TERMINAL APPLICATION

By Laylah De Paull

GCAS022002

Presentation Link

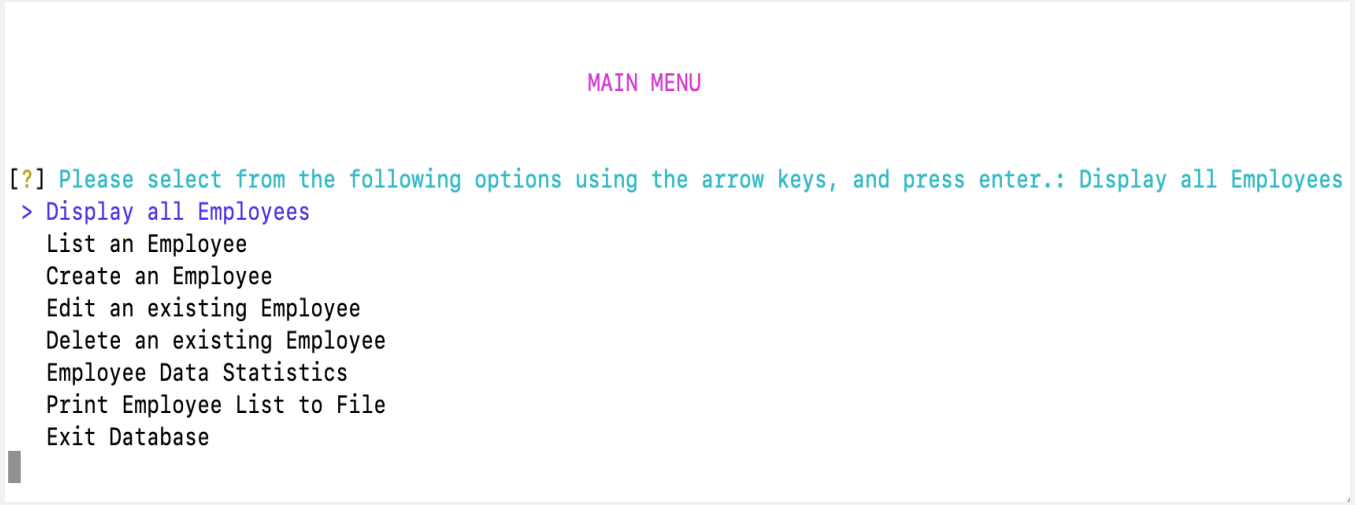
FEATURE WALKTHROUGH – OVERVIEW OF APPLICATION

```
Hello there, to log in to the: 'Initech' Employee Database Application, please enter your name?
Jane Citizen
It's nice to hear from you Jane Citizen, you are now logged in to the: 'Initech' Employee Database Applica
tion.

Logged In: Jane Citizen
Date: 2023-02-12
```

```
Logged In: Jane Citizen
Date: 2023-02-11

      Hello Jane Citizen Welcome to the 'Initech' Employee Database Application.

                                        MAIN MENU

[?] Please select from the following options using the arrow keys, and press enter.: Display all Employees
 > Display all Employees
   List an Employee
   Create an Employee
   Edit an existing Employee
   Delete an existing Employee
   Employee Data Statistics
   Print Employee List to File
   Exit Database
```

- My Terminal Application is called: "*The 'Initech' Employee Database Application*".
- My App basically functions as a '*Database*' that stores various current employee data, that includes an employee's: *name*, *age*, *role*, & *salary*.
- My App was designed to be used by a: HR Manager or Employment Manager of sorts, to access current company employee data.
- To log in a HR or Employment Manager must first enter their name.
- A HR or Employment Manager when 'logged into' the 'database' can: view, alter, print, or gain access to general employee statistical data.

FEATURE WALKTHROUGH - MAIN MENU



- The *MAIN MENU* is formatted in a list structure, with one option presented per line.
- The *MAIN MENU* is composed of eight different options or elements.
- Options include: *Display all Employees*, *List an Employee*, *Create an Employee*, *Edit an Existing Employee*, *Delete an Existing Employee*, *Employee Data Statistics*, *Print Employee List to File*, & *Exit Database*.
- Each individual option can be selected by simply using only the arrow keys: *up* or *down*, & then hitting: *enter/return* on the pointed (>) & highlighted option.

FEATURE WALKTHROUGH – DISPLAY ALL EMPLOYEE'S

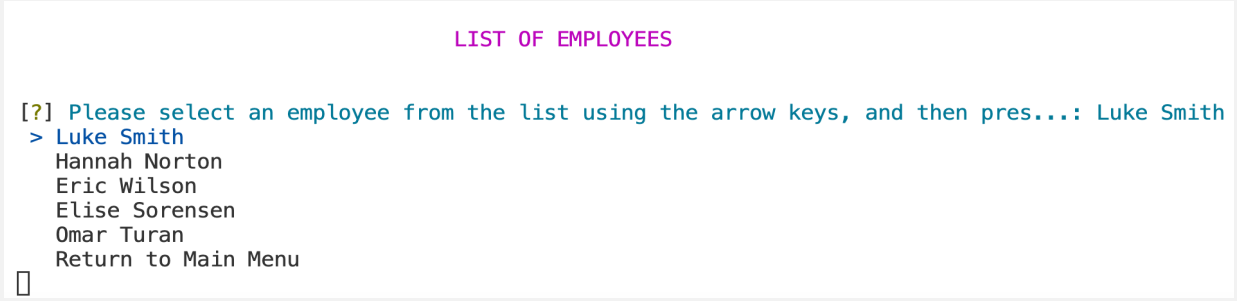
Initech Employee Database

Name	Age	Role	Salary
Luke Smith	20	Developer	100000
Hannah Norton	25	Graphic Design	80000
Eric Wilson	34	Management	120000
Elise Sorensen	25	Developer	100000
Omar Turan	35	Sales	120000

To Return to the Main Menu simply press: Enter/Return.

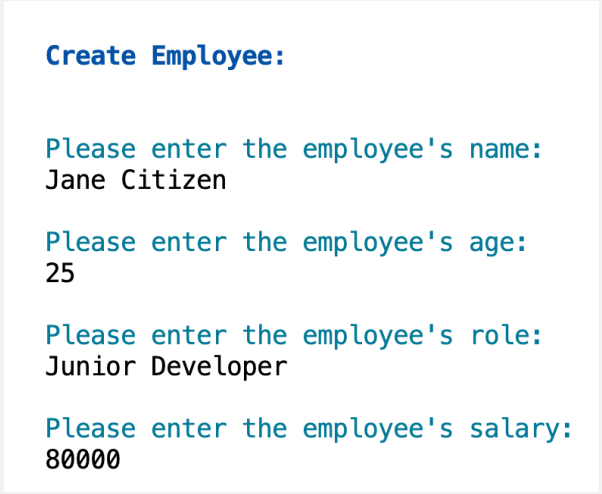
- The option: *Display all Employees* allows a user to view all currently existing employee's in the: *Initech Employee Database*.
- The *Display all Employees* option presents all employee's and their associated attributes in a neatly formatted table.
- The attribute labels of: *Name*, *Age, Role*, & *Salary* form the basis of each column in the table.
- Each employee and their associated attributes form the basis of each row in the table.
- All *String* attributes appear in green, while all *Integer* attributes appear in blue.
- The user controls when they would like to return to the: *Main Menu* by simply pressing *enter* or *return*.

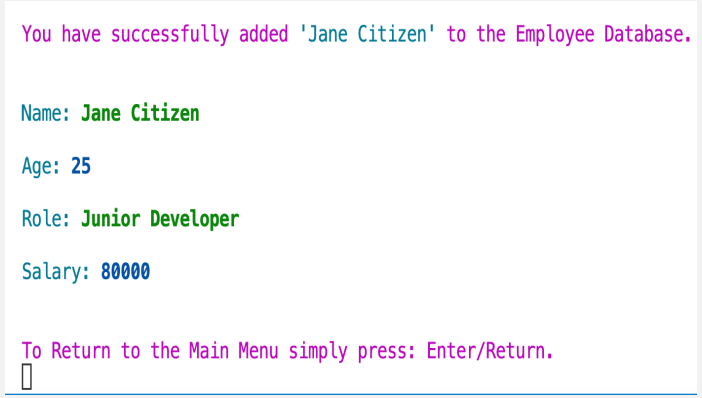
FEATURE WALKTHROUGH – LIST AN EMPLOYEE

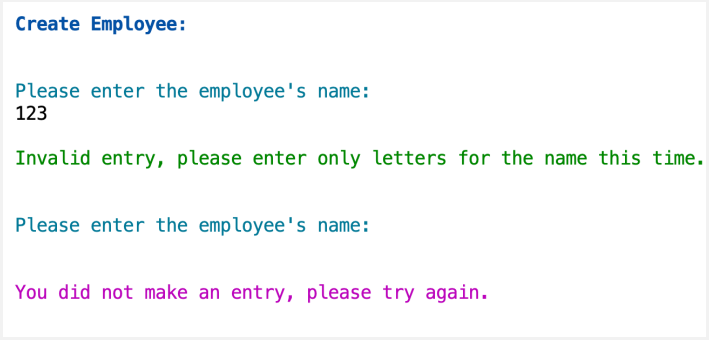


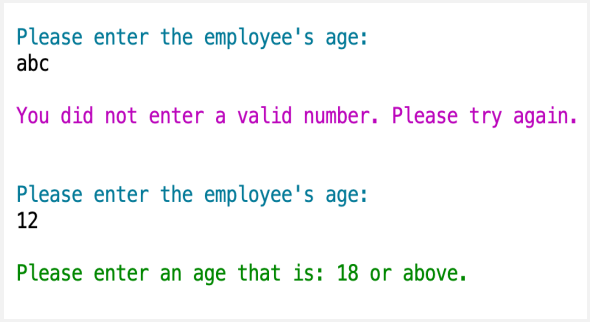
- The option: *List an Employee* allows a user to view a single employee and their associated attributes that exist in the: *Initech Employee Database*.
- When selected, the user again is faced with another menu to choose from. That is; *LIST OF EMPLOYEES*
- *LIST OF EMPLOYEES* is formatted in a list structure, with one option presented per line.
- The *LIST OF EMPLOYEES* menu is composed of the names of each *Employee* instance that exist in the database.
- Each individual name (option) can be selected by simply using only the arrow keys: *up* or *down*, & then hitting: *enter/return* on the pointed (>) & highlighted option.
- When a selection is made, the employee and their associated attributes are listed with each attribute clearly tagged.
- *String* attributes appear in green, while all *Integer* attributes appear in blue.

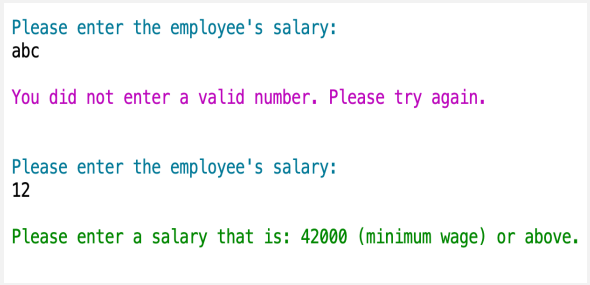
FEATURE WALKTHROUGH – CREATE AN EMPLOYEE





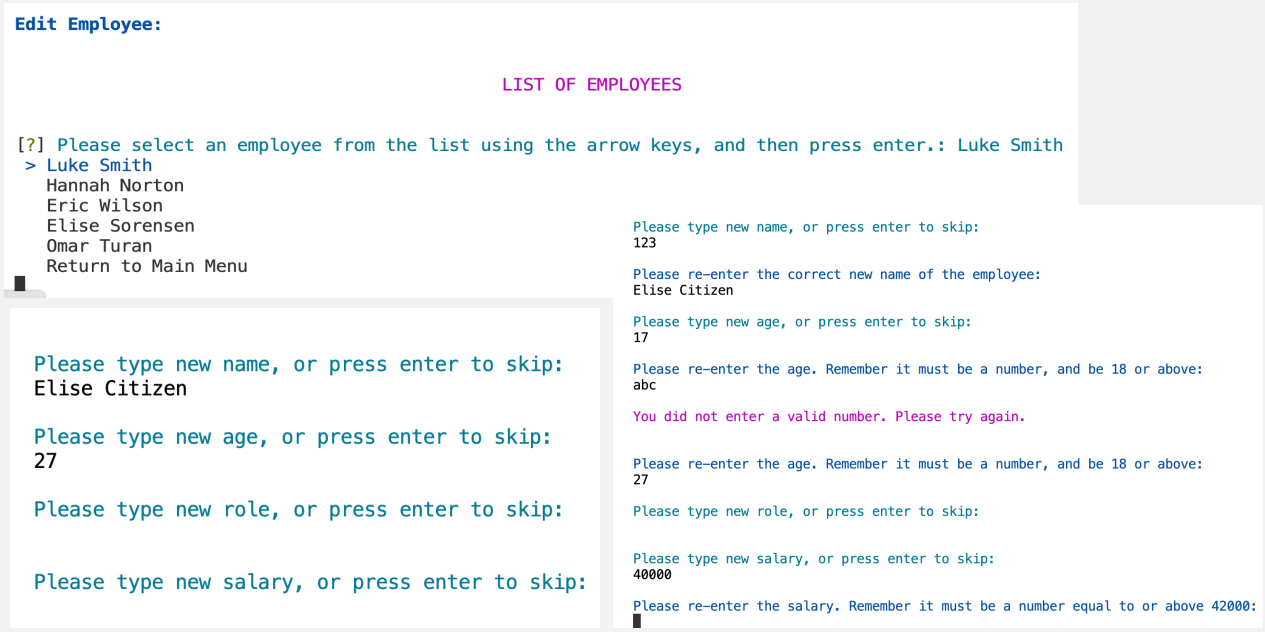


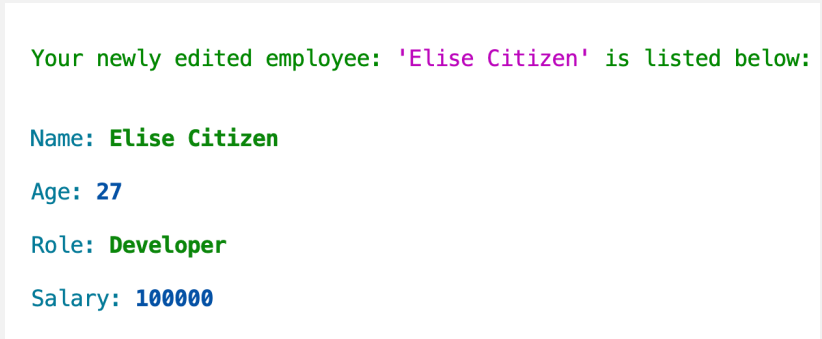




- The option: *Create an Employee* allows a user to create a single instance of an *Employee* and add their associated attributes to the: *Initech Employee Database*.
- To *Create an Employee*, the user is prompted to enter input to at least four different questions: *Please enter the employee's name*, *Please enter the employee's age*, *Please enter the employee's role*, & *Please enter the employee's salary*.
- For the inputs of: *Please enter the employee's name*, & *Please enter the employee's role*, both require inputs that are made up only alphabetical letters.
- For the inputs of: *Please enter the employee's age*, & *Please enter the employee's salary*, both require inputs that are made up of only numbered characters, and have minimum limits set of: *18* for *age*, & *42000* for *salary* entries.
- Once all attributes are correctly entered after any necessary prompting, the newly created *Employee* instance, and their associated attributes are listed with each attribute clearly tagged.

FEATURE WALKTHROUGH – EDIT AN EXISTING EMPLOYEE





- The option: *Edit an Existing Employee* allows a user to edit an existing employee in the: : *Initech Employee Database*, and change any of their associated attributes.
- When selected, the user is faced with another menu to choose from. That is; *LIST OF EMPLOYEES* menu, which is composed of the names of each employee that exist in the database.
- Each individual name (option) can be selected by simply using only the arrow keys: *up* or *down*, & then hitting: *enter/return* on the pointed (>) & highlighted option.
- When a selection is made, the employee and their associated attributes can now be edited.
- To *Edit an Existing Employee*, the user is prompted to enter input to at least four different questions: *Please type new name, or press enter to skip*, *Please type new age, or press enter to skip*, *Please type new role, or press enter to skip*, & *Please type new salary, or press enter to skip*.
- The user can choose which attributes to edit, or not edit. By simply entering input for those they wish to edit, and only hitting enter/return to those they don't. All inputs entered are still subject to the same validations as seen in *Create an Employee*.
- Once all chosen edits are entered, the newly edited employee, and their associated attributes are listed with each attribute clearly tagged.

FEATURE WALKTHROUGH – DELETE AN EXISTING EMPLOYEE

```
Delete Employee:

                                LIST OF EMPLOYEES

[?] Please select an employee from the list using the arrow keys, and then press enter.: Luke Smith
 > Luke Smith
   Hannah Norton
   Eric Wilson
   Elise Sorensen
   Omar Turan
   Return to Main Menu
```

```
You have successfully deleted 'Luke Smith' from the Employee Database:

                    Initech Employee Database

┌──────────────┬─────┬──────────────┬────────┐
│     Name     │ Age │     Role     │ Salary │
├──────────────┼─────┼──────────────┼────────┤
│ Hannah Norton│  25 │Graphic Design│  80000 │
│  Eric Wilson │  34 │  Management  │ 120000 │
│Elise Sorensen│  25 │   Developer  │ 100000 │
│  Omar Turan  │  35 │     Sales    │ 120000 │
└──────────────┴─────┴──────────────┴────────┘

To Return to the Main Menu simply press: Enter/Return.
```

- The option: *Delete an Existing Employee* allows a user to delete a single *Employee* instance,and their associated attributes from the: *Initech Employee Database*.
- When selected, the user is faced with another menu to choose from. That is the; *LIST OF EMPLOYEES* menu, which is composed of the names of each *Employee* instance that exists in the database.
- Each individual name (option) can be selected by simply using only the arrow keys: *up* or *down*, & then hitting: *enter/return* on the pointed (>) & highlighted option.
- When a selection is made, the employee and their associated attributes is immediately deleted when the user hits enter/return.
- The *Initech Employee Database* with all the remaining employee's is then shown to the user in a neatly formatted table, with the newly deleted employee now gone.

FEATURE WALKTHROUGH - EMPLOYEE DATA STATISTICS

```
GENERAL EMPLOYEE DATA


[?] Please select from the following options using the arrow keys, and press enter.: Display How Many Employee's Work for Initech
 > Display How Many Employee's Work for Initech
   Calculate Average Age of an Employee at Initech
   Calcaluate Average Salary of an Employee at Initech
   Return To Main Menu

The number of employee's currently employed at Initech is: '5'.
```

```
GENERAL EMPLOYEE DATA


[?] Please select from the following options using the arrow keys, and press enter.: Calculate Average Age of an Employee at Initech
   Display How Many Employee's Work for Initech
 > Calculate Average Age of an Employee at Initech
   Calcaluate Average Salary of an Employee at Initech
   Return To Main Menu

The average age of an employee at Initech is: '27' years.
```

```
GENERAL EMPLOYEE DATA


[?] Please select from the following options using the arrow keys, and press enter.: Calcaluate Average Salary of an Employee at Initech
   Display How Many Employee's Work for Initech
   Calculate Average Age of an Employee at Initech
 > Calcaluate Average Salary of an Employee at Initech
   Return To Main Menu

The average salary of an employee at Initech is: '$104000' dollars.
```

- The option: *Employee Data Statistics* allows a user to view general employee data calculations based on the current: *Initech Employee Database*.
- When selected, the user again is faced with another menu to choose from. That is; *GENERAL EMPLOYEE DATA*.
- Options include: *Display How Many Employee's Work for Initech*, *Calculate Average Age of an Employee at Initech*, *Calcaluate Average Salary of an Employee at Initech*, & *Return To Main Menu*.
- Each individual calculation (option) can be selected by simply using only the arrow keys: *up* or *down*, & then hitting: *enter/return* on the pointed (>) & highlighted option.
- When a selection is made, there is a clear print-out of the resulting data point relative to the current: *Initech Employee Database*.
- The user is always returned back to the: *GENERAL EMPLOYEE DATA* menu after the resultant output. They can then decide whether or not they wish to print out another data point, or *Return To Main Menu*.

FEATURE WALKTHROUGH - PRINT EMPLOYEE LIST TO FILE

```
                                PRINT EMPLOYEE LIST TO FILE


Congratulations; your 'Initech Employee Database' list has now been printed successfully to the
 file: 'list_of_employees.txt'. Below is a copy of what is printed in that text file:

Initech Employee Database
Date Printed: 2023-02-11

Employee 1:
 Name: Luke Smith
 Age: 20
 Role: Developer
 Salary: 100000

Employee 2:
 Name: Hannah Norton
 Age: 25
 Role: Graphic Design
 Salary: 80000

Employee 3:
 Name: Eric Wilson
 Age: 34
 Role: Management
 Salary: 120000

Employee 4:
 Name: Elise Sorensen
 Age: 25
 Role: Developer
 Salary: 100000

Employee 5:
 Name: Omar Turan
 Age: 35
 Role: Sales
 Salary: 120000


To Return to the Main Menu simply press: Enter/Return.
```

- The option: *Print Employee List to File* allows a user to print (write) the current *Initech Employee Database* to a text file: *list_of_employees.txt*.
- All additions, deletions, and changes made to employees and their attributes in the database, will be reflected in the writing of the text file. That is; it will reflect the most current version of the: *Initech Employee Database*.
- The user is firstly made aware of the successful outcome of the list being printed, given the name of the text file the list is being printed in, & shown a copy of what is printed in the text file directly in the terminal.
- The printed list includes; a heading that expresses what is being printed, the date it was printed, & an employee count with each employee and their associated attributes listed.
- The user controls when they would like to return to the: *Main Menu* by simply pressing *enter* or *return*.

FEATURE WALKTHROUGH - EXIT DATABASE

```
> Exit Database


Logged Out: Jane Citizen
Date: 2023-02-11

Print-out copy of: list_of_employees.txt:
Initech Employee Database
Date Printed: 2023-02-11

Employee 1:
 Name: Luke Smith
 Age: 20
 Role: Developer
 Salary: 100000

Employee 2:
 Name: Hannah Norton
 Age: 25
 Role: Graphic Design
 Salary: 80000

Employee 3:
 Name: Eric Wilson
 Age: 34
 Role: Management
 Salary: 120000

Employee 4:
 Name: Elise Sorensen
 Age: 25
 Role: Developer
 Salary: 100000

Employee 5:
 Name: Omar Turan
 Age: 35
 Role: Sales
 Salary: 120000

Jane Citizen, you have successfully logged out of the 'Initech' Employee Database Application. See you next
 time.
```

- The option: *Exit Database* 'log's out' the user from the: *Initech Employee Database; w*hen they have finished their viewing, altering, printing, or considering general employee statistical data.
- When a user chooses to exit the database, they are informed of this by a print statement that identifies their name, and the fact that they are: '*Logged Out*'. This statement also includes the date in which the user has logged out.
- Again, the user is shown a copy of what is printed in the text file directly in the terminal.
- Finally, the user is farewelled by name using the bash argument they initially entered, when prompted for their name input.

CODE LOGIC WALKTHROUGH – OVERVIEW

```
# WELCOME MESSAGE TO COLLECT INPUT FROM USER
echo "Hello there, to log in to the: 'Initech' Employee Database Application, please enter your name?"

read name

echo "It's nice to hear from you $name, you are now logged in to the: 'Initech' Employee Database Application."

# RUN APPLICATION COMMAND WITH ARGUMENT ENTERED:
python3 src/main.py $name
```

```
'''Alows for bash script input'''
from sys import argv

'''Import datetime module to get current date.'''
from datetime import date
```

```
'''Returns full name from bash script'''
def get_full_name():
    i = 1
    name = ""
    while i < len(argv):
        name = name + argv[i] + " "
        i = i + 1
    return name

# Log in details:
print(f"\n{Fore.BLACK}{Style.BRIGHT}Logged In: {Fore.BLUE}{get_full_name().title()} \n{Fore.BLACK}{Style.BRIGHT}Date: {Fore.BLUE}{date.today()}")
```

- For the user to run the application, they must first enter their name upon being asked. Using the bash command: *echo* this message is delivered.
- The bash command: *read* is used to allow the input. Conditionals are used ensure user input is entered.
- Together with *read*, the variable *name* is used to store the user's input, and is capitalised using: *awk*.
- Again, another message is delivered to the user with an *echo* that reflects the user input being captured successfully by adding the pointer to the *name* variable: *$name*.
- The dynamic variable: *$name* is now used as an argument when the app is executed in the terminal.
- By importing the *argv module*, the argument: *$name* given, can now be parsed using a *while loop* to collect each individual input initially given by the user, eg; first name + last name.
- The user's information can now be used directly in the running of the application, ie; as seen in: *Logged In & Out*.

CODE LOGIC WALKTHROUGH – MAIN MENU

```python
def main_menu():
    print("\n")
    print_centre(f"{Fore.MAGENTA}MAIN MENU")
    print("\n")

    questions = [
    inquirer.List(
        "Menu Options",
        message=f"{Fore.CYAN}Please select from the following options using the arrow keys, and press enter.",
        choices=["Display all Employees", "List an Employee", "Create an Employee", "Edit an Existing Employee", "Delete an Existing Employee", "Employee Dat
    answers = inquirer.prompt(questions)

    if answers["Menu Options"] == "Display all Employees":
        list_all_employees()
    if answers["Menu Options"] == "List an Employee":
        list_one_employee()
    if answers["Menu Options"] == "Create an Employee":
        create_employee()
    if answers["Menu Options"] == "Edit an Existing Employee":
        edit_employee()
    if answers["Menu Options"] == "Delete an Existing Employee":
        delete_employee()
    if answers["Menu Options"] == "Employee Data Statistics":
        data_statistics_search()
    if answers["Menu Options"] == "Print Employee List to File":
        print_employee_list()
    if answers["Menu Options"] == "Exit Database":
        print(f"\n{Fore.BLACK}{Style.BRIGHT}Logged Out: {Fore.BLUE}{get_full_name().title()} \n{Fore.BLACK}{Style.BRIGHT}Date: {Fore.BLUE}{date.today()}\n")
        exit()
    else:
        main_menu()


main_menu()
```

- The *MAIN MENU* is formatted in a list structure using a series of conditionals, that is; *if statements* that represent each option available to the user.
- The python import *Shutil* is used to centre the: *MAIN MENU* heading.
- The *MAIN MENU* uses the external python library: *Inquirer* integrated within the conditionals for it's main utility. Each option is implemented with the use of: *inquirer.List* & *inquirer.prompt* to create the menu options, & the functionality of selections being made using arrow keys.
- The "*Menu Options*" attribute that is initially set, is used to set each menu option prompt (*answers*). Each option that exists in the menu is stored in the variable: *choices*.
- The initial guidance communique that first alerts the user to how to select any menu option is stored in the variable: *message*.
- Using the variable: *answers*, which stores the prompt functionality; takes in the *questions* list as the argument. Whereby all menu options are now able to be set using the equality operator (==).
- All associated function calls are used under the requisite conditionals to execute each option selected by the user.

CODE LOGIC WALKTHROUGH – DISPLAY ALL EMPLOYEE'S

```python
from rich import print
from rich.console import Console
from rich.table import Table
```

```python
def list_all_employees():
    table = Table(title="\n[bold magenta]Initech Employee Database[/bold magenta]\n", style="cyan")
    table.add_column("Name", style="green", justify="center")
    table.add_column("Age", style="blue", justify="center")
    table.add_column("Role", style="green", justify="center")
    table.add_column("Salary", style="blue", justify="center")
    for employee in employee_list:
        table.add_row(str(employee.name), str(employee.age), str(employee.role), str(employee.salary))

    print(table)

    ask_return_main_menu()
```

```python
def ask_return_main_menu():
    selected = False
    while selected == False:
        selection = input(f"\n{Fore.MAGENTA}To Return to the Main Menu simply press enter/return:\n")
        if selection == "":
            selected = True
            go_back_to_main_menu()
```

- The option: *Display all Employees* is run using the function call: *list_all_employees*. The function employs logic to construct a table using the external python library of: *Rich*.
- The import *Console* is used to handle the styling of the table. That is; the *style*, & *justify* arguments to add colour & centre the inputs of the table.
- The import *Table* is used to construct the table itself, whereby table attributes can be set, & a *title* & *style* (*Console*) input can be used as arguments, of which all is stored in the variable: *table*.
- Each column is built using the *Table* class attribute of: *add_column*. Which takes arguments in a *string* format. Here I have used the *Employee* attributes of: *name, age, role, & salary* to label each column.
- Each row is built using the Table class attribute of: *add_row*. Here I have used a *for loop* to parse each row entry. That is; each instance of an Employee & their attributes that exist in the *employee_list* (database). Whereby all attributes have been formatted to strings to enter them as valid arguments.
- The table is then printed to the terminal using the import *print*, which takes the variable: *table* that stores the table itself, & all attributes given.
- The *ask_return_main_menu* function call is used, which applies simple Boolean logic to return the user back to the *main_menu* whenever they choose to proceed.

CODE LOGIC WALKTHROUGH – LIST AN EMPLOYEE

```
import colorama
from colorama import Fore, Back, Style
colorama.init(autoreset=True)
```

```
def list_one_employee():
    employee = employee_search()
    if employee:
        print(f"{Fore.MAGENTA}{Style.BRIGHT}\nEmployee listed:\n")
        list_employee(employee)
        list_one_employee()
```

```
def employee_search():
    employee_names = []
    for employee in employee_list:
        employee_names.append(employee.name)

    print("\n")
    print_centre(f"{Fore.MAGENTA}LIST OF EMPLOYEES")
    print("\n")

    questions = [
    inquirer.List(
        "Employees",
        message=f"{Fore.CYAN}Please select an employee from the list using the arrow keys, and then press enter.",
        choices= employee_names + ["Return to Main Menu"],)]
    selection = inquirer.prompt(questions)

    for name in employee_names:
        if selection["Employees"] == name:
            # Search through employee list:
            for employee in employee_list:
                # Check if employee name matches employee list name:
                if name == employee.name:
                    return employee
        if selection["Employees"] == "Return to Main Menu":
            go_back_to_main_menu()
```

```
def list_employee(employee):
    print(f"\n{Fore.CYAN}Name:{Fore.GREEN}{Style.BRIGHT} {employee.name}\n")
    print(f"{Fore.CYAN}Age:{Fore.BLUE}{Style.BRIGHT} {employee.age}\n")
    print(f"{Fore.CYAN}Role:{Fore.GREEN}{Style.BRIGHT} {employee.role}\n")
    print(f"{Fore.CYAN}Salary:{Fore.BLUE}{Style.BRIGHT} {employee.salary}\n")
```

- The option: *List an Employee* is run using the function call: *list_one_employee*. This function employs the logic of two other functions: *employee_search* & *list_employee*, together with the external python library of: *Inquirer* to display a chosen employee, & their associated attributes to the user. The import *Shutil* is used to centre the heading.
- The function: *employee_search* creates a list of employee's referenced by name only, & an option to return to the *main_menu*.
- This list is compiled using *Inquirer's: List* class that sets the *message variable* which provides the initial guidance to the user, & the *choices* variable that stores the options provided in that list.
- The *employee_names* variable is populated with the help of a series of for loops and conditionals used to retrieve each *name* attribute from all instances of an *Employee* that exist in the: *employee_list* (database).
- The variable: *selection* storing: *inquirer.prompt*, allows a user to make their selection using the arrow keys, taking *questions* as it's argument.
- The *return* value of the function: *employee_search* is then set to the local variable: *employee* in: *list_one_employee*.
- The condition of: *employee* in: *list_one_employee* is now met. Whereby a simple print statement is issued to the user, combined with the function call: *list_employee* that takes the *employee* variable returned as an argument, & then returns the attributes of the selected employee to the user in a series of print statements coloured by the external python feature: *Colorama*.

CODE LOGIC WALKTHROUGH – CREATE AN EMPLOYEE 1

```python
def create_employee():
    print(f"\n{Fore.BLUE}{Style.BRIGHT}Create Employee:\n")
    name = check_for_valid_string(f"\n{Fore.CYAN}Please enter the employee's name:{Fore.BLACK}\n").title()
    age = check_for_valid_number(f"\n{Fore.CYAN}Please enter the employee's age:{Fore.BLACK}\n", "age")
    role = check_for_valid_string(f"\n{Fore.CYAN}Please enter the employee's role:{Fore.BLACK}\n").title()
    salary = check_for_valid_number(f"\n{Fore.CYAN}Please enter the employee's salary:{Fore.BLACK}\n", "salary")
    employee_list.append(Employee(name, age, role, salary))

    '''Return instance of employee just created'''
    for employee in employee_list:
        if employee == name:
            return employee

    '''Confirm to the user the employee instance was created.'''
    print(f"\n{Fore.MAGENTA}You have successfully added {Fore.CYAN}'{employee.name}'{Fore.MAGENTA} to the Employee Database.\n")
    list_employee(employee)

    ask_return_main_menu()
```

```python
def check_for_valid_string(message):
    entry = ""
    while entry == "":
        entry = input(message)
        if entry == "":
            try:
                raise ValueError("You did not make an entry, please try again.")
            except ValueError as warning:
                print(f"\n{Fore.MAGENTA}{warning}\n")
        elif only_letters(entry) == False:
            entry = ""
            try:
                raise ValueError("Invalid entry, please enter only letters for the name this time.")
            except ValueError as warning:
                print(f"\n{Fore.GREEN}{warning}\n")
        else:
            return entry
```

```python
class Employee:
    def __init__(self, name, age, role, salary):
        self.name = name
        self.age = age
        self.role = role
        self.salary = salary
```

```python
def only_letters(string):
    return bool(re.match("^[a-zA-Z_ ]+$", string))
```

- The option: *Create an Employee* is run using the function call: *create_employee*. This function employs the logic of four other functions: *check_for_valid_string*, *check_for_valid_number*, *handle_number_errors*, *only_letters*, & the class: *Employee;* to instantiate a new *Employee* that is appended to: *employee_list* (database).
- This option is used together with the external python library of: *Colorama* to add emphasis to user input prompts, when being asked to enter their associated attributes to create another valid instance of an *Employee*.
- To create an employee, a user is first prompted to enter each attribute that is defined in the *Employee* class. All subsequent inputs are stored in the variables: *name*, *age*, *role*, & *salary* providing they are valid.
- The validity of each input is checked via, a series of functions that expect either a particular *string* entry, or *integer* entry.
- For the inputs of: *name*, & *role* the function: *check_for_valid_string* is called to run a sequence of checks that ensure that firstly; an input is made, & secondly; an input is made up of a string that consists solely of alphabetical letters. The function: *only_letters* nested within *check_for_valid*_string is part of the logic that returns a valid string.
- Conditional structures of: while loops, & if statements are used to control input prompts, together with error handling logic, until the user enters a valid input. The inbuilt *title* is used to capitalise *name* & *role*.

CODE LOGIC WALKTHROUGH – CREATE AN EMPLOYEE 2

```python
def check_for_valid_number(message, number_type):
    number = None
    while number == None:
        try:
            number = float(input(message))
            number = handle_number_errors(number, number_type)
            if number == False:
                return check_for_valid_number(message, number_type)
            else:
                return number
        except ValueError:
            print(f"\n{Fore.MAGENTA}You did not enter a valid number. Please try again.\n")
```

```python
def handle_number_errors(number, number_type):
    number = int(number)
    if number >= 18 and number_type == "age":
        return number
    elif number >= 42000 and number_type == "salary":
        return number
    else:
        if number_type == "age":
            print(f"{Fore.GREEN}\nPlease enter an age that is: 18 or above.\n")
        else:
            print(f"{Fore.GREEN}\nPlease enter a salary that is: 42000 (minimum wage) or above.\n")
        return False
```

- For the inputs of: *age*, & *role* the function: *check_for_valid_number* is called to run a sequence of checks that ensure that firstly; an input is made, & secondly; an input is made up of a string that consists solely of numbers. The function: *handle_number_errors is* nested within *check_for_valid_number*, & is part of the logic that returns a valid *integer*. An *integer* that is either *>=18* or *>=42000* for *age* & *salary* inputs respectively.
- Conditional structures of: while loops, & if statements are used to control input prompts, together with error handling logic until the user enters a valid input.
- Once all four valid inputs are received for the attributes: *name, age*, *role*, & *salary* from the user. The variables storing those input attributes are used as the argument for the append method being applied to add the new instance of the *Employee* to *employee_list*.
- Finally a print statement is made that includes the name of the *Employee* instance just created. A for loop, & if statement is used to return the *Employee* instance: *employee*. Whereby it can also be used as the argument for the function then called: *list_employee*, to reflect back to the user the successfully created employee & associated attributes.
- Again the user is prompted to choose when they would like to return back to the *Main Menu* using the function call: *ask_return_main_menu*.

CODE LOGIC WALKTHROUGH – EDIT AN EXISTING EMPLOYEE 1

```python
def edit_employee():
    print(f"{Fore.BLUE}{Style.BRIGHT}\nEdit Employee:")
    employee_exists = employee_search()
    if employee_exists:
        edit_employee_attributes(employee_exists)
```

```python
def employee_search():
    employee_names = []
    for employee in employee_list:
        employee_names.append(employee.name)

    print("\n")
    print_centre(f"{Fore.MAGENTA}LIST OF EMPLOYEES")
    print("\n")

    questions = [
    inquirer.List(
        "Employees",
        message=f"{Fore.CYAN}Please select an employee from the list using the arrow keys, and then press enter.",
        choices= employee_names + ["Return to Main Menu"],)]
    selection = inquirer.prompt(questions)

    for name in employee_names:
        if selection["Employees"] == name:
            # Search through employee list:
            for employee in employee_list:
                # Check if employee name matches employee list name:
                if name == employee.name:
                    return employee
        if selection["Employees"] == "Return to Main Menu":
            go_back_to_main_menu()
```

- The option: *Edit an Existing Employee* is run using the function call: *edit_employee*. This function employs the logic of six other functions: *edit_employee_attributes*, *check_for_valid_string*, *check_for_valid_number*, *handle_number_errors*, *only_letters*, & *employee_search* to edit the attributes of an existing *Employee* instance.
- This option is used together with the external python library of: *Colorama* to add emphasis to user input prompts, when being asked to enter their associated attributes to edit a particular instance of an *Employee*.
- The *edit_employee* functions firstly calls the: *employee_search* function to give the user the ability to select which *Employee* instance they would like to edit using the logic already discussed in the: *List an Employee* option.
- The return value of *employee_search* is then stored in the variable: *employee_exists*, & can now be used to satisfy a condition to call for the function: *edit_employee_attributes*.
- The: *edit_employee_attributes* function uses the: *employee_exists* variable storing the employee just selected as an argument.
- Four new variables are introduced in the function: *edit_employee_attributes*: *new_name*, *new_age*, *new_role*, & *new_salary*. All of which store only valid user input in relation to the selected employee's attributes.

CODE LOGIC WALKTHROUGH – EDIT AN EXISTING EMPLOYEE 2

```
def edit_employee_attributes(employee):
    # Check for a valid string if input given.
    new_name = input(f"\n{Fore.CYAN}Please type new name, or press enter to skip: {Fore.BLACK}\n").title()
    if new_name != "":
        try:
            if only_letters(new_name) == False:
                raise ValueError(f"\n{Fore.BLUE}Please re-enter the correct new name of the employee:{Fore.BLACK}\n")
            else:
                new_name
        except ValueError as warning:
            new_name = check_for_valid_string(warning).title()
    # Check for a valid number if input given.
    new_age = input(f"\n{Fore.CYAN}Please type new age, or press enter to skip:{Fore.BLACK}\n")
    if new_age != "":
        try:
            new_age = int(float(new_age))
            if new_age >= 18:
                new_age = handle_number_errors(new_age, "age")
            else:
                raise ValueError
        except ValueError:
            new_age = check_for_valid_number(f"\n{Fore.BLUE}Please re-enter the age. Remember it must be a number, and be 18 or above:{Fore.BLACK}\n", "age")
    # Check for a valid string if input given.
    new_role = input(f"\n{Fore.CYAN}Please type new role, or press enter to skip:{Fore.BLACK}\n").title()
    if new_role != "":
        try:
            if only_letters(new_role) == False:
                raise ValueError(f"\n{Fore.BLUE}Please re-enter the correct new role of the employee:{Fore.BLACK}\n")
            else:
                new_role
        except ValueError as warning:
            new_role = check_for_valid_string(warning).title()
    # Check for a valid number if input given.
    new_salary = input(f"\n{Fore.CYAN}Please type new salary, or press enter to skip:{Fore.BLACK}\n")
    if new_salary != "":
        try:
            new_salary = int(float(new_salary))
            if new_salary >= 42000:
                new_salary = handle_number_errors(new_salary, "salary")
            else:
                raise ValueError
        except ValueError:
            new_salary = check_for_valid_number(f"\n{Fore.BLUE}Please re-enter the salary. Remember it must be a number equal to or above 42000:{Fore.BLACK}\n", "salary")
```

```
if new_name != "":
    employee.name = new_name
if new_age != "":
    employee.age = new_age
if new_role != "":
    employee.role = new_role
if new_salary != "":
    employee.salary = new_salary
print(f"{Fore.GREEN}\nYour newly edited employee: {Fore.MAGENTA}'{employee.name}'{Fore.GREEN} is listed below:\n")
list_employee(employee)
edit_employee()
ask_return_main_menu()
```

- Any input or non-input entered by the user is first assessed by a conditional, that checks for non-empty strings, that is basically; whether an attribute is edited.
- If a string is not empty, ergo an input is given by the user, the condition is satisfied; & a series of validations are triggered. That is; conditionals, error handling logic, & further function calls, of which the logic has already been described, that handle for a particular input type for an attribute. For example; *new_age* must be able to be converted to an *integer*, & be *>=18*.
- Empty strings are accepted, & are immediately stored in the requisite variable, eg; *new_name* for the attribute *name*.
- Empty strings become important in the final stages of the logic of: *edit_employee_attributes*. The final conditional which is responsible for the re-setting or editing of an employees attributes, is only met if there is a requisite input entered by the user. Therefore empty strings are responsible for conserving employee attributes.
- It is only when an input is given, the final condition is met, that results in the selected employee's attribute being re-set to equal that input given.
- Finally, a print statement is made that includes the name or new name of the selected employee, & the function *list_employee* is called to reflect back to the user the newly edited employee's attributes.

CODE LOGIC WALKTHROUGH – DELETE AN EXISTING EMPLOYEE

```python
def delete_employee():
    print(f"{Fore.BLUE}{Style.BRIGHT}\nDelete Employee:")
    employee = employee_search()
    if employee:
        employee_list.remove(employee)
    print(f"\n{Fore.GREEN}You have successfully deleted {Fore.MAGENTA}'{employee.name}'{Fore.GREEN} from the Employee Database:\n")
    list_all_employees()
```

```python
def employee_search():
    employee_names = []
    for employee in employee_list:
        employee_names.append(employee.name)

    print("\n")
    print_centre(f"{Fore.MAGENTA}LIST OF EMPLOYEES")
    print("\n")

    questions = [
    inquirer.List(
        "Employees",
        message=f"{Fore.CYAN}Please select an employee from the list using the arrow keys, and then press enter.",
        choices= employee_names + ["Return to Main Menu"],)]
    selection = inquirer.prompt(questions)

    for name in employee_names:
        if selection["Employees"] == name:
            # Search through employee list:
            for employee in employee_list:
                # Check if employee name matches employee list name:
                if name == employee.name:
                    return employee
        if selection["Employees"] == "Return to Main Menu":
            go_back_to_main_menu()
```

```python
def list_all_employees():
    table = Table(title="\n[bold magenta]Initech Employee Database[/bold magenta]\n", style="cyan")
    table.add_column("Name", style="green", justify="center")
    table.add_column("Age", style="blue", justify="center")
    table.add_column("Role", style="green", justify="center")
    table.add_column("Salary", style="blue", justify="center")
    for employee in employee_list:
        table.add_row(str(employee.name), str(employee.age), str(employee.role), str(employee.salary))

    print(table)

    ask_return_main_menu()
```

- The option: *Delete an Existing Employee* is run using the function call: *delete_employee*. This function employs the logic of two other functions: *employee_search* & *list_all_employees*, to carry out the required deletion of an instance of an *Employee*, & *to* reflect back to the user that employee selected has been deleted from: *employee_list*.
- This option is used together with the external python library's of: *Rich*, *Colorama* & *Shutil* to add styling to the terminal output.
- The *delete_employee* function firstly calls the: *employee_search* function to give the user the ability to select which *Employee* instance they would like to delete using the logic already discussed in the: *List an Employee* option.
- The: *delete_employee* function uses the: *employee* variable that stores the employee just selected as an argument (*employee_search*), & can now satisfy a condition that enables the employee to be deleted.
- The *remove* method is applied to: *employee_list* with the selected employee used as the argument.
- A print statement including the name attribute is employed to inform the user of the successful deletion.
- Finally, the *list_all_employees* function is called to exhibit to the user the employee database (employee_list) in it's table format that conveys that the employee has in fact been deleted.

CODE LOGIC WALKTHROUGH – EMPLOYEE DATA STATISTICS 1

```python
def data_statistics_search():
    print("\n")
    print_centre(f"{Fore.MAGENTA}GENERAL EMPLOYEE DATA")
    print("\n")

    questions = [
    inquirer.List(
        "Menu Options",
        message=f"{Fore.CYAN}Please select from the following options using the arrow keys, and press enter.",
        choices=["Display How Many Employee's Work for Initech", "Calculate Average Age of an Employee at Initech", "Calcaluate Average Salary of an Employee at Initech", "Return To Main Menu"],)]
    answers = inquirer.prompt(questions)

    if answers["Menu Options"] == "Display How Many Employee's Work for Initech":
        display_number_of_employees()
    if answers["Menu Options"] == "Calculate Average Age of an Employee at Initech":
        calculate_avg_age_of_employees()
    if answers["Menu Options"] == "Calcaluate Average Salary of an Employee at Initech":
        calculate_avg_salary_of_employees()
    if answers["Menu Options"] == "Return To Main Menu":
        go_back_to_main_menu()
    else:
        data_statistics_search()
```

```python
def display_number_of_employees():
    i = 0
    try:
        for employee in employee_list:
            i = i + 1
        print(f"{Fore.GREEN}The number of employee's currently employed at Initech is: {Fore.MAGENTA}'{i}'{Fore.GREEN}.\n")
        return i
    except TypeError:
        data_statistics_search()
```

- The option: *Employee Data Statistics* is run using the function call: *data_statistics_search*, together with the external python libraries of: *Inquirer*, *Colorama*, & *Shutil*.
- This function employs the logic of three other functions: *display_number_of_employees, calculate_avg_age_of_employees*, & *calculate_avg_salary_of_employees*.
- *Employee Data Statistics* is used ultimately to create a list to select from, & display certain data points based on calculations made from existing employee's attributes that are listed in: *employee_list*.
- The function: *data_statistics_search* creates a list using a series of conditionals, together with: *Inquirer* that represent each option available to the user.
- This list is compiled using *Inquirer's: List* class that uses *message* to store the initial guidance given to the user, & the *choices* variable to store the options provided in that list.
- The variable: *answers* storing: *inquirer.prompt*, allows a user to make their selection using arrow keys, taking the *questions* variable as it's argument.
- All associated function calls are used under the requisite conditionals to execute each option selected by the user.

CODE LOGIC WALKTHROUGH – EMPLOYEE DATA STATISTICS 2

```python
def calculate_avg_age_of_employees():
    result = 0
    i = 0
    try:
        for employee in employee_list:
            i = i + 1
            result = result + employee.age
        final_result = int(result / i)
        print(f"{Fore.GREEN}The average age of an employee at Initech is: {Fore.MAGENTA}'{final_result}' {Fore.GREEN}years.\n")
        return final_result
    except TypeError:
        data_statistics_search()
```

```python
def calculate_avg_salary_of_employees():
    result = 0
    i = 0
    try:
        for employee in employee_list:
            i = i + 1
            result = result + employee.salary
        final_result = int(result / i)
        print(f"{Fore.GREEN}The average salary of an employee at Initech is: {Fore.MAGENTA}'${final_result}' {Fore.GREEN}dollars.\n")
        return final_result
    except TypeError:
        data_statistics_search()
```

```python
def go_back_to_main_menu():
    from main import main_menu
    main_menu()
```

- If the selection: '*Display How Many Employee's work for Initech*' is chosen. The function: *display_number_of_employees* is called.
- *display_number_of_employees* uses a variable *i* to store the count, as a for loop iterates through each employee listed in: *employee_list*. *try/except* is used to handle for any errors.
- A print statement is used to return the: *final_result*, & inform the user of the requested data point.
- *calculate_avg_age_of_employees* also uses a variable *i* to store the count, & the variable: *result* to tally all ages of employees using a for loop that iterates through each employee listed in: *employee_list*. Whereby *final_result* can then be divided by *i* to perform the calculation to attain the average age of employees. Again, *try/except* is used to handle for any errors.
- A print statement is used to return the: *final_result*, & inform the user of the requested data point.
- *calculate_avg_salary_of_employees:* uses the same logic as: *calculate_avg_age_of_employees*, however instead tallies the average salary's of all employee's that exist in: *employee_list*, & returns the *final_result* of it's calculated data point to the user.
- With each option chosen the user is returned to the: *General Employee Data* menu using the *else* statement in the function: *data_statistics_search*.
- The user is then free to search other datapoints, or return to the: *Main Menu* when they choose using the: *go_back_to_main_menu* function call.

CODE LOGIC WALKTHROUGH – PRINT EMPLOYEE LIST TO FILE

```
from datetime import date
```

```
import os
```

```
def print_employee_list():
    print("\n")
    print_centre(f"{Fore.MAGENTA}PRINT EMPLOYEE LIST TO FILE")
    print("\n")
    text_file = open("list_of_employees.txt", "w")
    num = 0
    today = date.today()
    text_file.write("Initech Employee Database\n")
    text_file.write(f"Date Printed: {str(today)} \n\n")
    for employee in employee_list:
        num += 1
        entry = f"Employee {num}:\n Name: {employee.name}\n Age: {employee.age}\n Role: {employee.role}\n Salary: {employee.salary} \n\n"
        text_file.write(entry)
    text_file.close()
    print_centre(f"{Fore.CYAN}Congratulations; your 'Initech Employee Database' list has now been printed successfully to the file: {Fore.GREEN}'list_of_employees.txt'. {Fore.CYAN}Below is a copy of what is printed in that text file:\n")
    os.system("cat list_of_employees.txt")
    ask_return_main_menu()
```

- The option: *Print Employee List to File* is run using the function call: *print_employee_list*, together with the python modules of: *Colorama, Shutil, Datetime, & Os*.
- This function employs the logic of: python's inbuilt function for writing files to complete it's main task of printing a text file.
- *Print Employee List to File* uses the variable: text_file to store the the newly opened: readable text file: *list_of_employees.txt* that is to be written. The "*w*" denotes: 'write only'.
- Using: "*w*", the text file can be overwritten at any time, that is; if any changes have been made to *employee_list* by a user.
- A variable *num* is defined to be used as a counter for each employee in *employee_list*.
- A variable *today* is defined to store the result *date.today* which functions to record the current date.
- The function *write* together with *text_file* is used to print or write each input given as an argument.
- A for loop is used to iterate through the: *employee_list* to capture all instances of an *Employee* that exist there, and print them to *list_of_employees.txt*.
- *entry,* which stores all iterations from the above for loop, is used as an argument for the function: *write* that is used together *text_file* to write the argument given (*entry*).
- The function *close* together with *text_file* is used to signal the end of the writing of the text file: *list_of_employees.txt*.
- A print statement is used to let the user know that the text file has been printed successfully, & points to where it can be found, that is: *list_of_employees.txt*.
- By using: *os.system*, which allows for command line arguments to be given, I have implemented: the printing of the contents of the text file just created in the terminal using the command: *cat list_of_employees.txt*.
- The user can return to the: *Main Menu* when they choose to, using the function call: *ask_return_main_menu*.

CODE LOGIC WALKTHROUGH – EXIT DATABASE

```
if answers["Menu Options"] == "Exit Database":
    print(f"\n{Fore.BLACK}{Style.BRIGHT}Logged Out: {Fore.BLUE}{get_full_name().title()} \n{Fore.BLACK}{Style.BRIGHT}Date: {Fore.BLUE}{date.today()}\n")
    exit()
```

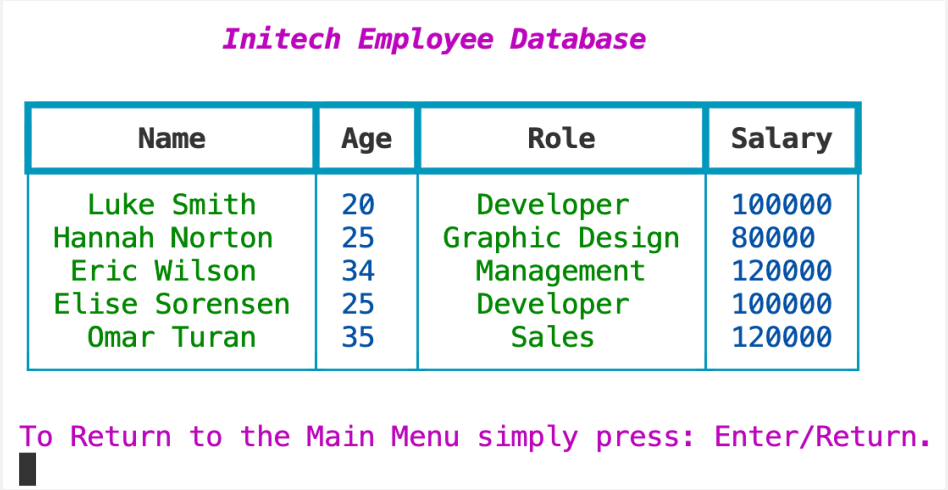
```
# FINAL MESSAGE AFTER APP EXITED:
echo "$name, you have successfully logged out of the 'Initech' Employee Database Application. See you next time."
```

- The option: *Exit Database* is run using the inbuilt python function call: *exit*, together with the python modules of: *Colorama, Datetime, & Sys.*
- Before the terminal application is exited, a print statement is used to inform the user by name that they are: "*logged Out*", using the *Sys* module's: *argv* that returns the users full name from the function call: *get_full_name*.
- The date that the user is: "*logged Out*" is also printed using *Datetime*'s: *date.today* which returns the actual date when called.
- The inbuilt python function call of *exit* is used to *exit* the application.
- When the app is exited, a final message is given using the the command: *echo* to print one final farewell message to the user. Including the initial name that was input, and captured by the bash variable: *$name*.

REVIEW BUILD PROCESS – MAIN MENU

- The challenge for the development of the: *Main Menu* was to build a menu that clearly listed all options that were available to the user in the application.
- The menu needed to be: clear, concise, and most importantly easy to navigate.
- The external python package of: *Inquirer* that I came upon in my search, was vital to achieving this goal.
- The python package: *Colorama,& Shutil* added style, and readability to the *Main Menu. Shutil* created a heading that was beautifully centred in the terminal, & *Colorama* helped highlight important information for the user to see.
- There were no ethical issues involved in the process of building the: *Main Menu*.
- My favourite part has to be the: *Inquirer* functionality that was added. That is; the use of arrow keys: up & down to select an option listed in the menu.

REVIEW BUILD PROCESS – DISPLAY ALL EMPLOYEE'S



- The challenge for the development of: *Display All Employee's* was to build a list that clearly showcased all employee's, & their associated attributes in a way that was easy to decipher for the user when output to the terminal.
- Because this option was responsible for exhibiting the entire database (*employee_list*), with every instance of an *Employee*, & their associated attributes needing to be displayed in the one output clearly, I searched for an external python library that could achieve this goal. I came upon the library: *Rich*.
- *Rich* was central to my success in formatting a structure that could easily convey to the user the entire employee database (*employee_list*) in clear & readable fashion.
- I used *Rich's Table* formatting functionality to successfully accomplish this; to output a table structure with styling that clearly conveys all employee's, & their associated attributes in an explicitly cogent manner for the user to view in the terminal.
- There were no ethical issues involved in the process of building: *Display All Employee's* table format.
- My favourite part has to be the table output that holds all the employee data, & the colouring used to discern specifically: strings (green) & integers (blue).

REVIEW BUILD PROCESS – LIST AN EMPLOYEE

```
                    LIST OF EMPLOYEES


[?] Please select an employee from the list using the arrow keys, and then pres...: Luke Smith
 > Luke Smith
   Hannah Norton
   Eric Wilson
   Elise Sorensen
   Omar Turan
   Return to Main Menu
```

- The challenge for the development of: *List an Employee* was to build a menu (*List of Employees*) that clearly listed all names of all existing employee's stored in the: *employee_list*.
- Again the external python package of: *Inquirer* was used in the build process create an easily manoeuvrable menu using only arrow keys: up or down.
- However, my main challenge lay in building a function (*search_employee*) that could be used with *Inquirer*'s *List* class to populate the: *choices* variable with all the employee *name* attributes stored in: *employee_list,* & allowing an *Employee* instance to be selected by the user by name only.
- There were no ethical issues involved in the process of building: *List an Employee*.
- My favourite part has to be the creation of the employee list of names that is able to be easily searched through, & a selection made. A lot of my time went into building this list, that is powered by the: *search_employee* function I created with the use of *Inquirer*.

REVIEW BUILD PROCESS – CREATE AN EMPLOYEE

```
Create Employee:

Please enter the employee's name:
123

Invalid entry, please enter only letters for the name this time.

Please enter the employee's name:

You did not make an entry, please try again.
```

```
Please enter the employee's age:
abc

You did not enter a valid number. Please try again.

Please enter the employee's age:
12

Please enter an age that is: 18 or above.
```

- The challenge for the development of: *Create an Employee* was to build a function that could handle for no user input, & user inputs that needed to be either strictly alphabetical letters, or strictly numbered characters over a certain threshold.
- My real challenge was in the building of three particular functions: *check_for_valid_number*, *check_for_valid_string*, & *handle_number_errors* that were responsible for handling user input, to ultimately return a valid string or integer from the user.
- There were no ethical issues involved in the process of building: *Create an Employee*.
- My favourite part has to be the functionality of: *check_for_valid_number*, *check_for_valid_string*, & *handle_number_errors* that is built into this option. Together with: *Colorama*, they are able to handle any input from a user, & alert the user with a series of colourful warnings to cajole them into entering a valid input.

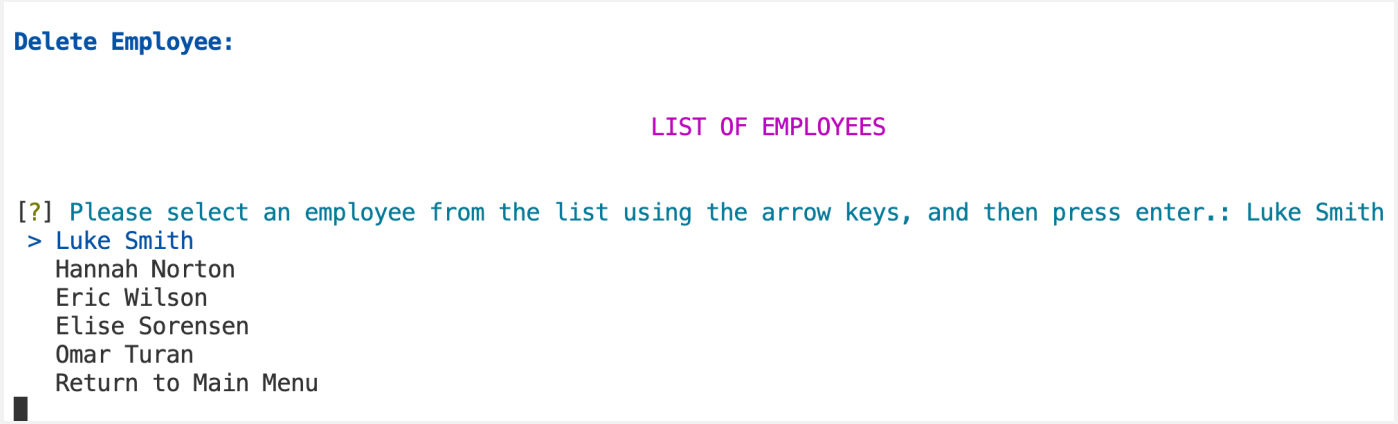
REVIEW BUILD PROCESS – EDIT AN EXISTING EMPLOYEE

```
Please type new name, or press enter to skip:
Elise Citizen

Please type new age, or press enter to skip:
27

Please type new role, or press enter to skip:


Please type new salary, or press enter to skip:
```

```
Your newly edited employee: 'Elise Citizen' is listed below:


Name: Elise Citizen

Age: 27

Role: Developer

Salary: 100000
```

- The challenge for the development of: *Edit an Existing Employee* was to build a function that could handle for attributes that were not intended to be edited, along with those that were. That is; a function that could conserve any attribute a user did not want to edit, while still validating those attributes that were sought to be edited with the requisite logic.
- My real challenge was building a function that could handle all aforementioned tasks, that was as DRY as possible, that would allow the user to easily skip through to the attributes they would actually like edit., & to use empty strings to power those attributes that were to be conserved. While also using the same logic (functions) that validates any input given by the user, to create a newly edited employee.
- There were no ethical issues involved in the process of building: *Edit an Existing Employee*.
- My favourite part has to be how easy it is for a user to edit a particular attribute, while not editing others. That is; a user only need press enter/return to skip over any attributes they do not wish to edit.

REVIEW BUILD PROCESS – DELETE AN EXISTING EMPLOYEE



- The challenge for the development of: *Delete an Existing Employee* was to build a list that included each employee by name that existed in: *employee_list* & to make that deletion a quick, & easy process for the user to carry out.
- My real challenge was creating the list of all employee names with the python library: *Inquirer*, together with the function: *search_employee*. Once that logic was built, a user was able to instantly delete an employee, & all of their associated attributes by simply making a selection from that list together with the: *delete_employee* function.
- There were no ethical issues involved in the process of building: *Delete an Existing Employee*.
- My favourite part has to be how easy it is for a user to delete an employee from: *employee_list*. Once a selection is highlighted in the list of names; all a user has to do is hit enter/return, & the employee is then deleted.

REVIEW BUILD PROCESS – EMPLOYEE DATA STATISTICS

```
                                    GENERAL EMPLOYEE DATA


[?] Please select from the following options using the arrow keys, and press enter.: Calcaluate Average Salary of an Employee at Initech
   Display How Many Employee's Work for Initech
   Calculate Average Age of an Employee at Initech
 > Calcaluate Average Salary of an Employee at Initech
   Return To Main Menu

The average salary of an employee at Initech is: '$104000' dollars.
```

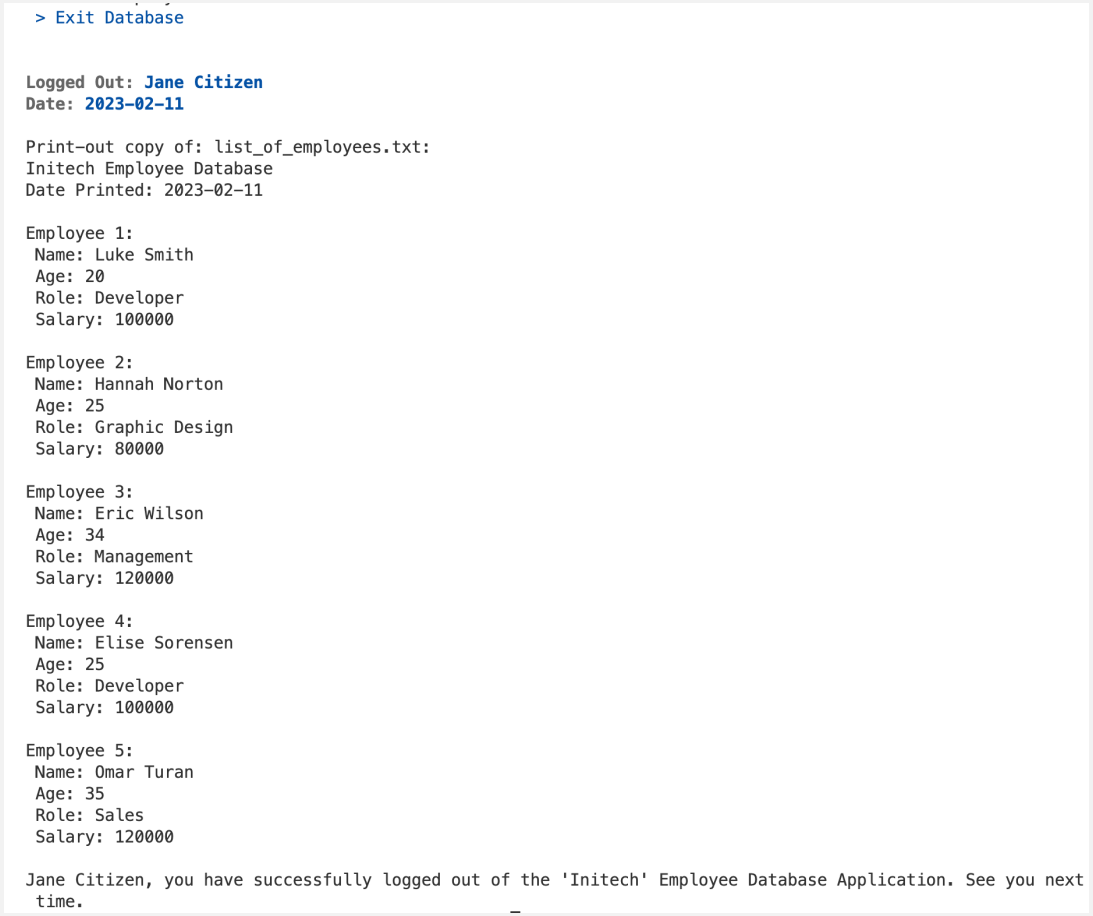
- The challenge for the development of: *Employee Data Statistics* was to build a list that included each option of: *Display How Many Employee's Work for Initech*, *Calculate Average Age of an Employee at Initech*, & *Calcaluate Average Salary of an Employee at Initech*; that instantly output those calculations when selected.
- My real challenge was creating the list of all the options listed above with the python library: *Inquirer.*
- There were no ethical issues involved in the process of building: *Employee Data Statistics*.
- My favourite part has to be how easy it is for a user to obtain each data point calculation, by simply making a selection, & hitting enter/return.

REVIEW BUILD PROCESS – PRINT EMPLOYEE LIST TO FILE

```
                    PRINT EMPLOYEE LIST TO FILE

Congratulations; your 'Initech Employee Database' list has now been printed successfully to the
 file: 'list_of_employees.txt'. Below is a copy of what is printed in that text file:

Initech Employee Database
Date Printed: 2023-02-11

Employee 1:
 Name: Luke Smith
 Age: 20
 Role: Developer
 Salary: 100000

Employee 2:
 Name: Hannah Norton
 Age: 25
 Role: Graphic Design
 Salary: 80000

Employee 3:
 Name: Eric Wilson
 Age: 34
 Role: Management
 Salary: 120000

Employee 4:
 Name: Elise Sorensen
 Age: 25
 Role: Developer
 Salary: 100000

Employee 5:
 Name: Omar Turan
 Age: 35
 Role: Sales
 Salary: 120000


To Return to the Main Menu simply press: Enter/Return.
```

- The challenge for the development of the: *Print Employee List to File* was to build a text file that could be written & overwritten when, any new addition or alteration was made to: *employee_list*, & for that list to be time-stamped with the actual date when printed.
- My real challenge was to learn more about the inbuilt python modules for writing text files, & adding real time date stamps.
- There were no ethical issues involved in the process of building: *Print Employee List to File*.
- My favourite part has to be the date stamp provided by the python *datetime* module, with each print out of a text file recording the actual date of writing.

REVIEW BUILD PROCESS – EXIT DATABASE



- The challenge in the development of the: *Exit Database* was to reflect back to the user the initial input of their name when they first executed the application.
- My real challenge was to include the users input name from the command line prompt in *bash* as an argument to be used in my python script.
- Firstly I needed to ensure a name was input by the user before the execution of the application, & that the users entire name, that is; first + last was included. I had to build another function: *get_full_name* to parse from *argv* the name/s input that had spaces in between them.
- There were no ethical issues involved in the process of building: *Exit Database*.
- My favourite part has to be the users own input name being used together with the print statement: '*Logged Out*'. It like that it adds a personal interactive element to the app I have built.

END OF PRESENTATION

THANK YOU FOR LISTENING

Laylah De Paull – T1A3 Terminal Application

GCAS022002