employee search fields: --min-salary 89000

1003 Charlie Jones 03003004 manager 89000 1/2/20

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

```
Enter the employee search fields: --max-start-date 1/02/21

1001 Esther Dennis 04000500 chef 85000 01/02/19
1003 Charlie Jones 03003004 manager 89000 1/2/20

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```


Search Employee

- The user can use multiple fields to search at the same time.
- For example, min salary and min start date together to search the employee. It will print out the employee which has minimum salary and minimum start date .

```
Enter the employee search fields: --min-salary 60000 --min-start-date 1/01/21

1002 Jesse Dennis 03880300 barista 70000 02/04/23
1004 Joe James 04220589 waitress 60000 01/03/22

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

```
Enter the employee search fields: --max-start-date 1/1/21 --min-salary 65000

1001 Esther Dennis 04000500 chef 85000 01/02/19
1003 Charlie Jones 03003004 manager 89000 1/2/20

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

Search Employee

- If you want to see all the employees, you can just enter zero fields.
- This will list all the employees.

Enter the employee search fields:

```
1001 Esther Dennis 04000500 chef 80000 01/01/20
1002 Emma Jones 05300030 waitress 65000 03/5/22
1003 Charlie Jones 03003004 manager 89000 1/2/20
```

Choose from one of these commands

Enter 1 to creates a new employee

Enter 2 to search for an employee

Enter 3 to update the data for one employee

Enter 4 to delete one employee

Enter 5 to exit

Enter your command: █

Search Employee

- Search employee also validates the input.
- If you input a wrong number of arguments, a wrong field name, or a bad data, it will print an error.

```
Enter the employee search fields: --employee id 1001
```

```
[ERROR]: Not a valid number of inputs.
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

```
Enter the employee search fields: --fist-name Esther
```

```
[ERROR]: Invalid field is given.
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

Update Employee

- Use command 3 to update an employee.
- It will show you instruction how to update the employee data.
- Type the employee id to update, then the field and data you want to update.
- For example 1001 --salary 85000
- It will print out to show the updated user.

```
Enter your command: 3
```

```
Enter the employee id followed by fields you want to update
```

```
<employee-id> --first-name      <first-name>
                --last-name     <last-name>
                --phone-number   <phone-number>
                --job-title      <job-title>
                --salary         <salary>
                --start-date     <start-date>
```

```
Enter the employee id followed by fields to be updated: █
```

```
Enter the employee id followed by fields to be updated: 1001 --salary 85000
```

```
Employee updated: 1001 Esther Dennis 04000500 chef 85000 01/01/20
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

Update Employee

- Update can also update more than one field.
- For example
 - 1001 --job-title cook --salary 80000

```
Enter the employee id followed by fields to be updated: 1001 --job-title cook --salary 80000
```

```
Employee updated: 1001 Esther Dennis 04000500 cook 80000 01/02/19
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
```

```
Enter 2 to search for an employee
```

```
Enter 3 to update the data for one employee
```

```
Enter 4 to delete one employee
```

```
Enter 5 to exit
```

```
Enter your command: █
```

Update Employee

- The update feature also validate the input.
- If the user enter wrong, it will print an error message
- For example, --startdate 0/2
- For example, --start-date date

```
Enter the employee id followed by fields you want to update
```

```
<employee-id> --first-name    <first-name>
                --last-name    <last-name>
                --phone-number  <phone-number>
                --job-title     <job-title>
                --salary        <salary>
                --start-date    <start-date>
```

```
Enter the employee id followed by fields to be updated: 1001 --startdate 0/2
```

```
[ERROR]: Invalid field is given.
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit
```

```
Enter your command: █
```

```
Enter the employee id followed by fields to be updated: 1001 --start-date date
```

```
[ERROR]: start-date is not valid.
```

```
Choose from one of these commands
```

```
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit
```

```
Enter your command: █
```

Delete Employee

- Enter 4 to delete one employee
- After you enter 4, it will print a message to ask you to enter the employee id to be deleted.
- For example, enter 1002, then it will delete immediately
- It also will have a message to show you employee is deleted and the deleted employee data.

```
Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit
```

```
Enter your command: 4
```

```
Enter the employee id to be deleted: █
```

```
Enter the employee id to be deleted: 1002
```

```
Employee deleted: 1002 Emma Jones 05300030 waitress 65000 03/5/22
```

```
Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit
```

```
Enter your command: █
```

Delete Employee

- Delete also validate the input and print an error message if the user enter wrong
- For example, if the employee-id does not exist, it prints an error message and not delete
- For example if user enter a invalid employee-id it prints an error message

```
Enter your command: 4

Enter the employee id to be deleted: 11

[ERROR]: Delete employee can not found, please input the existing employee id to delete.

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```

```
Enter the employee id to be deleted: 1d02

[ERROR]: The employee id should be a digital number, please input a correct employee to delete.

Choose from one of these commands
Enter 1 to creates a new employee
Enter 2 to search for an employee
Enter 3 to update the data for one employee
Enter 4 to delete one employee
Enter 5 to exit

Enter your command: █
```


Code Design

Choose a Feature

- Program runs in while loop so user can keep doing commands
- When user picks a command, it then check which command and print out more instruction and get more inputs
- When it get the extra inputs it validates the inputs
- Then it chooses which feature to run
- Each feature has own function

```
while(True):
    print("")
    print("Choose from one of these commands")
    print(" Enter 1 to creates a new employee")
    print(" Enter 2 to search for an employee")
    print(" Enter 3 to update the data for one employee")
    print(" Enter 4 to delete one employee")
    print(" Enter 5 to exit")
    print("")
    command = input("Enter your command: ")
    command = command.replace(" ", " ")

    print("")
    user_inputs = None
    if command == "1":
        print("Enter the new employee data in this format")
        print(" <employee-id> <first-name> <last-name> <phone-number> <job-title> <salary> <start-date>")
        print("")
        user_inputs = input("Enter the new employee data: ")
        print("")
```

```
# check if valid input
if not valid_inputs(command, inputs):
    continue

# create new_employee to a file
if command == "1":
    employee_data = inputs
    create_employee(filename, employee_data)
elif command == "2":
    search_data = inputs
    search_employee(filename, search_data)
elif command == "3":
    update_employee = inputs
    update_data(filename, update_employee)
elif command == "4":
    delete_employee = inputs[0]
    delete_data(filename, delete_employee)
```

Validate Inputs

- When user gives inputs to validate_inputs it runs differently for each command.
- It checks if inputs are right length, then checks each input if correct value
- Because most features need to check the same inputs, it used a function to check each input. This helped to make code DRY (don't repeat yourself).

```
def valid_inputs(command, inputs):  
    if command == "1":  
        return valid_create_inputs(inputs)  
    elif command == "2":  
        return valid_search_inputs(inputs)  
    elif command == "3":  
        return valid_update_inputs(inputs)  
    elif command == "4":  
        return valid_delete_inputs(inputs)
```

```
# valid command  
def valid_create_inputs(inputs):  
    # validate the number of input  
    if len(inputs) != 7:  
        print_error("Incorrect number of arguments given, please input format with space separate as create e  
        return False  
  
    # validate_ employee id - value  
    if not valid_number(inputs[0]):  
        print_error("The employee id should be a digital number.")  
        return False  
  
    # validate first name as alpha input  
    if not valid_name(inputs[1]):  
        print_error("The first name is invalid, please input a valid name.")  
        return False
```

```
# validate whether is number function  
def valid_number(num_as_string):  
    if num_as_string.isdigit():  
        return True  
    return False  
  
# validate whether name is alpha function  
def valid_name(name):  
    if name.isalpha():  
        return True  
    return False
```

Validate Error Message

- When the user enters wrong data for a field, it will print an error message.
- Every feature will validate the input and print a description error message if the user inputs a wrong value.

```
print("")
elif command == "5":
    return
else:
    print_error("Program needs a valid command.")
    continue
```

```
if field == "--employee-id":
    if (not valid_number(value)):
        print_error("employee-id is not valid.")
        return False
elif field == "--first-name":
    if (not valid_name(value)):
        print_error("first-name is not valid.")
        return False
elif field == "--last-name":
    if (not valid_name(value)):
        print_error("last-name is not valid.")
        return False
elif field == "--phone-number":
    if (not valid_number(value)):
        print_error("phone-number is not valid.")
        return False
elif field == "--job-title ":
    if (not valid_name(value)):
        print_error("job-title is not valid.")
        return False
elif field == "--max-salary":
    if (not valid_number(value)):
        print_error("max-salary is not valid.")
        return False
elif field == "--min-salary":
    if (not valid_number(value)):
        print_error("min-salary is not valid.")
        return False
elif field == "--max-start-date":
```

Create Employee Feature

- The create function will read the employee data and check if the employee id already exists.
- If so it will print an error message.
- Once id checked and is available to use, the app creates a new Employee object with the input data
- The new employee is appended to the list of employees.
- The whole list of employees is written back to the csv file.

```
# create employee function
def create_employee(filename, employee_data):
    id = employee_data[0]
    # check the employee id whether exist
    employees = read_employees_csv(filename)

    for employee in employees:
        if employee.employee_id == id:
            print_error("The employee id already exist, please input a valid employee id.")
            return False

    new_employee = Employee(employee_data[0], employee_data[1], employee_data[2], employee_data[3], employee_data[4], employee_data[5], employee_data[6])
    employees.append(new_employee)
    print(f"New employee created: {new_employee}")
    write_employees_csv(filename, employees)
    return True
```

```
# to read the employees
def read_employees_csv(filename):
    try:
        employees = []
        with open(filename, "r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                employees.append(
                    Employee(
                        row["employee_id"],
                        row["first_name"],
                        row["last_name"],
                        row["phone_number"],
                        row["job_title"],
                        row["salary"],
                        row["start_date"]
                    )
                )
        return(employees)
    except FileNotFoundError:
        return []
```

```
class Employee:
    # Constructor
    def __init__(self, employee_id, first_name, last_name, phone_number, job_title, salary, start_date):
        self.employee_id = employee_id
        self.first_name = first_name
        self.last_name = last_name
        self.phone_number = phone_number
        self.job_title = job_title
        self.salary = int(salary)
        self.start_date = start_date

    # print data from employee
    def __str__(self):
        return str(self.employee_id) + " " + \
            self.first_name + " " + \
            self.last_name + " " + \
            self.phone_number + " " + \
            self.job_title + " " + \
            str(self.salary) + " " + \
            self.start_date
```

Csv Helpers

- Since most of the feature need to read and write from the csv file. I created some helper functions to do the read and write.
- Write employee take a list of employee objects and writes them to the csv file.
- Read employee opens the file and read all the users in the file
- Working with a list of Employee objects was easier than always changing the file
- Could easy to just read the employees when I need them, change them, and easy to write them back to the file.

```
# write employee to file function
def write_employees_csv(filename, employees):

    header = ["employee_id", "first_name", "last_name", "phone_number", "job_title", "salary", "start_date"]
    employee_data = []

    for employee in employees:
        employee_data.append(employee.get_as_list())

    with open (filename, "w") as file:
        writer = csv.writer(file)
        writer.writerow(header)
        writer.writerows(employee_data)
```

```
# to read the employees
def read_employees_csv(filename):
    try:
        employees = []
        with open(filename, "r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                employees.append(
                    Employee(
                        row["employee_id"],
                        row["first_name"],
                        row["last_name"],
                        row["phone_number"],
                        row["job_title"],
                        row["salary"],
                        row["start_date"]
                    )
                )
    except:
        return (employees)
```

Other Features

- search_employee function for the search feature
- update_data function for the update feature
- delete_data function for the delete feature

```
# search function
def search_employee(filename, search_data):
    employees = read_employees_csv(filename)

    match_employees = []
    for employee in employees:
        is_match = True
        for i in range(0, len(search_data), 2):
            field = search_data[i]
            value = search_data[i+1]

            if field == '--employee-id' and employee.employee_id != value:
                is_match = False
                break

            if field == '--first-name' and employee.first_name != value:
                is_match = False
                break
```

```
#update employee data
def update_data(filename, update_employee):
    id = update_employee[0]
    updateData = update_employee[1:]

    employees = read_employees_csv(filename)

    employee = None

    for emp in employees:
        if emp.employee_id == id:
            employee = emp

    if employee == None:
        print_error("Update employee can not found, please input the correct employee id.")
        return False

    for i in range(0, len(updateData), 2):
        field = updateData[i]
        value = updateData[i+1]
        if field == '--first-name':
            employee.first_name = value
        elif field == '--last-name':
```

```
# delete employee
def delete_data(filename, delete_employee):
    id = delete_employee

    employees = read_employees_csv(filename)

    employee = None
    for emp in employees:
        if emp.employee_id == id:
            employee = emp

    if employee == None:
        print_error("Delete employee can not found, please input the existing employee id to delete.")
        return False

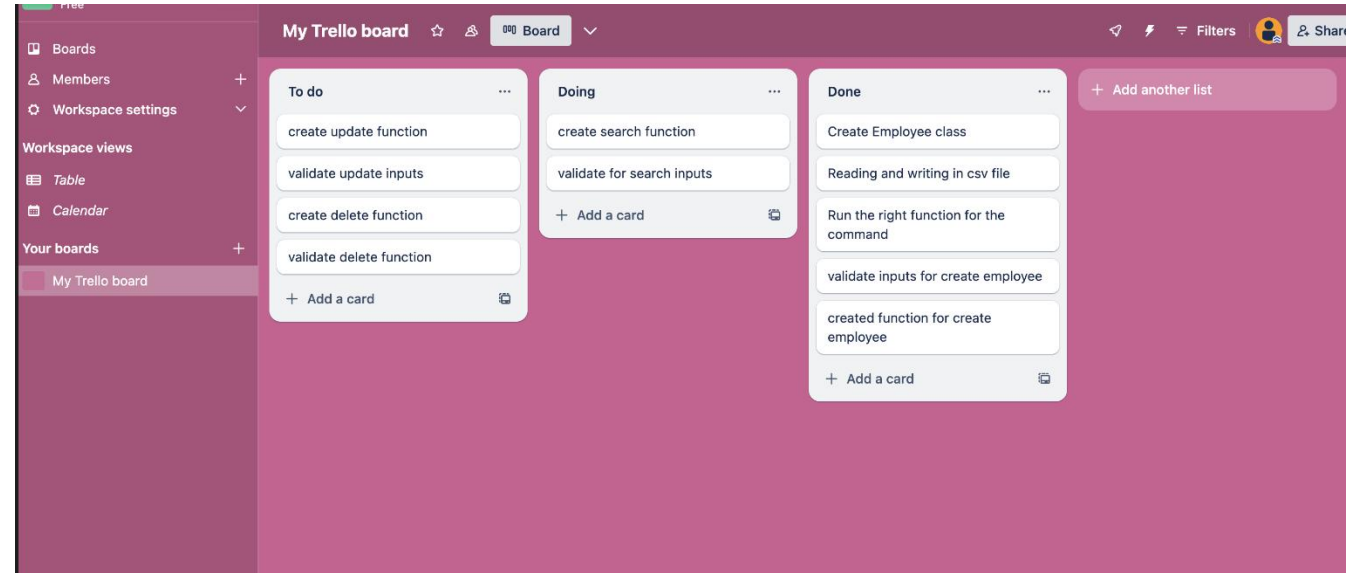
    print(f"Employee deleted: {employee}")
    employees.remove(employee)

    write_employees_csv(filename, employees)
    return True
```

Project Manegement

Project Management

- I have use Trello to help me to manage and keep on track what is not done, what am I doing, and what I have been done.



Code Styling

- Follow the PEP 8 python style guide
- Use autopep8 to auto style make it more efficiency.
- Consistency.
- Enhance code readability.

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>

Status: Active

Type: Process

Created: 05-Jul-2001

Post-History: 05-Jul-2001, 01-Aug-2013

Project description

autopep8 automatically formats Python code to conform to the [PEP 8](#) style guide. It uses the [pycodestyle](#) utility to determine what parts of the code needs to be formatted. autopep8 is capable of fixing most of the formatting [issues](#) that can be reported by pycodestyle.

Contents

- [Installation](#)

Code test - unittest

- Used unittest to set up automated testing.
- Test the create feature
- Test the search feature
- Confident after making changes that the app still works correctly.

```
import unittest
import os
from unittest.mock import patch
from main import run_app

test_filename = "test_storage.csv"

def check_csv(expected_employees):
    file = open(test_filename, "r")

    header = file.readline()
```

```
class Test_inputs(unittest.TestCase):
    @patch('builtins.input')
    def test_create_new_employee(self, mocked_input):
        mocked_input.side_effect = [
            "1", # create command
            "1234 Esther Dennis 0455111222 Barista 65000 01/01/23",
            "5" # exit
        ]
        run_app(filename=test_filename)

        result = check_csv([
            "1234,Esther,Dennis,0455111222,Barista,65000,01/01/23"
        ])
        os.remove(test_filename)
        self.assertTrue(result)
```

```
class Test_inputs(unittest.TestCase):
    @patch('builtins.input')
    def test_search_employee_id(self, mocked_input):
        create_users()
        mocked_input.side_effect = [
            "2", # search command
            "--employee-id 1001",
            "5" # exit
        ]

        saved_stdout = sys.stdout


        out = io.StringIO()
        sys.stdout = out
```

Script

- A script for run the code
- A script for run the code styling
- A script for run the test feature

```
coder-academy > EstherFaithDennis_T1A3 > src >  run_app.sh
1  #!/bin/bash
2
3  python3 -m venv .venv
4  source .venv/bin/activate
5  pip3 install --disable-pip-version-check --quiet -r requirements.txt
6  python3 main.py
```

```
coder-academy > EstherFaithDennis_T1A3 > src >  run_format.sh
1  #!/bin/bash
2
3  python3 -m venv .venv
4  source .venv/bin/activate
5  pip3 install --disable-pip-version-check --quiet -r requirements.txt
6  autopep8 --in-place --aggressive --aggressive *.py
7
```

```
coder-academy > EstherFaithDennis_T1A3 > src >  run_tests.sh
1  #!/bin/bash
2
3  verbose=$1
4
5  if [ $verbose = "-v" ]
6  then
7      python3 test_create_feature.py
8      python3 test_search_feature.py
9  else
10     python3 test_create_feature.py -b
11     python3 test_search_feature.py -b
12 fi
```

Development Process

- The development process was fairly simple once I knew my app idea
- Decide on an interface for the user to use the app
- Start by writing helper code like csv read and write
- Then create some validate function for validating inputs
- Then for each feature I worked on the validation first and then the logic for the feature
- This was easy to follow because it was all listed in the trello board.

Challenges - Validate and Testing

- Validating the input correctly
 - Users can input anything they want
 - The app needs a very specific format
 - The app needed a lot of inputs
- Testing
 - Writing a test is very new to me
 - Testing an app that needs user input was hard
 - Took a long time to get user input working in the tests
 - First time I tested I used the same csv filename and kept breaking my data for the app

Ethical Issues - User Data

- User data should be secure more carefully
 - Easy to access the CSV file with sensitive data
 - Should use more secure storage
 - Should use password access to the data

Favourite Part - CSV Helper

- I enjoyed making the CSV helper code
- Easy to read and write employee from csv
- Made it easier write the code for each feature
- Helped make the code DRY (don't repeat yourself)
- Working with list of employee easy compared to always read and write from csv.