

PROJECT 3 REPORT

Overview

This program is meant to be the start of a bookkeeping application that stores and saves information on clients, employees, products on hand, and transactions involving the business. As a result, it is very general purpose and could apply to any business that needs to maintain information on clients, keep track of money being paid to employees, profit generated from sales, and available funds to invest in the business. It performs all functions required of a database (creating, reading, updating, and deleting), albeit in a primitive manner. As of now, the data is optionally saved in a text file, but it cannot read files to add to them, which would be essential to proper function. This program contains the main file (P3_Sharff.py) which contains the bulk of the program and a class file (P3_Sharff_Classes.py) that contains the 4 class definitions, imported to P3_Sharff. These need to be together to work properly.

Expected I/O

Input is collected from the user at every step of the program. Yes/no decisions which decide whether to continue looping mechanisms are within the majority of functions (see yesorno()). Choices in the form of multiple-choice letters are collected and mapped using dictionaries containing the specific lists (for reading) and/or functions as key/value pairs associated with the user choice to select actions to perform and the objects on which to perform them. Deleting, updating, and logging sales requires the user to input a name associated with the given object type to reference it by iterating through the corresponding lists. Creating entries directly collects information to be set as attributes for the instantiated object in its corresponding list using setter-getter methods.

Output is printed as text to the console to indicate the processes occurring to the user in nearly every function. This could be information read from the lists regarding the objects, displaying transactions and balances, indicating when an object is created, etc. Additionally, the output can be stored as a text file for future reference but cannot be used as input for the program (for now).

Use of Computing Concepts

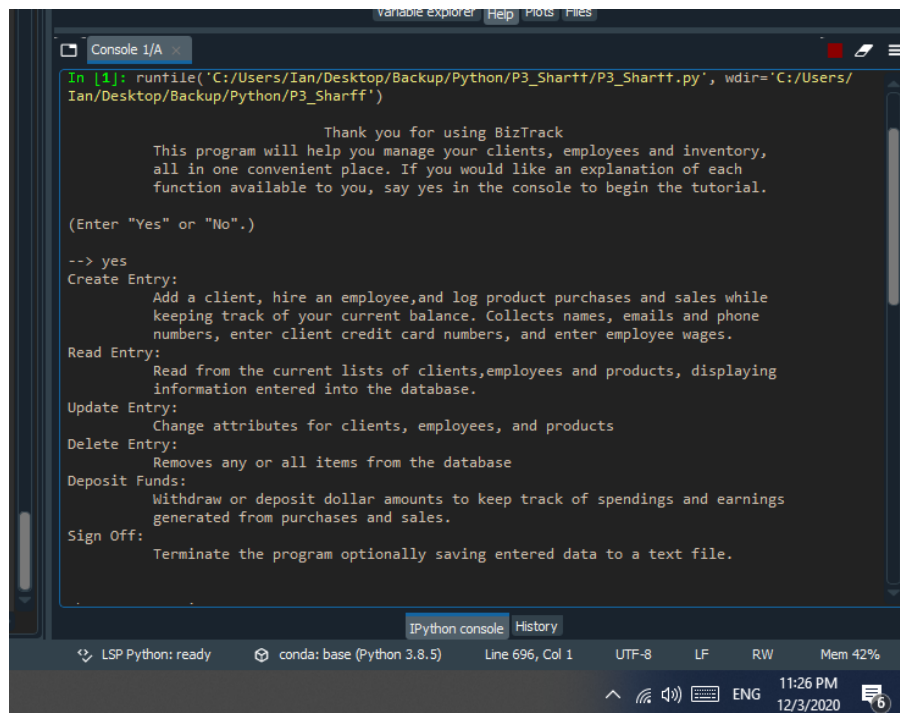
Classes were defined in the P3_Sharff_Classes.py file and are imported by the main file, P3_Sharff.py. The Person class is a parent class from which the Client and Employee classes inherit attributes (and their corresponding setter/getter methods) for last name, first name, phone number and email address. The Client subclass additionally has the option to enter credit card information, while the Employee subclass requires setting an hourly wage and has an attribute for hours worked, although this was not used in the program as it exists now. Functions were used extensively, the program allows for updating using the update_entry branch of functions, and the user can read information on given objects using the read_entry branch of functions. While loops and lists were used for each object. In hindsight, I am not sure I made use of string indexing in the sense that numbered indices were used to find an object.

To practice concepts done with independent study, I made use of regular expressions in the “re” library to check phone numbers, emails, and credit card numbers for proper input from the user. I also made use of class inheritance to simplify the Client and Employee classes , and “magic” methods like `__str__` and `__del__` to simplify displaying information about each object.

Limitations and Future Expansion Options

The program is bulky, and there are several functions within each branch that perform just about the same purpose, albeit for different object types. I found it difficult to create more abstract functions for updating and deleting objects, but they all execute well as far as I know. One item I thought could be improved was the need to pass each list into each function, even though each function generally only used one list. This was done so that I could make use of the dictionary mapping using the `make_choice()` function to cut back on repetitive code, but this introduced problems of its own. The mechanism to keep the loops working also had to be carefully designed in each function by having them return Boolean values to be returned to flag variables for while loops within the outer functions where they are called. Most significantly, I would want to make the output of the file in the form of a CSV in the future to allow for data to be accessed after the program is terminated.

Program Usage



```
variable explorer | Help | Plots | Files
Console 1/A
In [1]: runfile('C:/Users/Ian/Desktop/Backup/Python/P3_Sharff/P3_Sharff.py', wdir='C:/Users/Ian/Desktop/Backup/Python/P3_Sharff')

Thank you for using BizTrack
This program will help you manage your clients, employees and inventory,
all in one convenient place. If you would like an explanation of each
function available to you, say yes in the console to begin the tutorial.

(Enter "Yes" or "No".)

--> yes
Create Entry:
    Add a client, hire an employee, and log product purchases and sales while
    keeping track of your current balance. Collects names, emails and phone
    numbers, enter client credit card numbers, and enter employee wages.
Read Entry:
    Read from the current lists of clients, employees and products, displaying
    information entered into the database.
Update Entry:
    Change attributes for clients, employees, and products
Delete Entry:
    Removes any or all items from the database
Deposit Funds:
    Withdraw or deposit dollar amounts to keep track of spendings and earnings
    generated from purchases and sales.
Sign Off:
    Terminate the program optionally saving entered data to a text file.

IPython console | History
LSP Python: ready | conda: base (Python 3.8.5) | Line 696, Col 1 | UTF-8 | LF | RW | Mem 42%
11:26 PM
12/3/2020
```

Printing summary of each function

```

--> a
Enter client last name: Sharff
Enter client first name: Ian
Enter client phone number: 555-704-0087
Enter client email: iansharff.com
Invalid email address
Enter client email: iansharff@gmail.com
Ian Sharff was added to the database
Would you like to enter a credit card number?
(Enter "Yes" or "No".)
--> yes
Enter client credit card number: 4444-4444-4444-4444
Invalid credit card number.\Credit card numbers must be in one of the following formats:
•XXXX-XXXX-XXXX-XXXX
•16 sequential digits
Enter client credit card number: 4444-4444-4444-4444
Would you like to add another client?
(Enter "Yes" or "No".)
-->

```

Creating Client instance with credit card, invalid inputs continue loop

```

test - Notepad
File Edit Format View Help
____CLIENTS____
Ian Sharff : 5557040087 | iansharff@gmail.com | 4444444444444444
Paul Johnson : 2342348888 | paul@georgetown.edu | No card on file

____EMPLOYEES____
Walter Worker : 1111111111 | walter@gmail.com
Current Hourly Wage: $14.0
Hours Logged: 0

____ITEMS____
Bicycle | 43.0 in stock
Unit Cost: $98.0
Unit Sell Price: $100.0

____TRANSACTIONS____
-4214.0
10000
Current Balance: $ 5786.0

```

Example .txt file created by program with inputted values

__str__ method used to display item information

*Copy of .py Files***P3 Sharff Classes.py**

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Ian Sharff
COSC 010
Project 3
"""

class Person:
    """Person class containing attributes for name, phone #, email."""

    def __init__(self, lname, fname, phone, email):
        self._lname = lname
        self._fname = fname
        self._phone = phone
        self._email = email
        print(f'{self.fullname()} was added to the database')

    def lname(self, l=None):
        """Last name setter/getter."""
        if l:
            self._lname = l
        return self._lname

    def fname(self, f=None):
        """First name setter/getter."""
        if f:
            self._fname = f
        return self._fname

    def phone(self, p=None):
        """Phone number setter/getter."""
        if p:
            self._phone = p
        return self._phone

    def email(self, e=None):
        """Email setter/getter."""
        if e:
            self._phone = e
        return self._email
```

```
def fullname(self):
    """Return Person object's full name."""
    return self.fname() + ' ' + self.lname()
```

```
class Client(Person):
    """Person subclass representing clients with added attributes."""

    def __init__(self, lname, fname, phone, email, card=None):
        super().__init__(lname, fname, phone, email)
        if card is None:
            self._card = "No card on file"
        else:
            self._card = card

    def card(self, c=None):
        """Credit car number setter/getter."""
        if c: self._card = c
        return self._card

    def __str__(self):
        """Return string representation."""
        return f'{self.fullname()} : {self.phone()} | {self.email()} | {self.card()}'

    def __del__(self):
        """Print message to inform deletion of instance from list"""
        print(f'{self.fullname()} deleted from database.')
```

```
class Employee(Person):
    """Person subclass representing company employees."""

    def __init__(self, lname, fname, phone, email, wage):
        super().__init__(lname, fname, phone, email)
        self._wage = wage
        self._hours = 0

    def wage(self, w=None):
        if w: self._wage = w
        return self._wage

#Not implemented in the program, could allow for workers to log hours
    def hours(self, h=None, addition=False):
        if addition and h:
            self._hours += h
        elif h:
            self._hours = h
```

```
        return self._hours

    def payday(self, balance_list):
        paycheck = self.hours() * self.wage()
        self.hours(0)
        balance_list.append(-1 * paycheck)
        return paycheck

    def payraise(self, percent_increase):
        multiplier = percent_increase * 0.01 + 1
        self.wage(self._wage * multiplier)

    def __str__(self):
        return f'{self.fullname()} : {self.phone()} | {self.email()}\nCurrent Hourly Wage:
${self.wage()}\nHours Logged: {self.hours()}\n'

    def __del__(self):
        print(f'{self.fullname()} deleted from database.')

class Product:
    """Represent items purchased, sold or used by the company."""

    def __init__(self, name, quantity, unit_cost, unit_price):
        self._name = name
        self._quantity = quantity
        self._unit_cost = unit_cost
        self._unit_price = unit_price

    def name(self, p=None):
        if p: self._name = p
        return self._name

    def quantity(self, q=None):
        if q: self._quantity = q
        return self._quantity

    def unit_cost(self, uc=None):
        if uc: self._unit_cost = uc
        return self._unit_cost

    def unit_price(self, up=None):
        if up: self._unit_price = up
        return self._unit_price

    def buyitem(self, num):
        cost = num * self.unit_cost()
```

```

        self._quantity += num
        return cost

    def sellitem(self, num):
        self._quantity -= num
        pricetag = num * self.unit_price()
        return pricetag

    def __str__(self):
        return f'{self.name()} | {self.quantity()} in stock\nUnit Cost: ${self.unit_cost()}\nUnit Sell Price: ${self.unit_price()}\n'

    def __del__(self):
        print(f'{self.name()} item deleted from database.')

```

P3 Sharff.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Ian Sharff
COSC 010
Project 3
"""

from P3_Sharff_Classes import Client, Employee, Product
import re

def main():
    """Define empty lists, calls welcome function and terminal function."""
    client_list = []
    employee_list = []
    inventory = []
    balance = []
    welcome()
    terminal(client_list, employee_list, inventory, balance)

# MISCELLANEOUS FUNCTIONS
def yesorno():
    """Perform yes or no operation and return boolean value."""
    print('(Enter "Yes" or "No".)')
    while True:
        choice = input("--> ").capitalize()
        if choice == 'Yes':
            return True

```

```
elif choice == 'No':
    return False
else:
    print("Invalid entry. Try again.")
```

```
def check_numeric(arg):
    """Collect input and verify float type, return entry if positive and numeric."""
    while True:
        try:
            entry = float(input(arg))
            if entry < 0:
                raise ValueError
            else:
                return entry

        except ValueError:
            print("Entries must be positive and numeric")
```

```
def make_choice(dictionary):
    """Return function or list associated with key value entered by the user."""
    while True:
        choice = input("--> ").upper()
        if choice not in dictionary:
            print("Invalid entry. Try again")
            continue
        else:
            return dictionary[choice]
```

```
def check_phone(message):
    """Check entered phone number with regex pattern, return number as string."""
    while True:
        number = input(message).strip()
        if re.fullmatch(r"(\d{3})([.-]?)(\d{3})\2(\d{4})", number):
            return number.replace(".", "").replace("-", "")
        else:
            print("Invalid phone number.\nPhone numbers must be in one of the following formats:")
            print("•123-456-7890\n•123.456.7890\n•1234567890")
```

```
def check_email(message):
    """Check entered email with regex pattern, return email as string."""
    while True:
        email = input(message).rstrip().lower()
        if re.fullmatch(r"\w+@\w+\.(com|org|net|edu|gov)", email):
            return email
```



```

    else:
        print("Invalid email address")

def check_card(message):
    """Check entered card number with regex pattern, return number as string."""
    while True:
        card_number = input(message).rstrip()
        if re.fullmatch(r'(\d{4})([-]?)(\d{4})\2(\d{4})\2(\d{4})', card_number):
            return card_number.replace('-', '')
        else:
            print("Invalid credit card number.\Credit card numbers must be in one of the following formats:")
            print("•XXXX-XXXX-XXXX-XXXX\n•16 sequential digits")

def calculate_balance(b):
    """Calculate sum of balance list, return statement as string."""
    statement = "Current Balance: $ "
    if sum(b) >= 0:
        statement += str(sum(b))
    else:
        statement += f"({abs(sum(b))})"
    return statement

# MAIN FUNCTIONS
def welcome():
    """Welcome user and allow for option to print tutorial."""
    print("""
        Thank you for using BizTrack
        This program will help you manage your clients, employees and inventory,
        all in one convenient place. If you would like an explanation of each
        function available to you, say yes in the console to begin the tutorial.
        """)

    if yesorno():
        tutorial()
    else:
        print("You may now begin entering data...")

def tutorial():
    """Explain actions of each function in terminal."""
    print("""Create Entry:
        Add a client, hire an employee, and log product purchases and sales while
        keeping track of your current balance. Collects names, emails and phone
        numbers, enter client credit card numbers, and enter employee wages.""")

```

```
print("""Read Entry:
    Read from the current lists of clients,employees and products, displaying
    information entered into the database.""")

print("""Update Entry:
    Change attributes for clients, employees, and products""")

print("""Delete Entry:
    Removes any or all items from the database""")

print("""Deposit Funds:
    Withdraw or deposit dollar amounts to keep track of spendings and earnings
    generated from purchases and sales.""")

print("""Sign Off:
    Terminate the program optionally saving entered data to a text file.""")
print("\n")

def terminal(c, e, i, b):
    """Pass list parameters to be passed to functions returned by make choice."""
    working = True
    funcs = {'A': create_entry,
            'B': read_entry,
            'C': update_entry,
            'D': delete_entry,
            'E': deposit_funds,
            'F': sign_off}
    while working:
        print("Choose a Function:")
        print("A. Create Entry\nB. Read Entry\nC. Update Entry\nD. Delete Entry\nE:
Deposit/Withdraw Funds\nF. Sign Off")
        current_op = make_choice(funcs) # Function from dictionary assigned
        working = current_op(c, e, i, b) # Function called, returns boolean for loop

def create_entry(c1, e1, i1, b1):
    """Collect type of entry to be created, when done return True to loop Terminal Function."""
    creating = True
    print("Choose entry type: ")
    print("A. Add client\nB. Hire employee\nC. Purchase Item\nD. Record Sale")
    entries = {"A": add_client,
              "B": hire_employee,
              "C": buy_new_product,
              "D": log_sale}
    while creating:
```

```
    current_entry = make_choice(entries)
    creating = current_entry(c1, e1, i1, b1)
else:
    return True
```

```
def add_client(c2, e2, i2, b2):
    """Prompt user for client information, append Client instance to list."""
    while True:
        cli_lname = input("Enter client last name: ").capitalize()
        cli_fname = input("Enter client first name: ").capitalize()
        cli_phone = check_phone("Enter client phone number: ")
        cli_email = check_email("Enter client email: ")

        cli_obj = Client(cli_lname, cli_fname, cli_phone, cli_email)
        print("Would you like to enter a credit card number?")
        if yesorno():
            cli_card = check_card("Enter client credit card number: ")
            cli_obj.card(cli_card)
        else:
            print("No credit card recorded")
        c2.append(cli_obj)
        print("Would you like to add another client?")
        if yesorno():
            print("Adding another client...")
        else:
            print("Returning to Terminal...")
            return False
```

```
def hire_employee(c2, e2, i2, b2):
    """Prompt user for employee information, append Employee instance to list."""
    while True:
        emp_lname = input("Enter employee last name: ").capitalize()
        emp_fname = input("Enter employee first name: ").capitalize()
        emp_phone = check_phone("Enter employee phone number: ")
        emp_email = check_email("Enter employee email: ")
        emp_wage = check_numeric("Enter hourly wage: ")
        emp_obj = Employee(emp_lname, emp_fname, emp_phone, emp_email, emp_wage)

        e2.append(emp_obj)
        print("Would you like to hire another employee?")
        if yesorno():
            continue
        else:
            print("Returning to Terminal...")
```

```
return False
```

```
def buy_new_product(c2, e2, i2, b2):
    """Prompt user for product information, append Product instance to list."""
    while True:
        prod_name = input("Enter product name: ")
        prod_quantity = check_numeric("Enter quantity purchased: ")
        prod_cost = check_numeric("Enter product's unit cost: ")
        prod_price = check_numeric("Enter product's unit sell price: ")
        prod_obj = Product(prod_name, prod_quantity, prod_cost, prod_price)

        i2.append(prod_obj)
        b2.append(-1 * prod_cost * prod_quantity)
        print(calculate_balance(b2))
        print("Would you like to purchase another product?")
        if yesorno():
            continue
        else:
            print("Returning to Terminal...")
            return False

def log_sale(c2, e2, i2, b2):
    """Search item list for object with entered name, adds earnings to balance."""
    sold_item = None
    while sold_item is None:
        namesearch = input("Enter name of item to sell: ")
        for obj in i2:
            if obj.name() == namesearch:
                print(f"{namesearch} found")
                sold_item = obj
                break
        else:
            print(f"{namesearch} not found\nNames are case-sensitive.")
            print("Keep searching?")
            if yesorno():
                continue
            else:
                print("Returning to Terminal...")

    amt_sold = check_numeric(f"Enter quantity of {sold_item.name()} sold: ")
    earnings = sold_item.sellitem(amt_sold)
    b2.append(earnings)
    print(f"{amt_sold} {sold_item.name()} items were sold for ${earnings}")
    print(calculate_balance(b2)) # Calls function to display balance
```

```
print("Would you like to log another sale?")
if yesorno():
    continue
else:
    print("Returning to Terminal...")
    return False
```

```
def read_entry(c1, e1, i1, b1):
    """Read list of objects or calls print_ledger function accordingly."""
    if not (c1 or e1 or i1 or b1): # True if at least one list not empty
        print("All lists are empty, data must be entered first.")
        print("Returning to Terminal...")
        return True

    while True:
        print("Choose entry type to read: ")
        print("A. Client\nB. Employee\nC. Item\nD. Print Balance")
        types = {"A": c1, # lists of objects for read_list
                 "B": e1,
                 "C": i1,
                 "D": print_ledger} # print_ledger function for balance
        choice = make_choice(types)
        if isinstance(choice, list): # if list type, calls read list
            read_list(choice)
        else:
            choice(b1) # if not list, calls print_ledger passing balance list
        print("Read another list?") # prompts user to read another list
        if yesorno():
            continue
        else:
            print("Returning to Terminal...")
            return True
```

```
def read_list(li):
    """Pass list object, output object string representations to console."""
    if li:
        for obj in li:
            print(obj)
    else:
        print("List is empty")
```

```
def print_ledger(transaction_list):
    """Pass list of transactions, outputs transactions, in parentheses are negative."""
```

```
print("Transactions: ")
for transaction in transaction_list:
    if transaction >= 0:
        print(f"$ {transaction}")
    else:
        print(f"$ ({abs(transaction)})")
print(calculate_balance(transaction_list))
```

```
def update_entry(c1, e1, i1, b1):
    """Prompt user to choose type of entry to update."""
    updating = True
    while updating:
        print("Choose entry type to update: ")
        print("A. Clients\nB. Employees\nC. Items")
        object_types = {"A": edit_client,
                        "B": edit_employee,
                        "C": edit_item,}
        current_update = make_choice(object_types)
        updating = current_update(c1, e1, i1, b1)
    else:
        return True
```

```
def edit_client(c2, e2, i2, b2):
    """Search list of clients by full name, change attribute of Client instance in list."""
    if not c2: # True only if list not empty
        print("No clients found in database. Returning to Terminal...")
        return False
    cli_edit = None
    while cli_edit is None: # True when object found
        clientsearch = input("Enter the client's listed full name: ").title()
        for obj in c2:
            if clientsearch == obj.fullname():
                print(f"{clientsearch} found.") # States client was found
                cli_edit = obj # Assigns Client object to variable to edit
                break
        else: # Runs only if for loop completed without finding client full name
            print(f"{clientsearch} not found. Entries are case-sensitive.")
            print("Keep searching?") # Allows user to continue searching
            if yesorno():
                continue
            else:
                print("Returning to Terminal...") # If not returns to Terminal
                return False
```

```
print("Choose information to change: ")
print("A. Last Name\nB. First Name\nC. Phone Number\nD. Email Address\nE. Credit Card")
```

```
while True: # Prompts user attribute to change attributes
    choice = input("-->").upper()
    # Depending on choice, calls associated class method to change attribute
    if choice == "A":
        cli_edit.lname(input("Enter new last name: "))
    elif choice == "B":
        cli_edit.fname(input("Enter new first name: "))
    elif choice == "C":
        cli_edit.phone(check_phone("Enter new phone number: "))
    elif choice == "D":
        cli_edit.email(check_email("Enter new email address: "))
    elif choice == "E":
        cli_edit.card(check_card("Enter new credit card number: "))
    else:
        print("Invalid entry. Try again") # If not valid letter, continues looping
        continue
    print(f"Client updated:\n{cli_edit}")
    print(f"Change another attribute for {cli_edit.fullname()}?")
    if yesorno():
        continue
    else:
        print("Returning to Terminal...")
        return False
```

```
def edit_employee(c2, e2, i2, b2): # Same functionality as previous function
    """Search list of employees by full name, change attribute of Employee instance in list."""
    if not e2:
        print("No employees found in database. Returning to Terminal...")
        return False
```

```
emp_edit = None
while emp_edit is None:
    employee_search = input("Enter the client's listed full name: ").title()
    for obj in e2:
        if employee_search == obj.fullname():
            print(f"{employee_search} found.")
            emp_edit = obj
            break
    else:
        print(f"{employee_search} not found. Entries are case-sensitive.")
        print("Keep searching?")
        if yesorno():
```

```

        continue
    else:
        print("Returning to Terminal...")
        return False

while True:
    print("Choose information to change: ")
    print("A. Last Name\nB. First Name\nC. Phone Number\nD. Email Address\nE. Give
Raise\nF. Clock Hours\nG. Give Paycheck")
    choice = input("-->").upper()
    if choice == "A":
        emp_edit.lname(input("Enter new last name: "))
    elif choice == "B":
        emp_edit.fname(input("Enter new first name: "))
    elif choice == "C":
        emp_edit.phone(check_phone("Enter new phone number: "))
    elif choice == "D":
        emp_edit.email(check_email("Enter new email address: "))
    elif choice == "E":
        emp_edit.payraise(check_numeric("Enter percent raise to be given: "))
    elif choice == "F":
        emp_edit.hours(check_numeric("Enter added hours worked: "), True)
    elif choice == "G":
        emp_edit.payday(b2)
    else:
        print("Invalid entry. Try again.")
        continue

    print(f"Updated Employee:\n{emp_edit}")
    print(f"Change another attribute for {emp_edit.fullname()}?")
    if yesorno():
        continue
    else:
        print("Returning to Terminal...")
        return False

def edit_item(li): # Same functionality as previous function
    """Search list of items by name, change attribute of Product instance in list."""
    if not li:
        print("No items found in database. Returning to Terminal...")
        return False

    item_edit = None
    while item_edit is None:
        item_search = input("Enter item name: ")
        for obj in li:

```



```

        if item_search == obj.fullname():
            print(f"{item_search} found.")
            item_edit = obj
            break
    else:
        print(f"{item_search} not found. Entries are case-sensitive.")
        print("Keep searching?")
        if yesorno():
            continue
        else:
            print("Returning to Terminal...")
            return True
while True:
    print("Choose information to change: ")
    print("A. Last Name\nB. First Name\nC. Phone Number\nD. Email Address\nE. Give
Raise\nF. Clock Hours\nG. Give Paycheck")
    choice = input("-->").upper()
    if choice == "A":
        item_edit.lname(input("Enter new last name: "))
    if choice == "B":
        item_edit.fname(input("Enter new first name: "))
    if choice == "C":
        item_edit.phone(check_phone("Enter new phone number: "))
    if choice == "D":
        item_edit.email(check_email("Enter new email address: "))
    if choice == "E":
        item_edit.payraise(check_numeric("Enter percent raise to be given: "))
    else:
        print("Invalid entry. Try again.")
        continue

    print(f"Item Updated:\n{item_edit}")
    print(f"Change another attribute for {item_edit.fullname()}?")
    if yesorno():
        continue
    else:
        print("Returning to Terminal...")
        return False

def delete_entry(c1, e1, i1, b1):
    """Prompt user for data type to delete, call associated function."""
    deleting = True
    while deleting:
        print("Choose entry type to delete: ")
        print("A. Clients\nB. Employees\nC. Items\nD. Clear Database")
        object_types = {"A": delete_client,

```

```
        "B": delete_employee,
        "C": delete_item,
        "D": clear_database} # Option to delete all information
    current_deletion = make_choice(object_types)
    deleting = current_deletion(c1, e1, i1, b1)
else:
    return True

def delete_client(c2, e2, i2, b2):
    """Search client list by full name, delete Client instance."""
    if not c2: # True only if list not empty
        print("No clients in database to delete.")
        print("Returning to Terminal...")
        return False

    print("Delete all clients?") # Option to delete all clients
    if yesorno():
        c2.clear()
        print("Client database cleared.")
        print("Returning to Terminal...")
        return False
    while True:
        print("Enter full name of client to delete: ") # Searches by fullname
        clientsearch = input("-->").title()
        for obj in c2:
            if clientsearch == obj.fullname():
                cli_del = obj
                print(f"Client found:\n{cli_del}")
                break
        else:
            print("Client not found. Keep searching?")
            if yesorno():
                continue
            else:
                print("Returning to Terminal...")
                return False
        print(f"Are you sure you want to delete {cli_del.fullname()}?")
        if yesorno(): # If confirmed, then object is deleted, displaying destructor message
            c2.remove(cli_del)
        else:
            print("Client not deleted.") # Otherwise, does not delete and returns to terminal()

    print("Returning to Terminal")
    return False
```

```
def delete_employee(c2, e2, i2, b2): # Same functionality as above
    """Search employee list by full name, delete Employee instance."""
    if not e2:
        print("No employees in database to delete.")
        print("Returning to Terminal...")
        return False

    print("Delete all employees?")
    if yesorno():
        e2.clear()
        print("Employee database cleared.")
        print("Returning to Terminal...")
        return False
    while True:
        print("Enter full name of employee to delete: ")
        employeesearch = input("-->").title()
        for obj in e2:
            if employeesearch == obj.fullname():
                emp_del = obj
                print(f"Employee found:\n{emp_del}")
                break
        else:
            print("Employee not found. Keep searching?")
            if yesorno():
                continue
            else:
                print("Returning to Terminal...")
                return False
        print(f"Are you sure you want to delete {emp_del.fullname()}?")
        if yesorno():
            e2.remove(emp_del)
        else:
            print("Employee not deleted.")

        print("Returning to Terminal")
        return False

def delete_item(c2, e2, i2, b2): # Same functionality as above
    """Search item list by full name, delete Product object."""
    if not i2:
        print("No items in database to delete.")
        print("Returning to Terminal...")
        return False

    print("Delete all items?")
    if yesorno():
```

```
i2.clear()
print("Item database cleared.")
print("Returning to Terminal...")
return False
while True:
    print("Enter name of item to delete: ")
    itemsearch = input("-->")
    for obj in i2:
        if itemsearch == obj.fullname():
            item_del = obj
            print(f"Item found:\n{item_del}")
            break
    else:
        print("Item not found. Keep searching?")
        if yesorno():
            continue
        else:
            print("Returning to Terminal...")
            return False
    print(f"Are you sure you want to delete {item_del.fullname()}?")
    if yesorno():
        i2.remove(item_del)
    else:
        print("Item not deleted")

    print("Returning to Terminal")
    return False

def clear_database(*args):
    """Delete all objects from all lists."""
    print("Are you sure you want to clear all stored data?")
    if yesorno():
        print("Like really sure?")
        if yesorno():
            for li in args:
                li.clear()
            print("Done. All data deleted.")
    print("Returning to Terminal...")
    return False

def deposit_funds(c1, e1, i1, b1):
    """Add postive or negative transaction to balance list."""
    sign = 0
    transaction = "
    while not sign: # Since 0 returns False
```

```

print("Deposit or Withdraw?")
choice = input("-->").title()
if choice == "Withdraw":
    sign = -1 # When sign defined, exits loop
    transaction = 'withdrawn'
if choice == "Deposit":
    sign = 1
    transaction = 'deposited'
else: # if neither, continues looping
    print("Invalid entry: Choose Deposit or Withdraw")
while True:
    try:
        quant = eval(input("Enter quantity: "))
        if quant < 0:
            raise ValueError
        break
    except ValueError:
        print("Please enter a positive numeric value")
        continue
    b1.append(quant * sign)
    print(f"$ {quant} {transaction}.")
    print(calculate_balance(b1))
    print("\nMake another entry?")
    if yesorno():
        sign = 0
    else:
        print("Returning to Terminal...")
        return True

```

```

def sign_off(c1, e1, i1, b1):
    """Ask user to save data, calling write_file function."""
    print("""
        WARNING: Data not saved.
        Would you like to save
        your data as a text file?
        """)
    if yesorno():
        write_file(c1, e1, i1, b1)
    print("Thank you,")
    print("Signing off...")
    return False # Returning False will stop the while loop in Terminal and end the program

```

```

def write_file(c2, e2, i2, b2):
    """Write objects to text file named as user input."""

```

```
filename = input("Create file name without an extension: ") + ".txt"
with open(filename, 'w') as f:
    f.write("____CLIENTS____\n")
    for client in c2:
        f.write(str(client) + "\n") # objects in lists printed using their string representations
    else:
        f.write("\n")

    f.write("____EMPLOYEES____\n")
    for employee in e2:
        f.write(str(employee) + "\n")
    else:
        f.write("\n")

    f.write("____ITEMS____\n")
    for item in i2:
        f.write(str(item) + "\n")
    else:
        f.write("\n")

    f.write("____TRANSACTIONS____\n")
    for transaction in b2:
        f.write(str(transaction) + "\n")
    else:
        f.write(calculate_balance(b2))
        f.write("\n")

print(f"{filename} saved.")

if __name__ == '__main__': main()
```