Module Code & Module Title

CS4051NI Fundamentals of Computing

Assessment Weightage & Type

60% Individual Coursework

Year and Semester

2021-22 Summer

Student Name: Priyans Bhatt

Group: C10

London Met ID: 21049638

College ID: np01cp4a210219

Assignment Due Date:26 August, 2022

Assignment Submission Date:26 August 2022

Table of Contents

Table of Contents

List of Figures

List to Tables

Introduction

A well-liked programming language is Python. In 1991, Guido van Rossum produced it, and it became available.

It's used for website creation, software creation, mathematics, scripting for systems.

Python is used for:

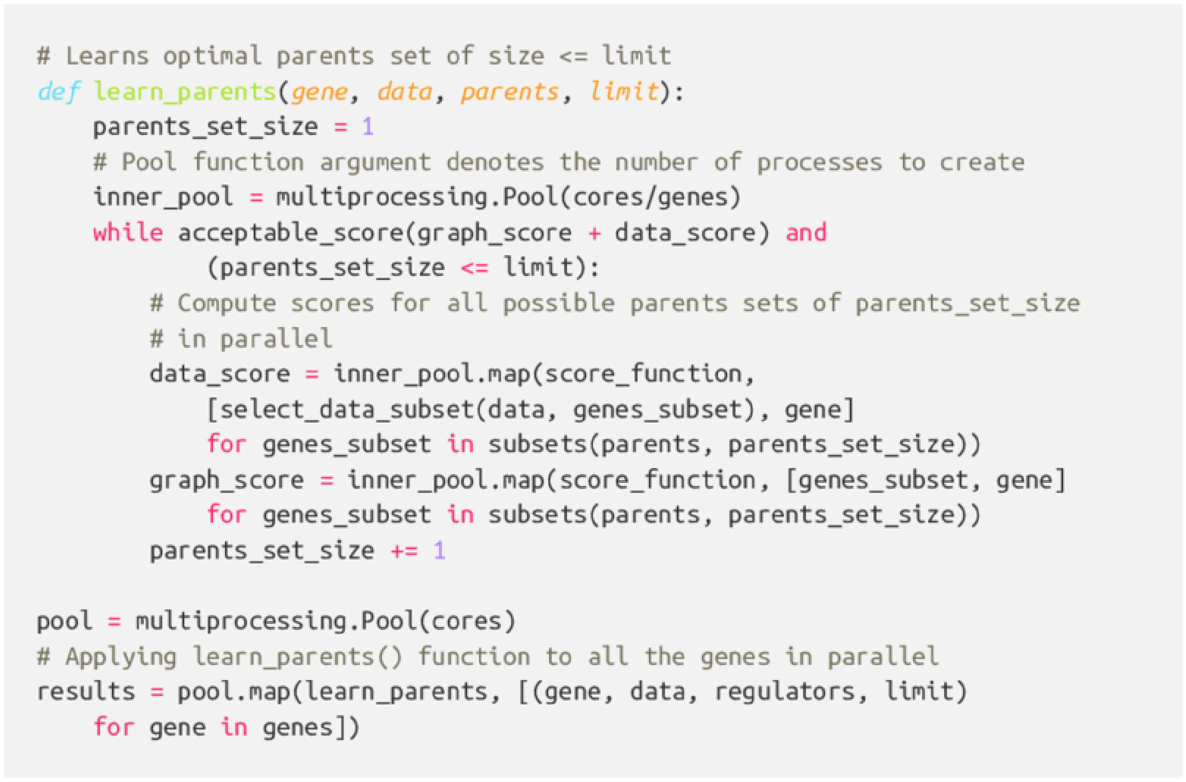
1)Web applications can be developed on a server using Python.
2)Workflows can be made with Python and other technologies.
3)Database systems are connectable with Python. Files can also be read and changed by it.
4)Big data management and advanced mathematical operations can both be done with Python.
5)Python can be used to produce software that is ready for production or for rapid prototyping.

Figure 1: Picture of Python

Python is cross-platform compatible. The syntax of Python is straightforward and resembles that of English. Python’s syntax differs from various other programming languages in that it enables programmers to construct applications with fewer lines of code. Python operates on an interpreter system, allowing for the immediate execution of written code. As a result, prototyping can proceed quickly. Python can be used in a functional, object-oriented, or procedural manner.

Algorithms.

An algorithm is a set of instructions for completing a task or solving a conundrum. A common example of an algorithm is a recipe, which includes thorough instructions for making a dish or meal. Every piece of computerized equipment uses algorithms, whether they be in the form of software or hardware routines, to carry out its activities.



```
# Learns optimal parents set of size <= limit
def learn_parents(gene, data, parents, limit):
    parents_set_size = 1
    # Pool function argument denotes the number of processes to create
    inner_pool = multiprocessing.Pool(cores/genes)
    while acceptable_score(graph_score + data_score) and
            (parents_set_size <= limit):
        # Compute scores for all possible parents sets of parents_set_size
        # in parallel
        data_score = inner_pool.map(score_function,
            [select_data_subset(data, genes_subset), gene]
            for genes_subset in subsets(parents, parents_set_size))
        graph_score = inner_pool.map(score_function, [genes_subset, gene]
            for genes_subset in subsets(parents, parents_set_size))
        parents_set_size += 1

pool = multiprocessing.Pool(cores)
# Applying learn_parents() function to all the genes in parallel
results = pool.map(learn_parents, [(gene, data, regulators, limit)
    for gene in genes])
```

Figure 2: Algorithm

The algorithm of this project is listed below :

Step 1: Start.

Step 2: Access the desired variables in a csv database.

Step 3: Post greetings or a banner.

Step 4: Go to Step 5 if the item has already been hired; otherwise, go to Step 9.

Step 5: Access the desired local variables from the desired global variables.

Step 6: Step-9 should be taken if rents.db contains nothing; else, Step 7.

Step 7: If today is more recent than the item return date for each record in rents.db, then ask the user if the consumer has responded; if so, go Step-31

Step 8: Immediately go to Step 19 if any of the items have been returned; otherwise, go to Step 9.

Step 9: Access the desired local variables from the desired global variables.

Step 10: Prompt user available ID of an item.

Step 11: Allow entry to be obtained from the DB using the requested ID.

Step 12: Prompt user quantity of item to rent.

Step 13: Reduce rent quantity from DB according to the prompt from above.

Step 14: Rental unit days should be prompt (1 unit day = 5 days).

Step 15: Let rent quantity from, rental unit days from prompt, and total price be a multiplication of the price from entry.

Step 16: Display the invoice together with the entry and the calculations above.

Step 17: Final rental confirmation; if yes, move on to step 18; otherwise, move on to step 36.

Step 18: Add fresh updates to the csv database.

Step 19: Access local variables and a csv database.

Step 20: Please provide the item's return ID.

Step 21: Allow entity from accessible DB to be parsed using prompted ID.

Step 22: Calculate the total cost from the entries by multiplying the entry's price by the rent quantity and the rental unit's days from the database.

Step 23: If the return date is the same as today, go on to Step 24, if it is early, move on to Step 26, and if it is late, move on to Step 31.

Step 24: Display Invoice.

Step 25: Commit modifications within csv files, goto Step-1

Step 26: Access the locals' csv DB..

Step 27: If you need to lower the price in the event of an early return, move to Step-28; otherwise, go to Step-29. To locals, DB

Step 28: Subtract the anticipated cost of the days not used from the total cost.

Step 29: Display Invoice.

Step 30: Commit modifications within csv files, goto Step-36

Step 31: Access csv DB, to locals.

Step 32: prompt if late return fee is necessary and if no go-to Step-34

Step 33: By figuring out the cost each day per unit of the item, add the fine price to the overall cost.

Step 34: Display Invoice.

Step 35: Commit modifications within csv files.

Step 36: Ask the user if they wish to stop; if not, move on to Step 1.

Step 37: Stop.

Flowchart

The visual representation of a program's flow is a flowchart. A flow chart typically uses four basic shapes. An illustration of a process' individual steps in chronological sequence is called a flowchart. It is a versatile tool that may be customized for a wide range of uses and used to describe a number of processes, including manufacturing processes, administrative or service processes, project plans, and others. It is a typical tool for process analysis and one of the seven fundamental quality tools. Flowcharts, often known as flow charts, use forms like rectangles, ovals, diamonds, and perhaps many more to identify the type of step and connecting arrows to denote flow and sequence. Flowcharts are among the most frequently used diagrams in the world, being utilized in a variety of industries by both technical and non-technical people.

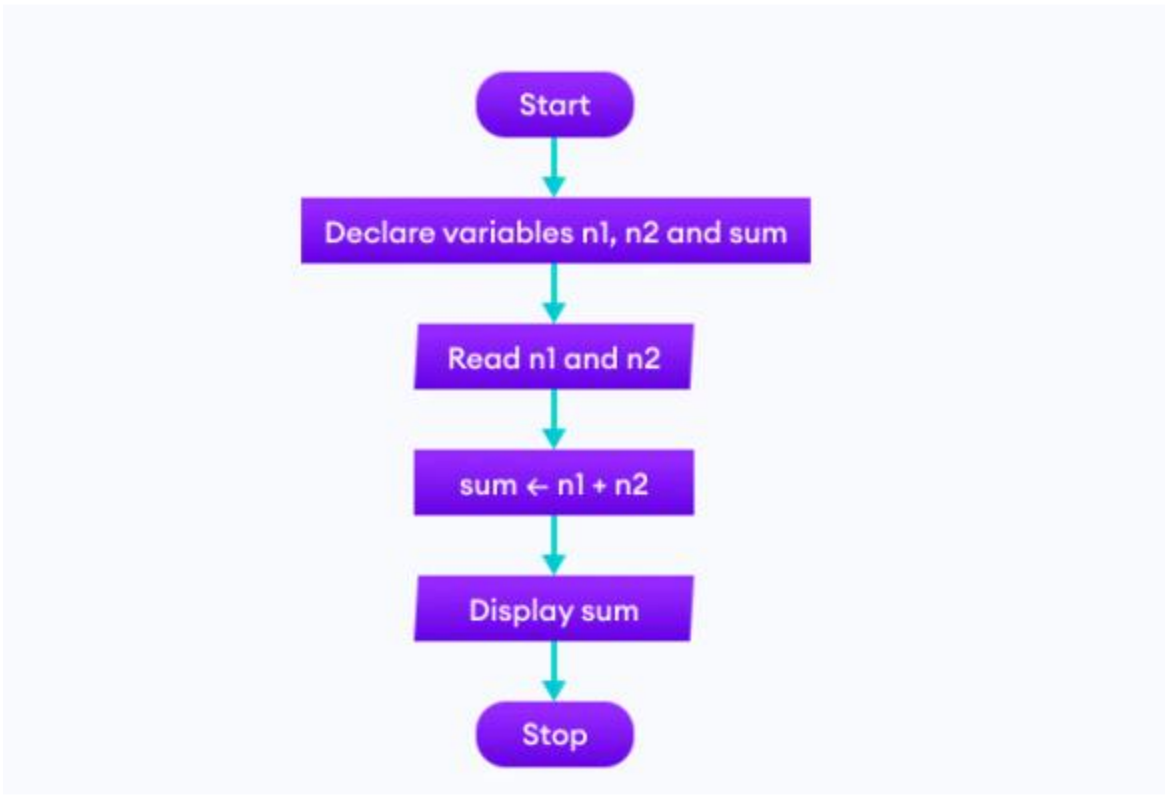
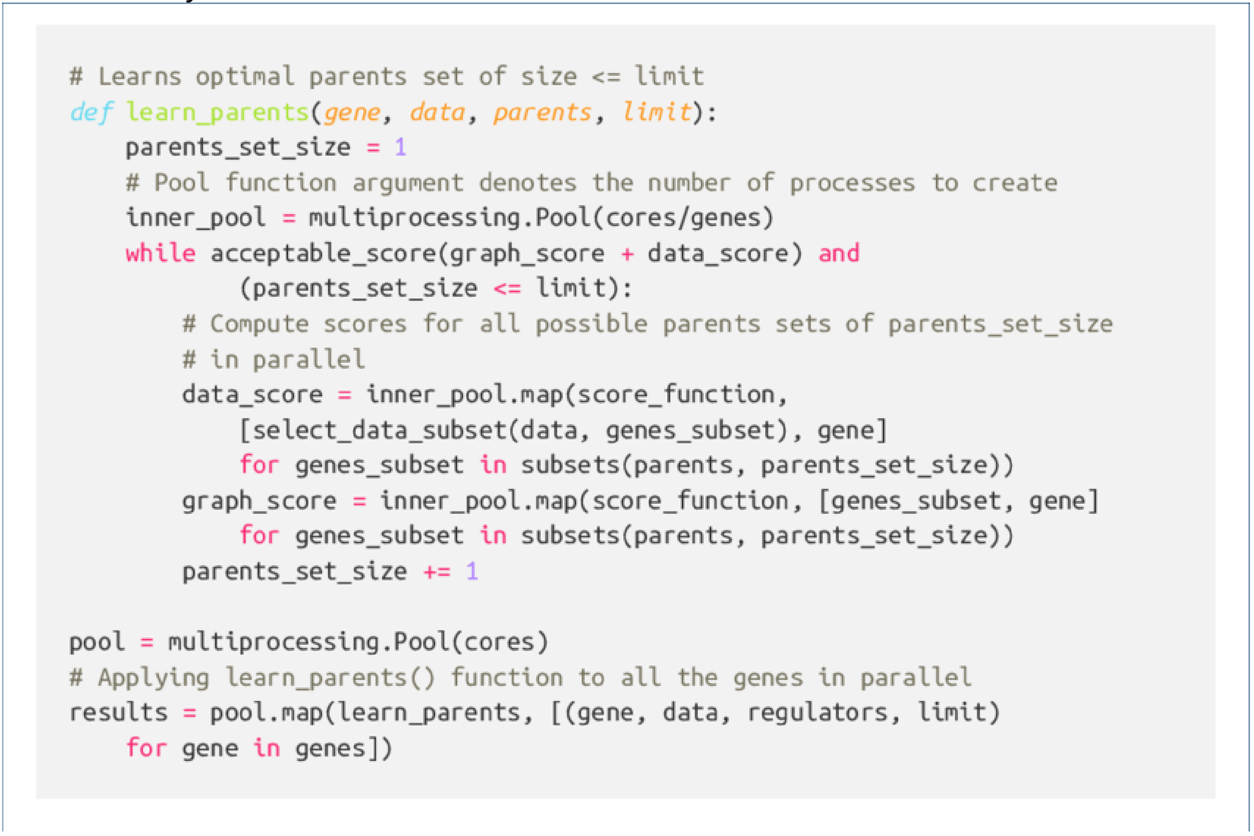


Figure 3: Flowchart to add two numbers

Pseudocode

Pseudocode is a made-up, informal language that aids in the creation of algorithms by programmers. A "text-based" detail (algorithmic) design tool is pseudocode. The Pseudocode rules are not too complicated. Statements that demonstrate "dependence" must all be indented. Some of these are while, do, for, if, and switch. It is employed to create the rough draft or design for a program. Pseudocode streamlines a program's flow but leaves out supporting details. System designers produce pseudocode to ensure that programmers understand the requirements of a software project and align their code correctly.



```
# Learns optimal parents set of size <= limit
def learn_parents(gene, data, parents, limit):
    parents_set_size = 1
    # Pool function argument denotes the number of processes to create
    inner_pool = multiprocessing.Pool(cores/genes)
    while acceptable_score(graph_score + data_score) and
            (parents_set_size <= limit):
        # Compute scores for all possible parents sets of parents_set_size
        # in parallel
        data_score = inner_pool.map(score_function,
            [select_data_subset(data, genes_subset), gene]
            for genes_subset in subsets(parents, parents_set_size))
        graph_score = inner_pool.map(score_function, [genes_subset, gene]
            for genes_subset in subsets(parents, parents_set_size))
        parents_set_size += 1

pool = multiprocessing.Pool(cores)
# Applying learn_parents() function to all the genes in parallel
results = pool.map(learn_parents, [(gene, data, regulators, limit)
    for gene in genes])
```

Figure 4: Example of Pseudo Code

```
func prompt_yn(txt):
  _input = input(_args_0 + " [y/n]: ").lower()
  if _input not in ('y', 'n'):
    _input = prompt_yn(txt)
  if choi_input == 'y':
    return True
  return False

func handle_known_rentals:
  if not DB.rents: return
  for entry in DB.rents:
    if datetime.date.today() > entry.return_date:
      print("You might want to see this entry: " + entry)
      if prompt_yn("Did he answer?/Did you call him?"):
        handle_late_rental(entry.id)
  if prompt_yn("Was any stuff returned?"):
    handle_returns()
  else:
    handle_new_rentals()

func handle_new_rentals():
  table_print(DB.stocks)
  id = input("Enter ID of item you want to rent: ")
  db_entry = DB.stocks[id]
  rent_qty = input("Enter quantity you wish to rent: ")
  entry.qty -= rent_qty
  rent_unit_days = input("Enter unit days you want to rent (1 unit day = 5 days): ")
  total_price = entry.price * rent_qty * rent_unit_days
  _print_invoice_
  if cofirm("Do you want to rent now?"):
    name = input("Enter you name: ")
    contact = input("Enter you phone number: ")
    DB.commit(entry, [today(), rent_qty, rent_unit_days, name, contact])


func handle_returns():
  print_table(DB.rents)
  id = input("Enter the ID of item that had been returned")
  entry = DB.rents[id]
  stock_entry = DB.stocks[entry.IID]
  total_price = stock_entry.price * entry.rent_qty * entry.rent_unit_days
  _print_invoice_
  if entry.return_date == datetime.date.today():
    stock_entry.qty += entry.qty
    commit(rm(entry), stock_entry)
  elif (entry.return_date - today).days > 0:
    handle_early_return(id)
  else:
    handle_new_return(id)

func handle_late_rental(id):
  entry = DB.rents[id]
  stock_entry = DB.stocks[entry.IID]
  total_price = stock_entry.price * entry.rent_qty * entry.rent_unit_days
  days_late = (datetime.date.today() - entry.return_date).days
  if prompt_yn("Do you want to fine the costumer?"):
    fine_price = ( stock_entry.price / 5 ) * entry.qty * days_late
    total_price += fine_price
  _print_invoice_

function handle_early_return(id):
  entry = DB.rents[id]
  stock_entry = DB.stocks[entry.IID]
  total_price = stock_entry.price * entry.rent_qty * entry.rent_unit_days
  days_early = ( entry.return_date - datetime.date.today() ).days
  if prompt_yn("Do you want to deduct price from the costumer (un-used days)?"):
    discount_price = ( stock_entry.price / 5 ) * entry.qty * days_early
    total_price -= discount_price
  _print_invoice_

func main():
  print("Welcome - banner")
  print_table(DB.stocks)
  if DB.rent:
    handle_known_rentals()
  else:
    handle_new_return()
  if prompt_yn("Do you wish to exit?"):
    exit(0)
  else:
    main()

main()
```

Data Structure

A data structure is not just employed for data organization. Data processing, retrieval, and archiving are further uses for it. Almost every software system or program that has been created uses many fundamental and sophisticated forms of data structures. Therefore, we need to be knowledgeable about data structures. A data structure may be chosen or created in computer science and computer programming to store data in order to be used with different methods. In some circumstances, the design of the data structure and the algorithm's fundamental operations are closely related. Each data structure includes details about the values of the data, the connections between the data, and, occasionally, functions that can be used to manipulate the data.

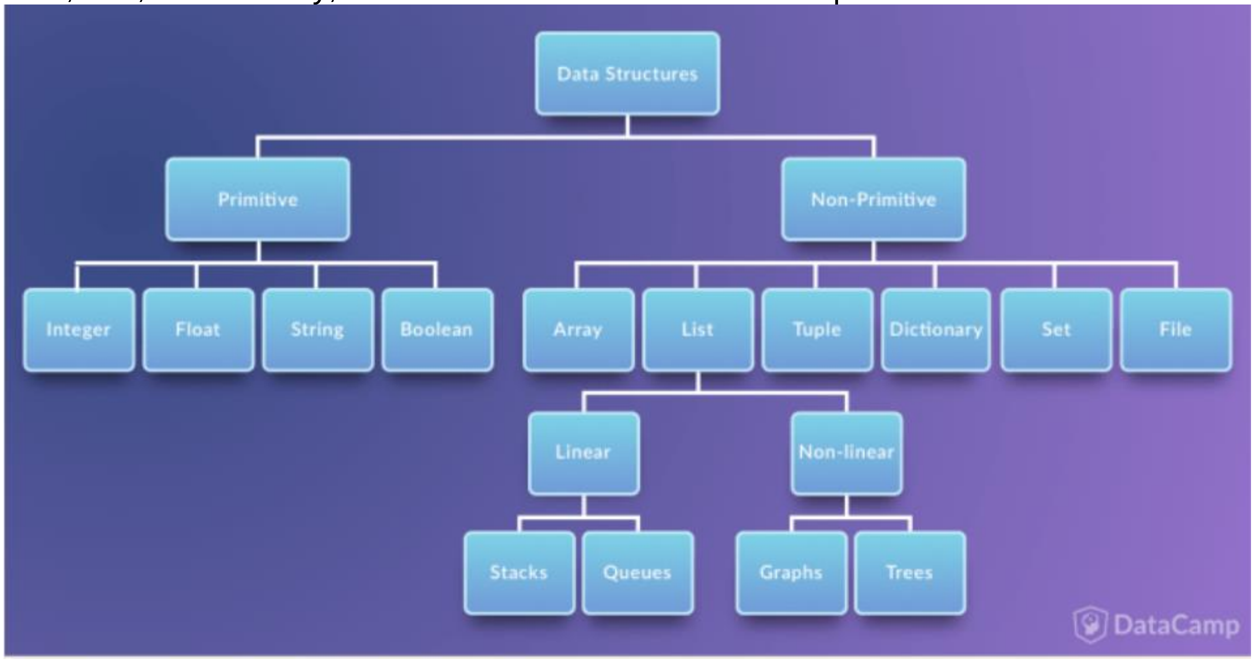
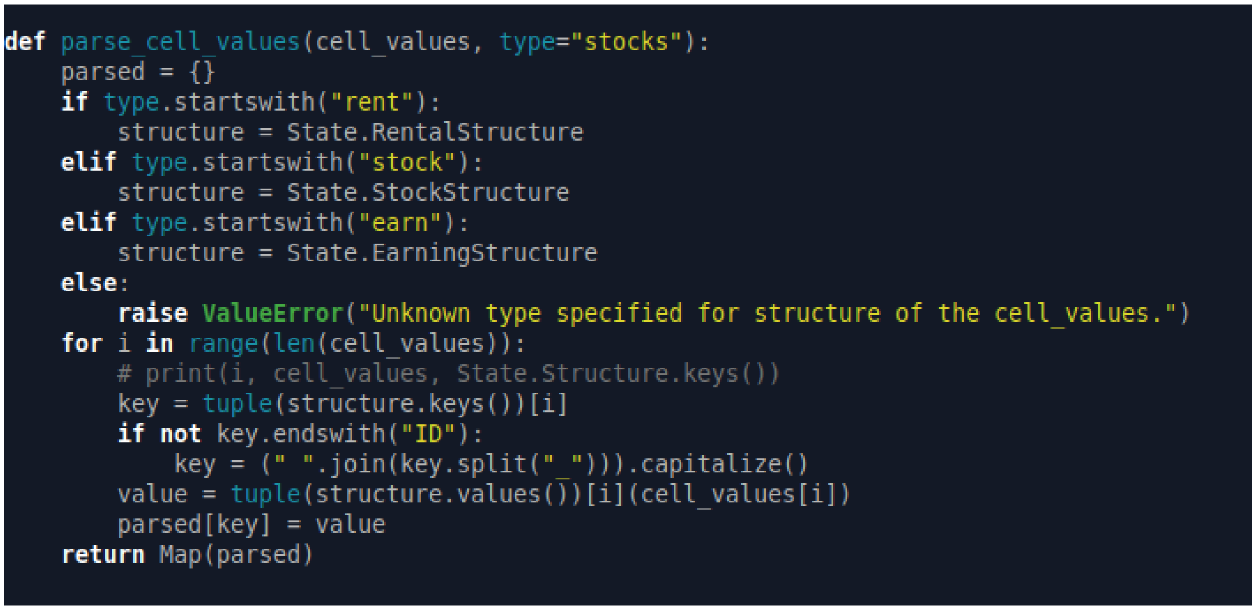
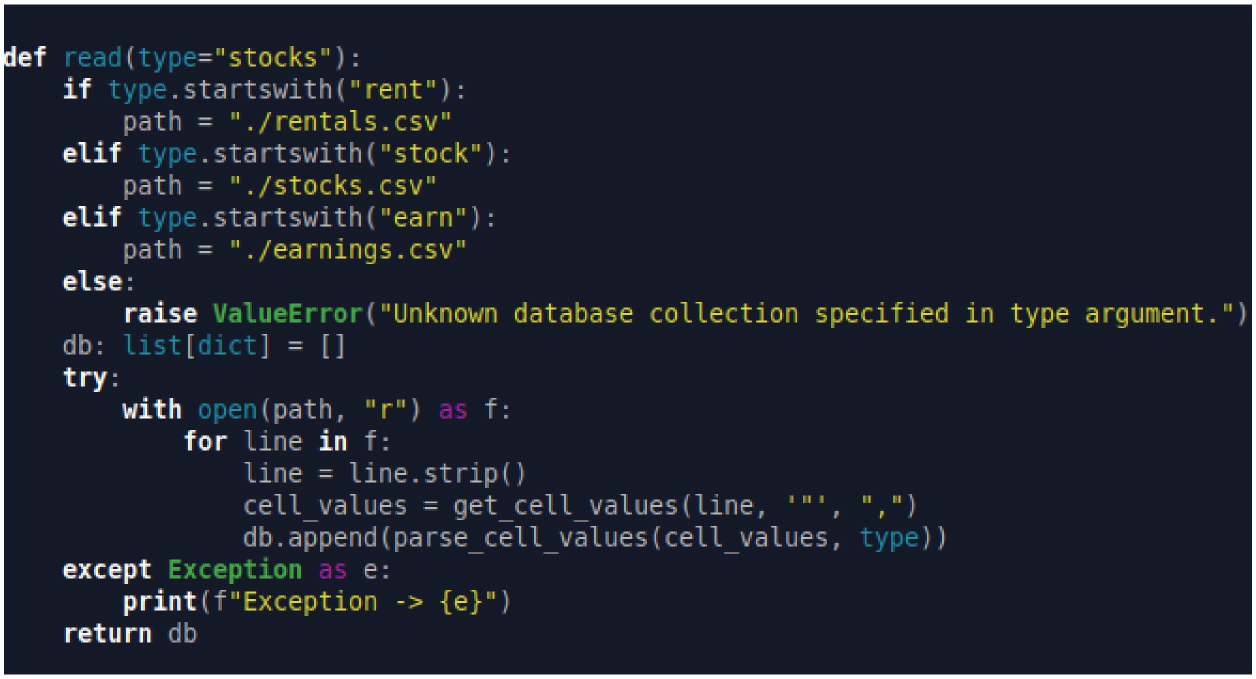


Figure 5: Data Structure Explained



```python
def parse_cell_values(cell_values, type="stocks"):
    parsed = {}
    if type.startswith("rent"):
        structure = State.RentalStructure
    elif type.startswith("stock"):
        structure = State.StockStructure
    elif type.startswith("earn"):
        structure = State.EarningStructure
    else:
        raise ValueError("Unknown type specified for structure of the cell_values.")
    for i in range(len(cell_values)):
        # print(i, cell_values, State.Structure.keys())
        key = tuple(structure.keys())[i]
        if not key.endswith("ID"):
            key = (" ".join(key.split("_"))).capitalize()
        value = tuple(structure.values())[i](cell_values[i])
        parsed[key] = value
    return Map(parsed)
```

An ordered group of objects that can be updated or modified is referred to as a list in Python. Any elements or values present within a list are considered its items. Values must be enclosed in square brackets [to define a list, just as characters must be enclosed in quotation marks to define a string].



```python
def read(type="stocks"):
    if type.startswith("rent"):
        path = "./rentals.csv"
    elif type.startswith("stock"):
        path = "./stocks.csv"
    elif type.startswith("earn"):
        path = "./earnings.csv"
    else:
        raise ValueError("Unknown database collection specified in type argument.")
    db: list[dict] = []
    try:
        with open(path, "r") as f:
            for line in f:
                line = line.strip()
                cell_values = get_cell_values(line, '"', ",")
                db.append(parse_cell_values(cell_values, type))
    except Exception as e:
        print(f"Exception -> {e}")
    return db
```

Dictionary:A dictionary is an implementation of the associative array data structure in Python. A set of key-value pairs constitutes a dictionary. Each key-value pair represents a key and the value it relates to.

By enclosing it in curly brackets, you can define a list of key-value pairs as a dictionary (). There is a colon between each key and its matching value (:).

The entire project does not utilize it.

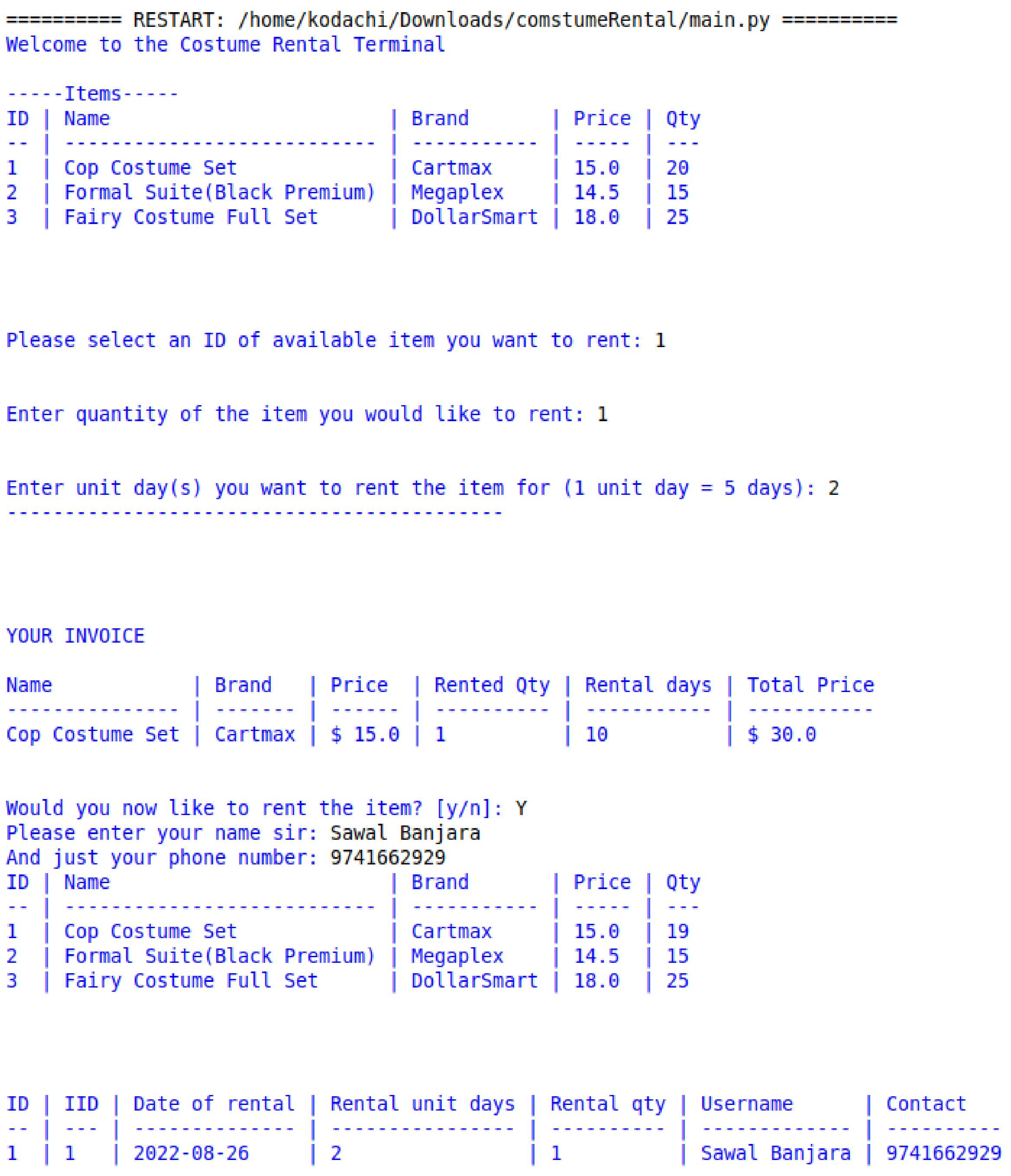
Python, plural A tuple is a group of items with commas separating them. A tuple is somewhat akin to a list in terms of indexing, nested items, and repetition, however a tuple is immutable as opposed to mutable lists.

Unordered items are placed together to form a set. Each set element must be immutable, without any duplication (cannot be changed). However, a set can be altered on its own. We can add or remove stuff from it. Union, intersection, symmetric difference, and other mathematical set operations

Program

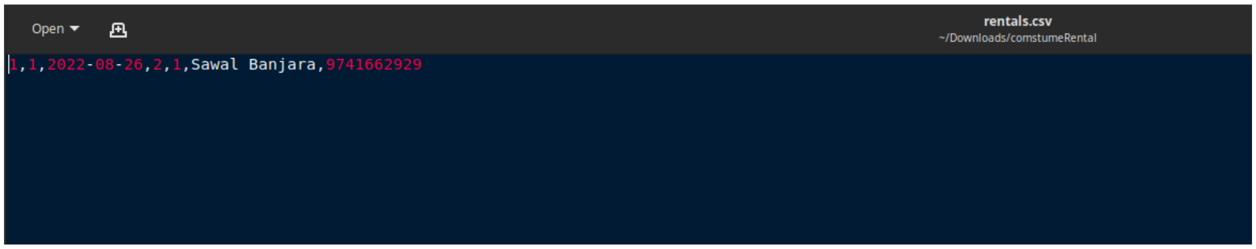
The whole process and the code for making an order for a costume with the costume rental system are provided below. The code also includes an illustration of the section for processing both new and returning rentals.

```
def handle_new_rentals():
    global db
    _t = "stocks"
    tdb = db[_t]
    _db = db.rents
    char = prompt_num(
        "Please select an ID of available item you want to rent: ", len(tdb), 1
    )
    item_id = char - 1
    entry = tdb[item_id]
    char = prompt_num(
        "Enter quantity of the item you would like to rent: ",
        entry["Qty"],
        1,
        True,  # Saving an entity, since we need at least one quantity of an item.
    )
    rental_qty = char
    entry["Qty"] -= rental_qty
    char = prompt_num(
        "Enter unit day(s) you want to rent the item for (1 unit day = 5 days): ",
        367,
        1,
    )
    rental_unit_days = char
    total_price = entry["Price"] * rental_qty * rental_unit_days
    invoice_table = [
        {
            "Name": entry["Name"],
            "Brand": entry["Brand"],
            "Price": f'$ {entry["Price"]}',
            "Rented Qty": rental_qty,
            "Rental days": rental_unit_days * 5,
            "Total Price": f"$ {total_price}",
        }
    ]
    print("-" * 43)
    print("\n" * 3)
    print("YOUR INVOICE", end="\n\n")
    print_table(invoice_table)
    char = prompt_yn("Would you now like to rent the item?")
    if char == "y":
        tdb[item_id] = entry
        db[_t] = tdb
        _id = 1
        if len(_db):
            _id = _db[-1]["ID"] + 1
        username = input("Please enter your name sir: ")
        contact = input("And just your phone number: ")
        rental_entry = {
            "ID": _id,
            "IID": entry["ID"],
            "Date of rental": today(),
            "Rental unit days": rental_unit_days,
            "Rental qty": rental_qty,
            "Username": username,
            "Contact": contact,
        }
        if len(_db):
            _db.append(rental_entry)
            commit(_db, "rents")
            db.rents = _db
        else:
            commit([rental_entry], "rents")
            db.rents = [Map(rental_entry)]
        commit(tdb, _t)
        print_table(db.stocks)
        print("\n" * 3)
        print_table(db.rents)
        print(end="\n" * 2)
    else:
        entry = {}
        tdb = []


def handle_known_rentals():
    global db
    _t = "rents"
    _db = db.stocks
    tdb = db[_t]
    if not tdb:
        return
    print("-----Rentals-----")
    print_table(tdb)
    for entry in tdb:
        _entry = _db[entry["IID"] - 1]
        return_date = parse_date_str(entry["Date of rental"]) + unit_days(
            entry["Rental unit days"]
        )
        if today() > return_date:
            print(end="\n" * 2)
            more_days = (today() - return_date).days
            print(
                f'{entry["Username"]} was supposed to return {_entry["Name"]} on
{return_date}, it has been {more_days} days since no return. You may want to call him
on {entry["Contact"]}'
            )
            char = prompt_yn("Did he respond/you called?")
            if char == "y":
                handle_late_return(entry["ID"])
                return
    char = prompt_yn("Has any of above costume been returned?")
    if char == "y":
        handle_returns()
    else:
        handle_new_rentals()
```



```
========== RESTART: /home/kodachi/Downloads/comstumeRental/main.py ==========
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                        | Brand       | Price | Qty
-- | --------------------------- | ----------- | ----- | ---
1  | Cop Costume Set             | Cartmax     | 15.0  | 20
2  | Formal Suite(Black Premium) | Megaplex    | 14.5  | 15
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25



Please select an ID of available item you want to rent: 1


Enter quantity of the item you would like to rent: 1


Enter unit day(s) you want to rent the item for (1 unit day = 5 days): 2
-------------------------------------------




YOUR INVOICE

Name            | Brand   | Price  | Rented Qty | Rental days | Total Price
--------------- | ------- | ------ | ---------- | ----------- | -----------
Cop Costume Set | Cartmax | $ 15.0 | 1          | 10          | $ 30.0


Would you now like to rent the item? [y/n]: Y
Please enter your name sir: Sawal Banjara
And just your phone number: 9741662929
ID | Name                        | Brand       | Price | Qty
-- | --------------------------- | ----------- | ----- | ---
1  | Cop Costume Set             | Cartmax     | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex    | 14.5  | 15
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25



ID | IID | Date of rental | Rental unit days | Rental qty | Username      | Contact
-- | --- | -------------- | ---------------- | ---------- | ------------- | ----------
1  | 1   | 2022-08-26     | 2                | 1          | Sawal Banjara | 9741662929
```

Figure 6: Costume Rental Terminal



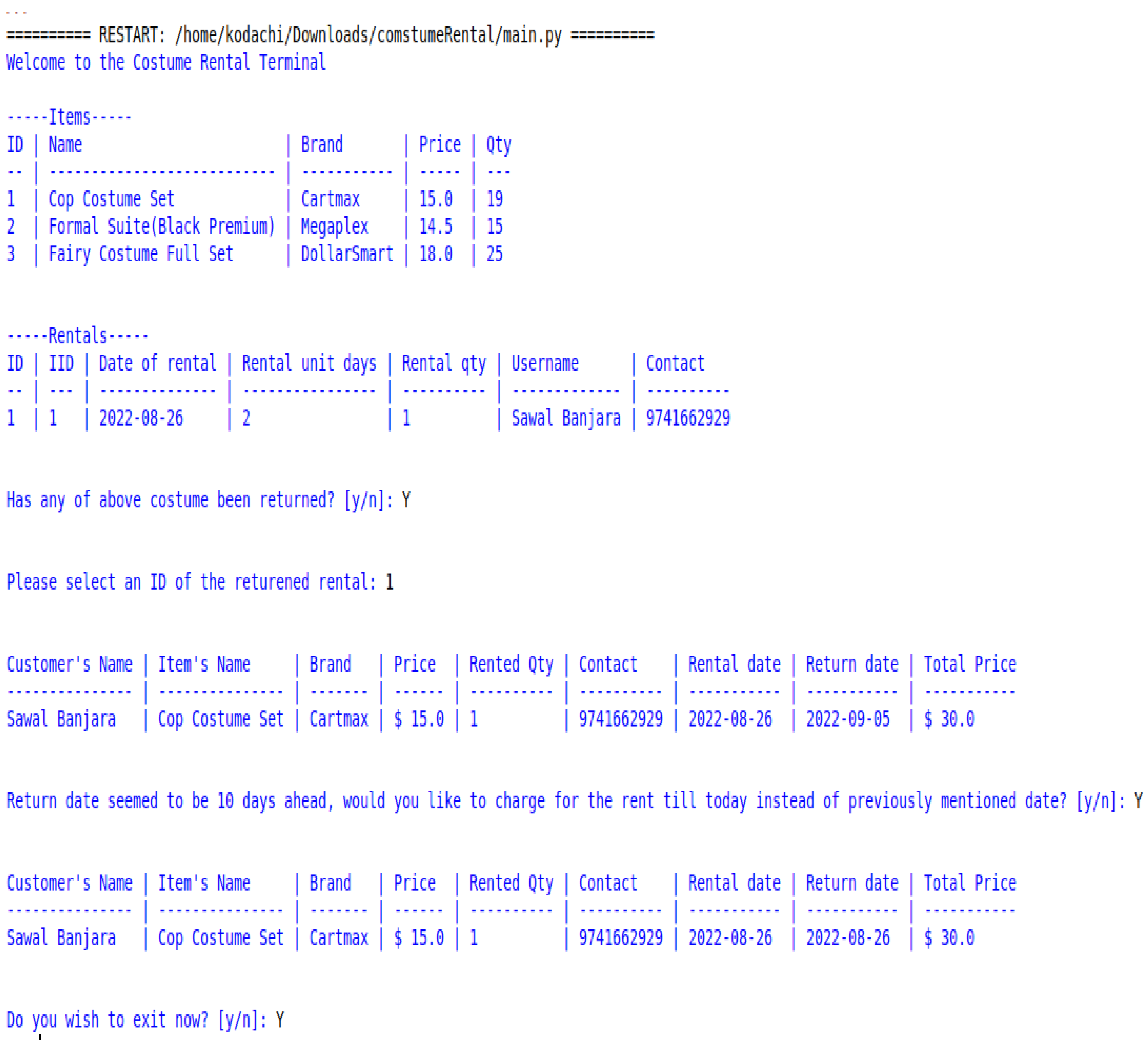
rentals.csv
~/Downloads/comstumeRental

```
1,1,2022-08-26,2,1,Sawal Banjara,9741662929
```

Figure 7: Output in the text file

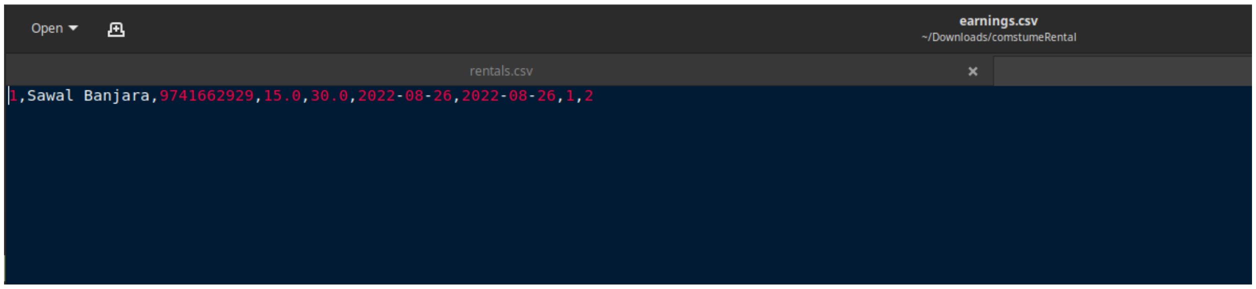
The whole method for returning a costume to the costume rental system is listed here. The code demonstrates the steps for a typical return, a late return, and even an early return in three different segments.

```
def handle_returns():
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
    char = prompt_num("Please select an ID of the retureneed rental: ", len(tdb), 1)
    item_id = char - 1
    entry = tdb[item_id]
    _entry = _db[entry["IID"] - 1]
    print(end="\n" * 2)
    total_price = _entry["Price"] * entry["Rental unit days"] * entry["Rental qty"]
    return_date = parse_date_str(entry["Date of rental"]) + unit_days(
        entry["Rental unit days"]
    )
    invoice_table = [
        {
            "Customer's Name": entry["Username"],
            "Item's Name": _entry["Name"],
            "Brand": _entry["Brand"],
            "Price": f'$ {_entry["Price"]}',
            "Rented Qty": entry["Rental qty"],
            "Contact": entry["Contact"],
            "Rental date": entry["Date of rental"],
            "Return date": return_date,
            "Total Price": f"$ {total_price}",
        }
    ]
    print_table(invoice_table)
    if return_date == today():
        _id = 1
        if len(edb):
            _id = edb[-1]["ID"]
        earning_entry = {
            "ID": _id,
            "username": entry["Username"],
            "contact": entry["Contact"],
            "price_per_unit": _entry["Price"],
            "total_price": total_price,
            "date_of_rental": entry["Date of rental"],
            "date_of_return": return_date,
            "rental_qty": entry["Rental qty"],
            "rental_unit_days": entry["Rental unit days"],
        }
        if len(edb):
            edb.append(earning_entry)
            db.earnings = edb
            commit(edb, "earnings")
        else:
            commit([earning_entry], "earnings")
            db.earnings = [Map(earning_entry)]
        for _index, _entry in enumerate(_db):
            if entry["IID"] == _entry["ID"]:
                _entry["Qty"] += entry["Rental qty"]
                _db[_index] = _entry
                commit(_db, "stocks")
                db.stocks = _db
        for index, _entry in enumerate(tdb):
            if _entry["ID"] == entry["ID"]:
                del tdb[index]
                commit(tdb, "rents")
                db.rents = tdb
    elif (return_date - today()).days > 0:
        handle_early_return(entry["ID"])
    else:
        handle_late_return(entry["ID"])


def handle_late_return(_id):
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
    entry = tdb[_id - 1]
    _entry = _db[entry["IID"] - 1]
    item_qty = entry["Rental qty"]
    total_price = _entry["Price"] * entry["Rental unit days"] * item_qty
    return_date = parse_date_str(entry["Date of rental"]) + unit_days(
        entry["Rental unit days"]
    )
    days_late = (today() - return_date).days
    return_date += timedelta(days=days_late)
    choice = prompt_yn(
        f"Looks like you've submitted the item {days_late} days late, would you like to
charge costumer fine for it?"
    )
    print(end="\n" * 2)
    if choice == "y":
        price_per_day_per_item = _entry["Price"] / 5
        fine_price = price_per_day_per_item * item_qty * days_late
        total_price += fine_price
    invoice_table = [
        {
            "Customer's Name": entry["Username"],
            "Item's Name": _entry["Name"],
            "Brand": _entry["Brand"],
            "Price": f'$ {_entry["Price"]}',
            "Rented Qty": entry["Rental qty"],
            "Contact": entry["Contact"],
            "Rental date": entry["Date of rental"],
            "Return date": return_date,
            "Total Price": f"$ {total_price}",
        }
    ]
    print_table(invoice_table)
    _id = 1
    if len(edb):
        _id = edb[-1]["ID"]
    earning_entry = {
        "ID": _id,
        "username": entry["Username"],
        "contact": entry["Contact"],
        "price_per_unit": _entry["Price"],
        "total_price": total_price,
        "date_of_rental": entry["Date of rental"],
        "date_of_return": return_date,
        "rental_qty": entry["Rental qty"],
        "rental_unit_days": entry["Rental unit days"],
    }
    if len(edb):
        edb.append(earning_entry)
        db.earnings = edb
        commit(edb, "earnings")
    else:
        commit([earning_entry], "earnings")
        db.earnings = [Map(earning_entry)]
    for _index, _entry in enumerate(_db):
        if entry["IID"] == _entry["ID"]:
            _entry["Qty"] += entry["Rental qty"]
            _db[_index] = _entry
            commit(_db, "stocks")
            db.stocks = _db
    for index, _entry in enumerate(tdb):
        if _entry["ID"] == entry["ID"]:
            del tdb[index]
            commit(tdb, "rents")
            db.rents = tdb


def handle_early_return(_id):
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
    entry = tdb[_id - 1]
    _entry = _db[entry["IID"] - 1]
    item_qty = entry["Rental qty"]
    total_price = _entry["Price"] * entry["Rental unit days"] * item_qty
    return_date = parse_date_str(entry["Date of rental"]) + unit_days(
        entry["Rental unit days"]
    )
    days_early = (return_date - today()).days
    return_date -= timedelta(days=days_early)
    choice = prompt_yn(
        f"Return date seemed to be {days_early} days ahead, would you like to charge for
the rent till today instead of previously mentioned date?"
    )
    print(end="\n" * 2)
    if choice == "n":
        price_per_day_per_item = _entry["Price"] / 5
        deduction_price = price_per_day_per_item * item_qty * days_early
        total_price -= deduction_price
    invoice_table = [
        {
            "Customer's Name": entry["Username"],
            "Item's Name": _entry["Name"],
            "Brand": _entry["Brand"],
            "Price": f'$ {_entry["Price"]}',
            "Rented Qty": entry["Rental qty"],
            "Contact": entry["Contact"],
            "Rental date": entry["Date of rental"],
            "Return date": return_date,
            "Total Price": f"$ {total_price}",
        }
    ]
    print_table(invoice_table)
    _id = 1
    if len(edb):
        _id = edb[-1]["ID"]
    earning_entry = {
        "ID": _id,
        "username": entry["Username"],
        "contact": entry["Contact"],
        "price_per_unit": _entry["Price"],
        "total_price": total_price,
        "date_of_rental": entry["Date of rental"],
        "date_of_return": return_date,
        "rental_qty": entry["Rental qty"],
        "rental_unit_days": entry["Rental unit days"],
    }
    if len(edb):
        edb.append(earning_entry)
        db.earnings = edb
        commit(edb, "earnings")
    else:
        commit([earning_entry], "earnings")
        db.earnings = [Map(earning_entry)]
    for _index, _entry in enumerate(_db):
        if entry["IID"] == _entry["ID"]:
            _entry["Qty"] += entry["Rental qty"]
            _db[_index] = _entry
            commit(_db, "stocks")
            db.stocks = _db
    for index, _entry in enumerate(tdb):
        if _entry["ID"] == entry["ID"]:
            del tdb[index]
            commit(tdb, "rents")
            db.rents = tdb
```



```
...
========== RESTART: /home/kodachi/Downloads/comstumeRental/main.py ==========
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                       | Brand      | Price | Qty
-- | -------------------------- | ---------- | ----- | ---
1  | Cop Costume Set            | Cartmax    | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex   | 14.5  | 15
3  | Fairy Costume Full Set     | DollarSmart | 18.0  | 25


-----Rentals-----
ID | IID | Date of rental | Rental unit days | Rental qty | Username     | Contact
-- | --- | -------------- | ---------------- | ---------- | ------------ | ----------
1  | 1   | 2022-08-26     | 2                | 1          | Sawal Banjara | 9741662929


Has any of above costume been returned? [y/n]: Y


Please select an ID of the returened rental: 1


Customer's Name | Item's Name     | Brand   | Price  | Rented Qty | Contact    | Rental date | Return date | Total Price
--------------- | --------------- | ------- | ------ | ---------- | ---------- | ----------- | ----------- | -----------
Sawal Banjara   | Cop Costume Set | Cartmax | $ 15.0 | 1          | 9741662929 | 2022-08-26  | 2022-09-05  | $ 30.0


Return date seemed to be 10 days ahead, would you like to charge for the rent till today instead of previously mentioned date? [y/n]: Y


Customer's Name | Item's Name     | Brand   | Price  | Rented Qty | Contact    | Rental date | Return date | Total Price
--------------- | --------------- | ------- | ------ | ---------- | ---------- | ----------- | ----------- | -----------
Sawal Banjara   | Cop Costume Set | Cartmax | $ 15.0 | 1          | 9741662929 | 2022-08-26  | 2022-08-26  | $ 30.0


Do you wish to exit now? [y/n]: Y
```

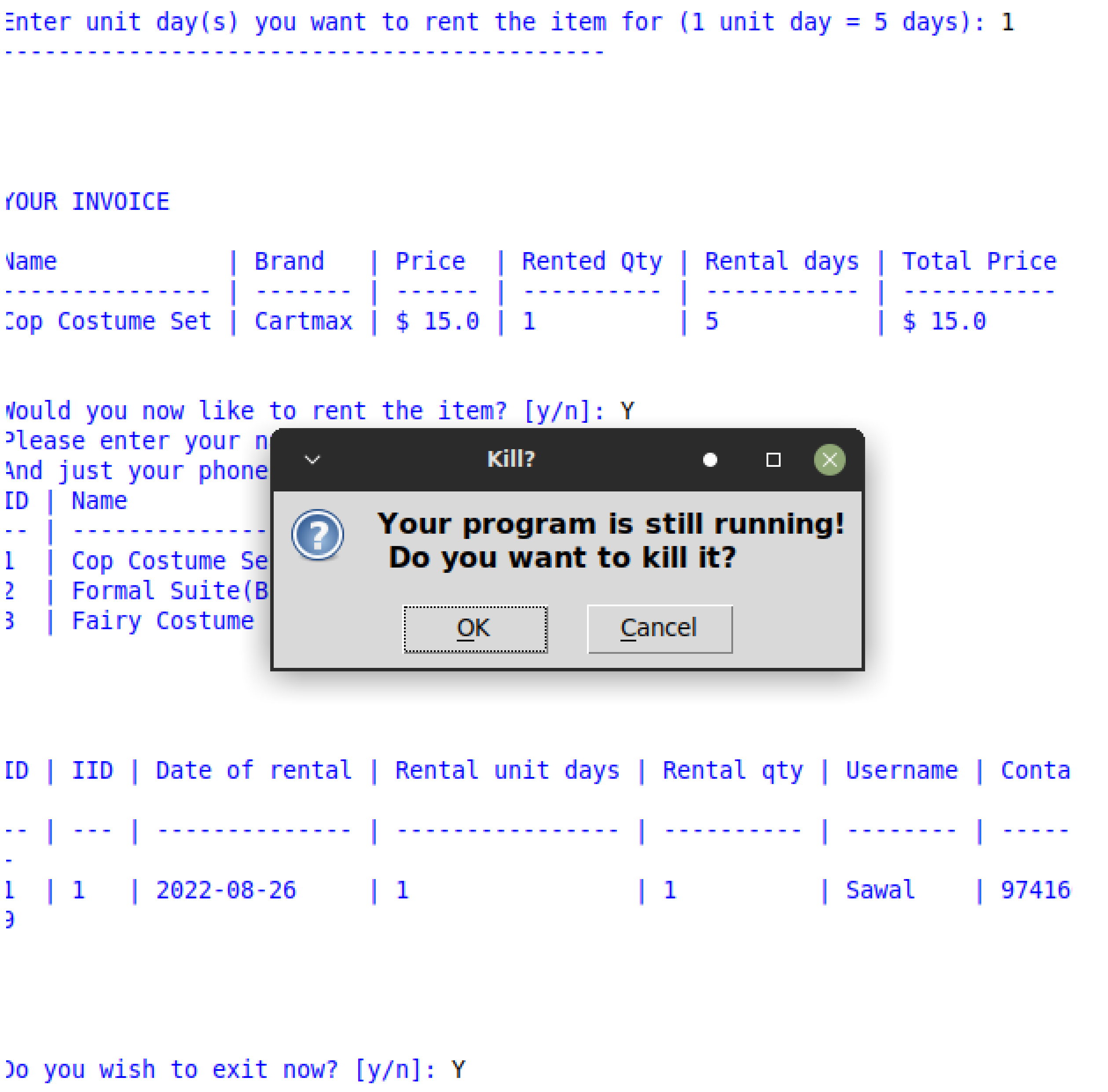
Figure 8: Costume returned



```
1,Sawal Banjara,9741662929,15.0,30.0,2022-08-26,2022-08-26,1,2
```

Figure 9: Costume returned in the text file.

The termination of the program:



```
Enter unit day(s) you want to rent the item for (1 unit day = 5 days): 1
-------------------------------------------



YOUR INVOICE

Name             | Brand   | Price  | Rented Qty | Rental days | Total Price
---------------- | ------- | ------ | ---------- | ----------- | -----------
Cop Costume Set  | Cartmax | $ 15.0 | 1          | 5           | $ 15.0

Would you now like to rent the item? [y/n]: Y
Please enter your n
And just your phone
ID | Name
-- | --------------
1  | Cop Costume Se
2  | Formal Suite(B
3  | Fairy Costume


ID | IID | Date of rental | Rental unit days | Rental qty | Username | Conta

-- | --- | -------------- | ---------------- | ---------- | -------- | -----
-
1  | 1   | 2022-08-26     | 1                | 1          | Sawal    | 97416
9



Do you wish to exit now? [y/n]: Y
```

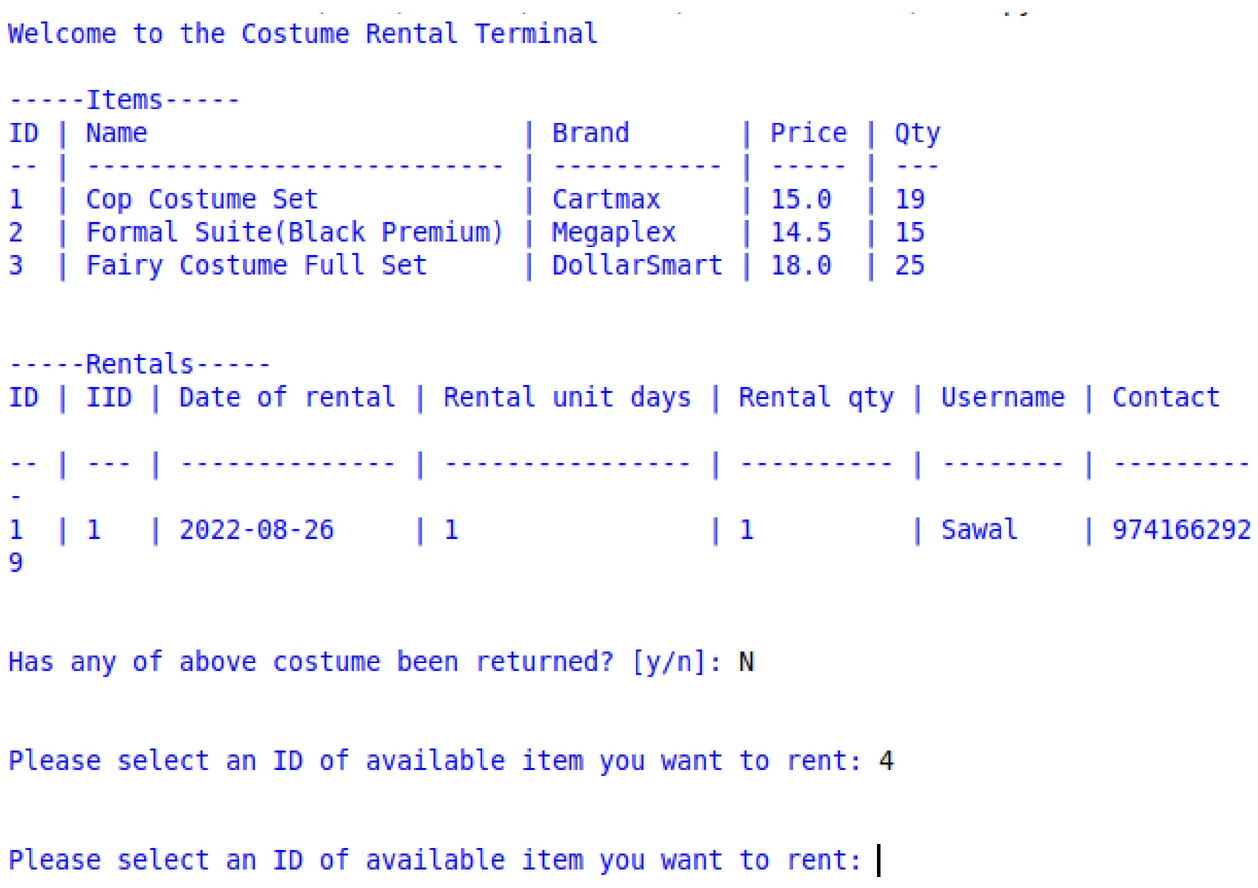
Figure 10: Terminal of program.

Testing

Testing 1:

Table 1: To demonstrate how try and expect is being used

Objective	To demonstrate how try and expect is being used
Action	The else clause, which is optional in the try... except statement, must come after all except clauses when it is present. It's helpful for code that has to run even if the try clause doesn't throw an error.
Expected Results	Once more, the code solicits user input. Even if the inputted number exceeds the command, the code continues to function normally and prompts the user for input.
Actual Results	As in the instruction, the programme once more requests human input.
Conclusion	The experiment was a success.



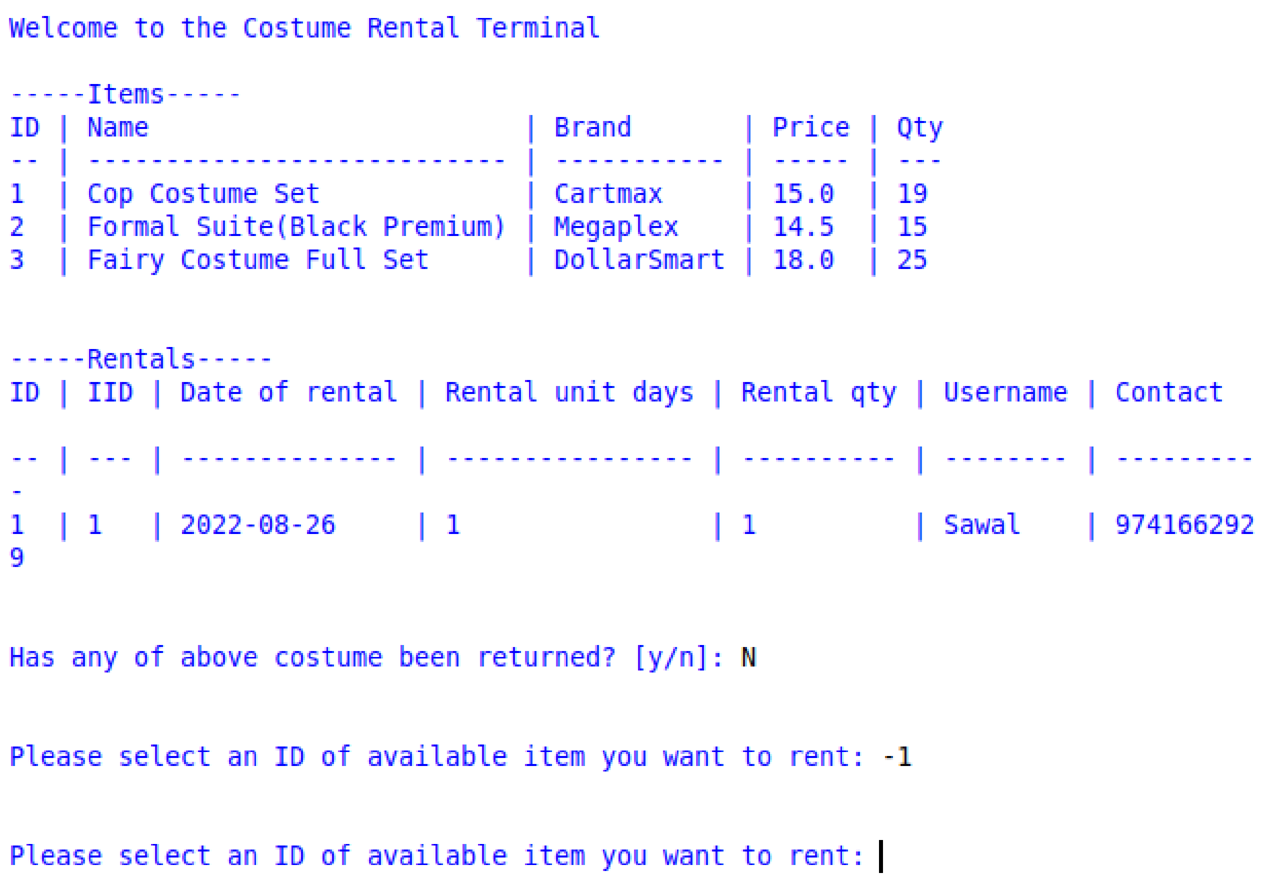
```
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                        | Brand       | Price | Qty
-- | --------------------------- | ----------- | ----- | ---
1  | Cop Costume Set             | Cartmax     | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex    | 14.5  | 15
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25


-----Rentals-----
ID | IID | Date of rental | Rental unit days | Rental qty | Username | Contact

-- | --- | -------------- | ---------------- | ---------- | -------- | ---------
-
1  | 1   | 2022-08-26     | 1                | 1          | Sawal    | 974166292
9


Has any of above costume been returned? [y/n]: N


Please select an ID of available item you want to rent: 4


Please select an ID of available item you want to rent: |
```

Figure 11:Try and except

Testing 2:

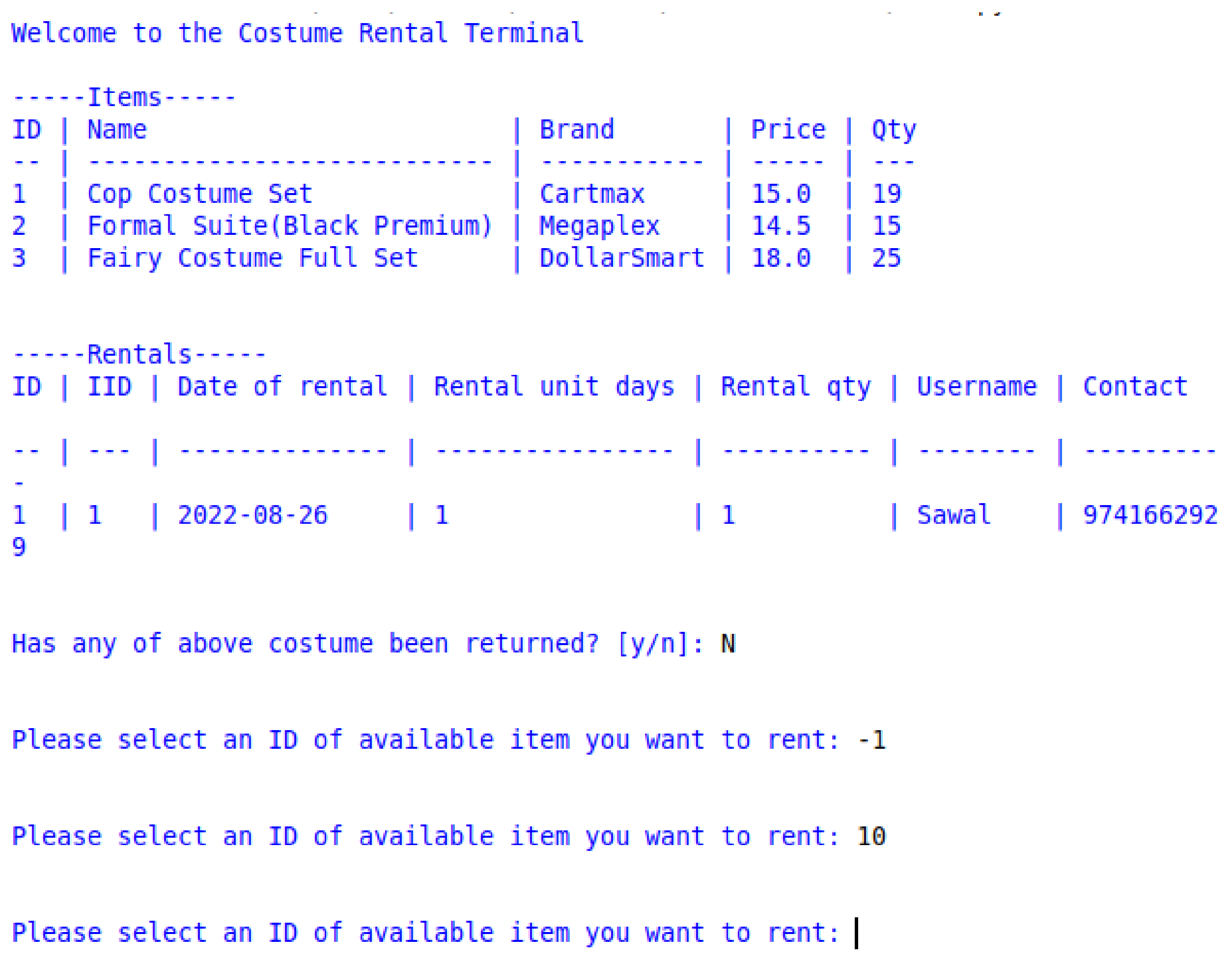
Table 2: Costume selection, rental, and return with incorrect numbers.

Objective	Costume selection, rental, and return with incorrect numbers.
Action	The demonstration illustrates how the code reacts to invalid input during renting.
Expected Results	Even when a number is entered that is outside of the selection range, the code still functions correctly and prompts the user for input.
Actual Results	The code requests user input once more, just like in the instruction.
Conclusion	The experiment was a success.



```
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                        | Brand       | Price | Qty
-- | --------------------------- | ----------- | ----- | ---
1  | Cop Costume Set             | Cartmax     | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex    | 14.5  | 15
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25


-----Rentals-----
ID | IID | Date of rental | Rental unit days | Rental qty | Username | Contact

-- | --- | -------------- | ---------------- | ---------- | -------- | ---------
-
1  | 1   | 2022-08-26     | 1                | 1          | Sawal    | 974166292
9


Has any of above costume been returned? [y/n]: N


Please select an ID of available item you want to rent: -1


Please select an ID of available item you want to rent: 
```

Figure 12: Negative value in terminal



```
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                        | Brand       | Price | Qty
-- | --------------------------- | ----------- | ----- | ---
1  | Cop Costume Set             | Cartmax     | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex    | 14.5  | 15
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25


-----Rentals-----
ID | IID | Date of rental | Rental unit days | Rental qty | Username | Contact

-- | --- | -------------- | ---------------- | ---------- | -------- | ---------
-
1  | 1   | 2022-08-26     | 1                | 1          | Sawal    | 974166292
9


Has any of above costume been returned? [y/n]: N


Please select an ID of available item you want to rent: -1


Please select an ID of available item you want to rent: 10


Please select an ID of available item you want to rent: |
```

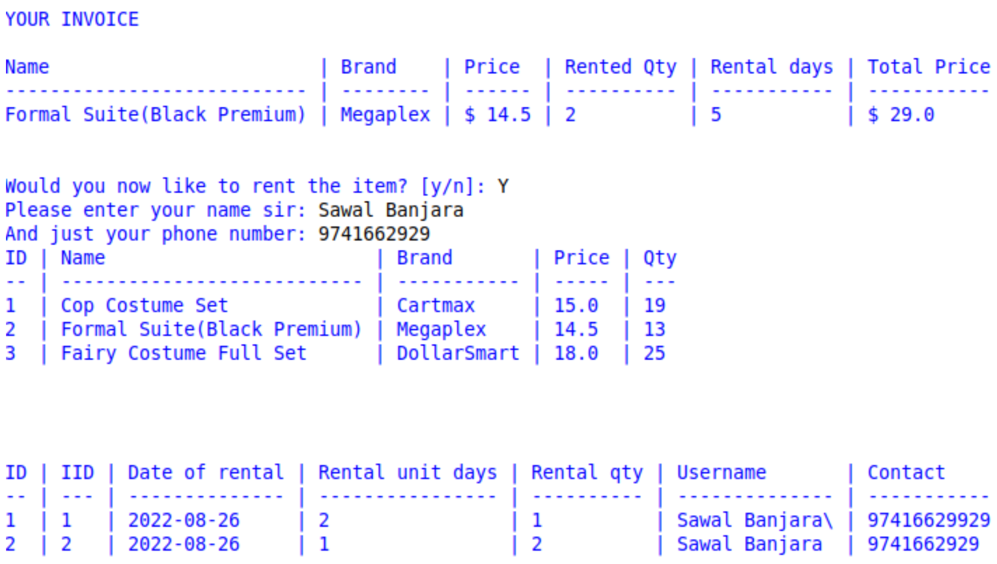
Figure 13: Input value exceeding the instruction

Testing 3:

Table 3: File generation of renting costume (Renting multiple customers)

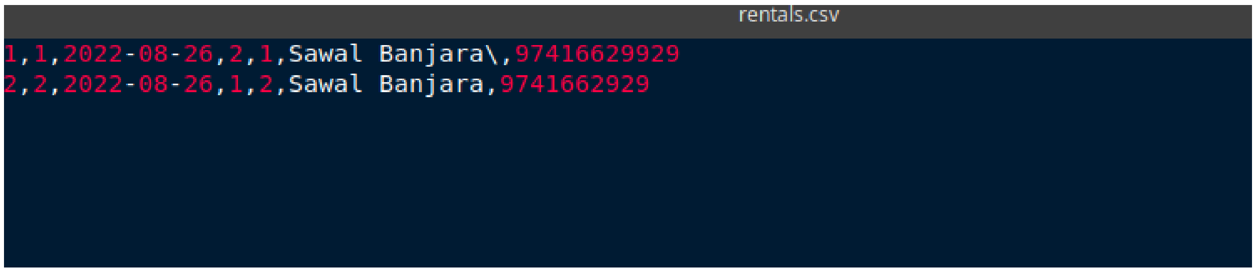
Objective	File generation of renting costume (Renting multiple customers)
Action	Creating text files and displaying the results directly in the shell is done while renting several costumes.
Expected Results	the generation of a text file including a note about renting, information about multiple rentals, and shell output.
Actual Results	The output is shown in the shell, the text

	file is formed, and several costumes are hired.
Conclusion	The experiment was a success.



```
YOUR INVOICE

Name                       | Brand    | Price  | Rented Qty | Rental days | Total Price
-------------------------- | -------- | ------ | ---------- | ----------- | -----------
Formal Suite(Black Premium) | Megaplex | $ 14.5 | 2          | 5           | $ 29.0


Would you now like to rent the item? [y/n]: Y
Please enter your name sir: Sawal Banjara
And just your phone number: 9741662929
ID | Name                        | Brand      | Price | Qty
-- | --------------------------- | ---------- | ----- | ---
1  | Cop Costume Set             | Cartmax    | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex   | 14.5  | 13
3  | Fairy Costume Full Set      | DollarSmart | 18.0  | 25



ID | IID | Date of rental | Rental unit days | Rental qty | Username       | Contact
-- | --- | -------------- | ---------------- | ---------- | -------------- | -----------
1  | 1   | 2022-08-26     | 2                | 1          | Sawal Banjara\ | 97416629929
2  | 2   | 2022-08-26     | 1                | 2          | Sawal Banjara  | 9741662929
```

Figure 14: Costume rented



rentals.csv

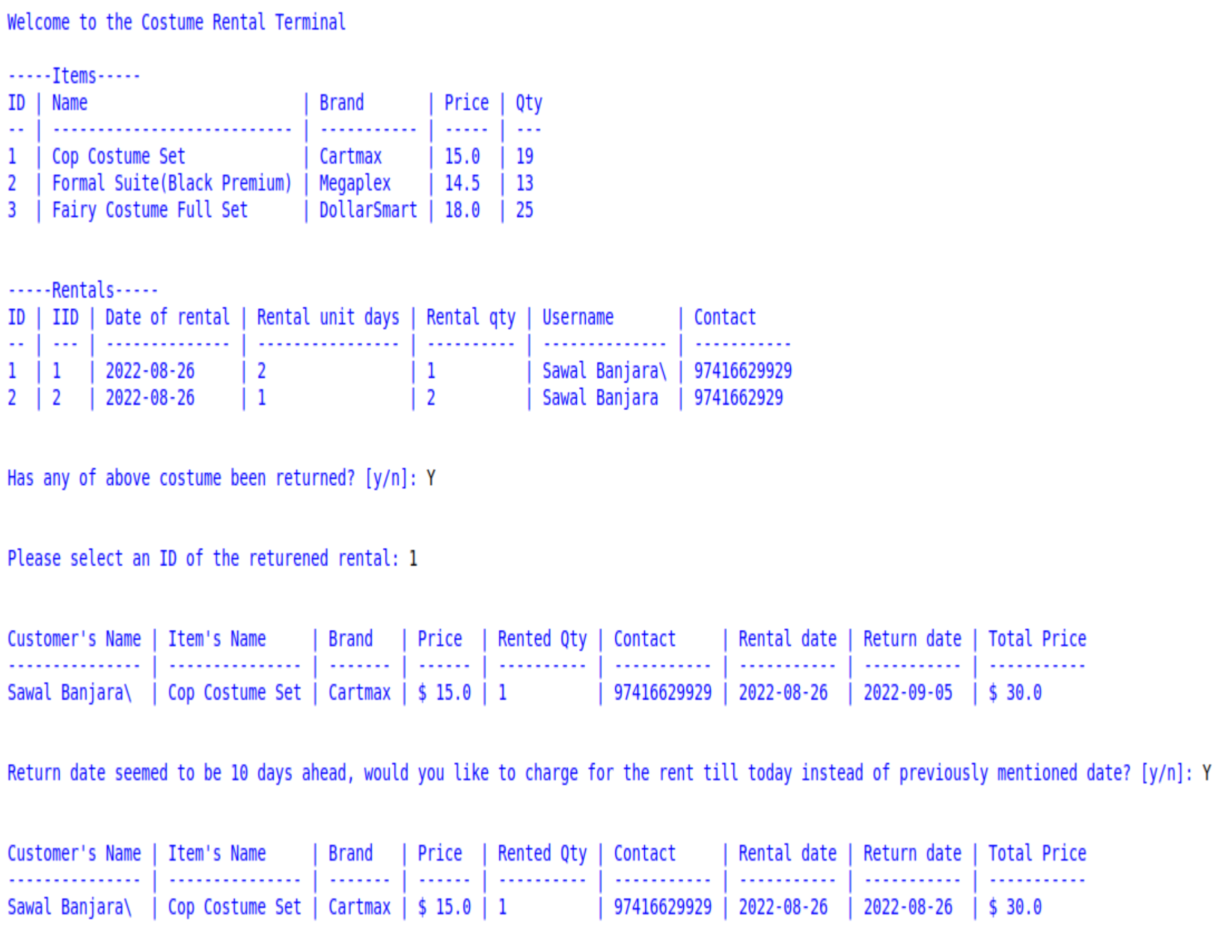
```
1,1,2022-08-26,2,1,Sawal Banjara\,97416629929
2,2,2022-08-26,1,2,Sawal Banjara,9741662929
```

Figure 15: Multiple costume rented in text file

Testing 4:

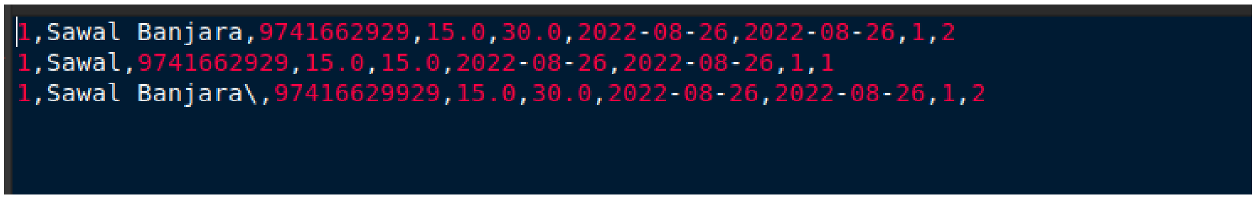
Table 4: File generation of returning process of costume

Objective	File generation of returning process of costume
Action	The creation of text files demonstrates both the shell-return and shell-return of numerous outfits.
Expected Results	The text files are created, and the output is shown in a shell.
Actual Results	The output was produced to a text file, which the shell then showed.
Conclusion	The experiment was a success.



```
Welcome to the Costume Rental Terminal

-----Items-----
ID | Name                      | Brand      | Price | Qty
-- | ------------------------- | ---------- | ----- | ---
1  | Cop Costume Set           | Cartmax    | 15.0  | 19
2  | Formal Suite(Black Premium) | Megaplex | 14.5  | 13
3  | Fairy Costume Full Set    | DollarSmart | 18.0  | 25


-----Rentals-----
ID | IID | Date of rental | Rental unit days | Rental qty | Username      | Contact
-- | --- | -------------- | ---------------- | ---------- | ------------- | -----------
1  | 1   | 2022-08-26     | 2                | 1          | Sawal Banjara\ | 97416629929
2  | 2   | 2022-08-26     | 1                | 2          | Sawal Banjara  | 9741662929


Has any of above costume been returned? [y/n]: Y


Please select an ID of the returened rental: 1


Customer's Name | Item's Name     | Brand   | Price  | Rented Qty | Contact     | Rental date | Return date | Total Price
--------------- | --------------- | ------- | ------ | ---------- | ----------- | ----------- | ----------- | -----------
Sawal Banjara\  | Cop Costume Set | Cartmax | $ 15.0 | 1          | 97416629929 | 2022-08-26  | 2022-09-05  | $ 30.0


Return date seemed to be 10 days ahead, would you like to charge for the rent till today instead of previously mentioned date? [y/n]: Y


Customer's Name | Item's Name     | Brand   | Price  | Rented Qty | Contact     | Rental date | Return date | Total Price
--------------- | --------------- | ------- | ------ | ---------- | ----------- | ----------- | ----------- | -----------
Sawal Banjara\  | Cop Costume Set | Cartmax | $ 15.0 | 1          | 97416629929 | 2022-08-26  | 2022-08-26  | $ 30.0
```

Figure 16: Return in Terminal



```
1,Sawal Banjara,9741662929,15.0,30.0,2022-08-26,2022-08-26,1,2
1,Sawal,9741662929,15.0,15.0,2022-08-26,2022-08-26,1,1
1,Sawal Banjara\,97416629929,15.0,30.0,2022-08-26,2022-08-26,1,2
```

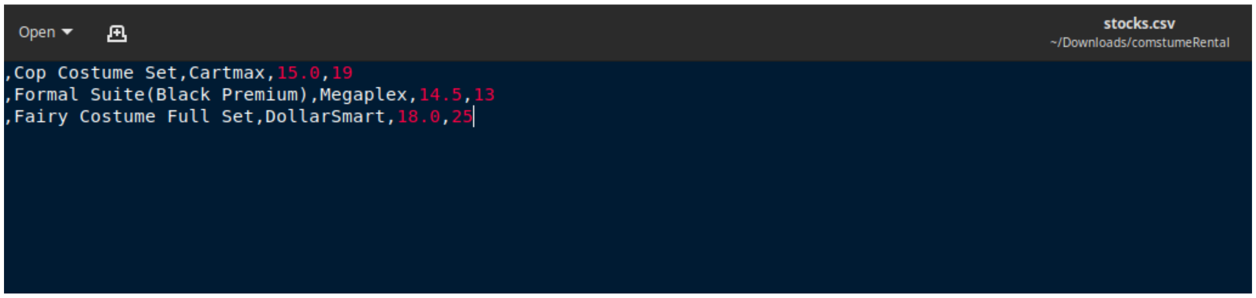
Figure 17: Return in Text File

Testing 5:

Table 5: Show the update in stock of costume

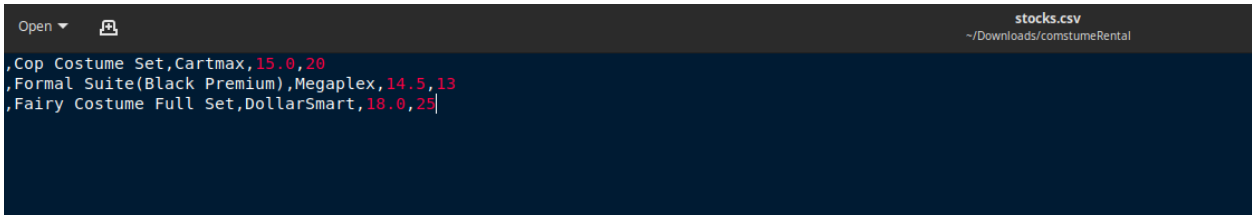
Objective	Show the update in stock of costume
Action	Stock.txt has been modified.

Expected Results	When a product is returned, more inventory is added, and when you're renting, less inventory is produced.
Actual Results	The stock went up and down, correspondingly.
Conclusion	The experiment was a success.



```
,Cop Costume Set,Cartmax,15.0,19
,Formal Suite(Black Premium),Megaplex,14.5,13
,Fairy Costume Full Set,DollarSmart,18.0,25
```

Figure 18: Stock changed by Renting



```
,Cop Costume Set,Cartmax,15.0,20
,Formal Suite(Black Premium),Megaplex,14.5,13
,Fairy Costume Full Set,DollarSmart,18.0,25
```

Figure 19: Stock changed by returning

Conclusion

This project gave us practical experience with the costume rental system while also assisting in our efforts to improve the Python programming language. With the aid of this project, I was able to comprehend many data structures used in Python programming, including list, tuple, dictionary, and others. Overall, because of the project's intricacy, I was able to learn the fundamentals of programming while also sharpening my problem-solving abilities. The renting system for costumes is entirely discussed in the coursework and may be used in daily life. Since the rental and return of the user information is generated in a text file together with the revenue and the quantity of the stock also increases and decreases in accordance with those earnings, the business operated project helped to understand how the business operates.

Appendix:

Main.py

```
#!/usr/bin/env python3

from utils import (Map, commit, exit_prompt, parse_date_str, print_table,
                   prompt_num, prompt_yn, read, timedelta, today, unit_days)

db = Map({"stocks": read(), "rents": read("rents"), "earnings": read("earnings")})


def handle_returns():
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
```

```
char = prompt_num("Please select an ID of the returened rental: ", len(tdb), 1)
item_id = char - 1
entry = tdb[item_id]
_entry = _db[entry["IID"] - 1]
print(end="\n" * 2)
total_price = _entry["Price"] * entry["Rental unit days"] * entry["Rental qty"]
return_date = parse_date_str(entry["Date of rental"]) + unit_days(
    entry["Rental unit days"]
)
invoice_table = [
    {
        "Customer's Name": entry["Username"],
        "Item's Name": _entry["Name"],
        "Brand": _entry["Brand"],
        "Price": f'$ {_entry["Price"]}',
        "Rented Qty": entry["Rental qty"],
        "Contact": entry["Contact"],
        "Rental date": entry["Date of rental"],
        "Return date": return_date,
        "Total Price": f"$ {total_price}",
    }
]
print_table(invoice_table)
if return_date == today():
    _id = 1
    if len(edb):
        _id = edb[-1]["ID"]
    earning_entry = {
        "ID": _id,
        "username": entry["Username"],
        "contact": entry["Contact"],
        "price_per_unit": _entry["Price"],
        "total_price": total_price,
        "date_of_rental": entry["Date of rental"],
        "date_of_return": return_date,
        "rental_qty": entry["Rental qty"],
        "rental_unit_days": entry["Rental unit days"],
    }
    if len(edb):
        edb.append(earning_entry)
        db.earnings = edb
        commit(edb, "earnings")
    else:
        commit([earning_entry], "earnings")
        db.earnings = [Map(earning_entry)]
    for _index, _entry in enumerate(_db):
            if entry["IID"] == _entry["ID"]:
                _entry["Qty"] += entry["Rental qty"]
                _db[_index] = _entry
                commit(_db, "stocks")
                db.stocks = _db
        for index, _entry in enumerate(tdb):
            if _entry["ID"] == entry["ID"]:
                del tdb[index]
                commit(tdb, "rents")
                db.rents = tdb
    elif (return_date - today()).days > 0:
        handle_early_return(entry["ID"])
    else:
        handle_late_return(entry["ID"])


def handle_late_return(_id):
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
    entry = tdb[_id - 1]
    _entry = _db[entry["IID"] - 1]
    item_qty = entry["Rental qty"]
    total_price = _entry["Price"] * entry["Rental unit days"] * item_qty
    return_date = parse_date_str(entry["Date of rental"]) + unit_days(
        entry["Rental unit days"]
    )
    days_late = (today() - return_date).days
    return_date += timedelta(days=days_late)
    choice = prompt_yn(
        f"Looks like you've submitted the item {days_late} days late, would you like to
charge costumer fine for it?"
    )
    print(end="\n" * 2)
    if choice == "y":
        price_per_day_per_item = _entry["Price"] / 5
        fine_price = price_per_day_per_item * item_qty * days_late
        total_price += fine_price
    invoice_table = [
        {
            "Customer's Name": entry["Username"],
            "Item's Name": _entry["Name"],
            "Brand": _entry["Brand"],
            "Price": f'$ {_entry["Price"]}',
            "Rented Qty": entry["Rental qty"],
            "Contact": entry["Contact"],
            "Rental date": entry["Date of rental"],
            "Return date": return_date,
            "Total Price": f"$ {total_price}",
        }
    ]
    print_table(invoice_table)
    _id = 1
    if len(edb):
        _id = edb[-1]["ID"]
    earning_entry = {
        "ID": _id,
        "username": entry["Username"],
        "contact": entry["Contact"],
        "price_per_unit": _entry["Price"],
        "total_price": total_price,
        "date_of_rental": entry["Date of rental"],
        "date_of_return": return_date,
        "rental_qty": entry["Rental qty"],
        "rental_unit_days": entry["Rental unit days"],
    }
    if len(edb):
        edb.append(earning_entry)
        db.earnings = edb
        commit(edb, "earnings")
    else:
        commit([earning_entry], "earnings")
        db.earnings = [Map(earning_entry)]
    for _index, _entry in enumerate(_db):
        if entry["IID"] == _entry["ID"]:
            _entry["Qty"] += entry["Rental qty"]
            _db[_index] = _entry
            commit(_db, "stocks")
            db.stocks = _db
    for index, _entry in enumerate(tdb):
        if _entry["ID"] == entry["ID"]:
            del tdb[index]
            commit(tdb, "rents")
            db.rents = tdb


def handle_early_return(_id):
    global db
    _t = "rents"
    tdb = db[_t]
    edb = db.earnings
    _db = db.stocks
    entry = tdb[_id - 1]
    _entry = _db[entry["IID"] - 1]
    item_qty = entry["Rental qty"]
    total_price = _entry["Price"] * entry["Rental unit days"] * item_qty
    return_date = parse_date_str(entry["Date of rental"]) + unit_days(
        entry["Rental unit days"]
    )
    days_early = (return_date - today()).days
    return_date -= timedelta(days=days_early)
    choice = prompt_yn(
        f"Return date seemed to be {days_early} days ahead, would you like to charge for
the rent till today instead of previously mentioned date?"
    )
    print(end="\n" * 2)
    if choice == "n":
        price_per_day_per_item = _entry["Price"] / 5
        deduction_price = price_per_day_per_item * item_qty * days_early
        total_price -= deduction_price
    invoice_table = [
        {
            "Customer's Name": entry["Username"],
            "Item's Name": _entry["Name"],
            "Brand": _entry["Brand"],
            "Price": f'$ {_entry["Price"]}',
            "Rented Qty": entry["Rental qty"],
            "Contact": entry["Contact"],
            "Rental date": entry["Date of rental"],
            "Return date": return_date,
            "Total Price": f"$ {total_price}",
        }
    ]
    print_table(invoice_table)
    _id = 1
    if len(edb):
        _id = edb[-1]["ID"]
    earning_entry = {
        "ID": _id,
        "username": entry["Username"],
        "contact": entry["Contact"],
        "price_per_unit": _entry["Price"],
        "total_price": total_price,
        "date_of_rental": entry["Date of rental"],
        "date_of_return": return_date,
        "rental_qty": entry["Rental qty"],
        "rental_unit_days": entry["Rental unit days"],
    }
    if len(edb):
        edb.append(earning_entry)
        db.earnings = edb
        commit(edb, "earnings")
    else:
        commit([earning_entry], "earnings")
        db.earnings = [Map(earning_entry)]
    for _index, _entry in enumerate(_db):
        if entry["IID"] == _entry["ID"]:
            _entry["Qty"] += entry["Rental qty"]
            _db[_index] = _entry
            commit(_db, "stocks")
            db.stocks = _db
    for index, _entry in enumerate(tdb):
        if _entry["ID"] == entry["ID"]:
            del tdb[index]
            commit(tdb, "rents")
            db.rents = tdb


def handle_new_rentals():
    global db
    _t = "stocks"
    tdb = db[_t]
    _db = db.rents
    char = prompt_num(
        "Please select an ID of available item you want to rent: ", len(tdb), 1
    )
    item_id = char - 1
    entry = tdb[item_id]
    char = prompt_num(
        "Enter quantity of the item you would like to rent: ",
        entry["Qty"],
        1,
        True,  # Saving an entity, since we need at least one quantity of an item.
    )
    rental_qty = char
    entry["Qty"] -= rental_qty
    char = prompt_num(
        "Enter unit day(s) you want to rent the item for (1 unit day = 5 days): ",
        367,
        1,
    )
    rental_unit_days = char
```

```
total_price = entry["Price"] * rental_qty * rental_unit_days
invoice_table = [
    {
        "Name": entry["Name"],
        "Brand": entry["Brand"],
        "Price": f'$ {entry["Price"]}',
        "Rented Qty": rental_qty,
        "Rental days": rental_unit_days * 5,
        "Total Price": f"$ {total_price}",
    }
]
print("-" * 43)
print("\n" * 3)
print("YOUR INVOICE", end="\n\n")
print_table(invoice_table)
char = prompt_yn("Would you now like to rent the item?")
if char == "y":
    tdb[item_id] = entry
    db[_t] = tdb
    _id = 1
    if len(_db):
        _id = _db[-1]["ID"] + 1
    username = input("Please enter your name sir: ")
    contact = input("And just your phone number: ")
    rental_entry = {
        "ID": _id,
        "IID": entry["ID"],
        "Date of rental": today(),
        "Rental unit days": rental_unit_days,
        "Rental qty": rental_qty,
        "Username": username,
        "Contact": contact,
    }
    if len(_db):
        _db.append(rental_entry)
        commit(_db, "rents")
        db.rents = _db
    else:
        commit([rental_entry], "rents")
        db.rents = [Map(rental_entry)]
    commit(tdb, _t)
    print_table(db.stocks)
    print("\n" * 3)
    print_table(db.rents)
    print(end="\n" * 2)
else:
        entry = {}
        tdb = []


def handle_known_rentals():
    global db
    _t = "rents"
    _db = db.stocks
    tdb = db[_t]
    if not tdb:
        return
    print("-----Rentals-----")
    print_table(tdb)
    for entry in tdb:
        _entry = _db[entry["IID"] - 1]
        return_date = parse_date_str(entry["Date of rental"]) + unit_days(
            entry["Rental unit days"]
        )
        if today() > return_date:
            print(end="\n" * 2)
            more_days = (today() - return_date).days
            print(
                f'{entry["Username"]} was supposed to return {_entry["Name"]} on
{return_date}, it has been {more_days} days since no return. You may want to call him
on {entry["Contact"]}'
            )
            char = prompt_yn("Did he respond/you called?")
            if char == "y":
                handle_late_return(entry["ID"])
                return
    char = prompt_yn("Has any of above costume been returned?")
    if char == "y":
        handle_returns()
    else:
        handle_new_rentals()


def main():
    global db
    db = Map({"stocks": read(), "rents": read("rents"), "earnings": read("earnings")})
    print("Welcome to the Costume Rental Terminal", end="\n" * 2)
    print("-----Items-----")
    print_table(db.stocks)
    try:
        print(end="\n" * 2)
        if read("rent"):
            handle_known_rentals()
        else:
            handle_new_rentals()
        exit_prompt("Do you wish to exit now?")
        main()
    except Exception as e:
        print()
        print(f"Exception -> {e}")
        print(end="\n" * 2)
        print("Press enter to continue..", end="", flush=True)
        input()
    exit_prompt("Do you wish to exit now?")
    main()


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

Utils.py

```
from datetime import date, datetime, timedelta


class State:
    StockStructure = {
        "ID": int,
        "name": str,
        "brand": str,
        "price": float,
        "qty": int,
    }
    RentalStructure = {
        "ID": int,
        "IID": int,
        "date_of_rental": str,
        "rental_unit_days": int,
        "rental_qty": int,
        "username": str,
        "contact": str,
    }
    EarningStructure = {
        "ID": int,
        "username": str,
        "contact": str,
        "price_per_unit": float,
        "total_price": float,
        "date_of_rental": str,
        "date_of_return": str,
        "rental_qty": int,
        "rental_unit_days": int,
    }
    DateFormat = "%Y-%m-%d"
    IN_NON_DELIMITED_CELL = 1
    IN_DELIMITED_CELL = 2


# This helps use use dot-notation. (Instead of using dict['key'], we'll also be able to use
dict.key)
class Map(dict):
    def __init__(self, *args, **kwargs):
        super(Map, self).__init__(*args, **kwargs)
        for arg in args:
            if isinstance(arg, dict):
                for k, v in arg.items():
                    self[k] = v

        if kwargs:
            for k, v in kwargs.items():
                self[k] = v

    def __getattr__(self, attr):
        return self.get(attr)

    def __setattr__(self, key, value):
        self.__setitem__(key, value)

    def __setitem__(self, key, value):
        super(Map, self).__setitem__(key, value)
        self.__dict__.update({key: value})

    def __delattr__(self, item):
        self.__delitem__(item)

    def __delitem__(self, key):
        super(Map, self).__delitem__(key)
        del self.__dict__[key]


def get_cell_values(line, quotechar="\"", delimiter=","):
    stack = []
    stack.append(State.IN_NON_DELIMITED_CELL)
    cell_values = [""]
    for character in line:
        current_state = stack[-1]
        if current_state == State.IN_NON_DELIMITED_CELL:
            if character == quotechar:
                stack.append(State.IN_DELIMITED_CELL)
            elif character == delimiter:
                cell_values.append("")
            else:
                cell_values[-1] += character

        if current_state == State.IN_DELIMITED_CELL:
            if character == quotechar:
                stack.pop()
            else:
                cell_values[-1] += character
    return cell_values


def parse_cell_values(cell_values, type="stocks"):
    parsed = {}
    if type.startswith("rent"):
        structure = State.RentalStructure
    elif type.startswith("stock"):
        structure = State.StockStructure
    elif type.startswith("earn"):
        structure = State.EarningStructure
    else:
        raise ValueError("Unknown type specified for structure of the cell_values.")
    for i in range(len(cell_values)):
        # print(i, cell_values, State.Structure.keys())
        key = tuple(structure.keys())[i]
        if not key.endswith("ID"):
            key = (" ".join(key.split("_"))).capitalize()
        value = tuple(structure.values())[i](cell_values[i])
        parsed[key] = value
    return Map(parsed)
```

```
def read(type="stocks"):
    if type.startswith("rent"):
        path = "./rentals.csv"
    elif type.startswith("stock"):
        path = "./stocks.csv"
    elif type.startswith("earn"):
        path = "./earnings.csv"
    else:
        raise ValueError("Unknown database collection specified in type argument.")
    db: list[dict] = []
    try:
        with open(path, "r") as f:
            for line in f:
                line = line.strip()
                cell_values = get_cell_values(line, "", ",")
                db.append(parse_cell_values(cell_values, type))
    except Exception as e:
        print(f"Exception -> {e}")
    return db


def commit(db, type="stocks"):
    if type.startswith("rent"):
        path = "./rentals.csv"
    elif type.startswith("stock"):
        path = "./stocks.csv"
    elif type.startswith("earn"):
        path = "./earnings.csv"
    else:
        raise ValueError("Unknown database specified in type argument.")
    try:
        with open(path, "w") as f:
            for entry in db:
                values = tuple(entry.values())
                values_str = []
                for value in values:
                    values_str.append(str(value))
                # values_str[1:] => Making sure ID isn't there.
                # f.write(",".join(values_str[1:]) + "\n")
                f.write(",".join(values_str) + "\n")
    except Exception as e:
        print(f"Exception -> {e}")
    return True


def print_table(db):
    colList = list(db[0].keys() if db else [])
    _list = [colList]  # 1st row = header
    for entry in db:
        _list.append(
            [str(entry[col] if entry[col] is not None else "") for col in colList]
        )
    colSize = [max(map(len, col)) for col in zip(*_list)]
    formatStr = " | ".join(["{{:<{}}}".format(i) for i in colSize])
    _list.insert(1, ["-" * i for i in colSize])  # Separating line
    for item in _list:
        print(formatStr.format(*item))


def today():
    return date.today()


def parse_date_str(date_str):
    if type(date_str) != str:
        date_str = str(date_str)
    return datetime.strptime(date_str, State.DateFormat).date()


def unit_days(_unit_days):
    return timedelta(days=_unit_days * 5)


def prompt_num(prompt, less_than=9, greater_than=0, save_an_entity=False):
    print(end="\n" * 2)
    try:
        user_input = input(prompt)
        user_input = int(user_input)
    except KeyboardInterrupt:
        exit_prompt()
        return prompt_num(prompt, less_than, greater_than, save_an_entity)
    except Exception as e:
        print(e)
        return prompt_num(prompt, less_than, greater_than, save_an_entity)
    if save_an_entity:
        # Saving an entity through >=
        while user_input >= less_than or user_input < greater_than:  # pyright: ignore
            user_input = prompt_num(prompt, greater_than, less_than)
    else:
        while user_input > less_than or user_input < greater_than:  # pyright: ignore
            user_input = prompt_num(prompt, greater_than, less_than)
    return user_input
```

```
def prompt_yn(prompt):
    print(end="\n" * 2)
    try:
        char = input(f"{prompt} [y/n]: ").lower()
        while char not in ("y", "n"):
            char = prompt_yn(prompt)
    except KeyboardInterrupt:
        exit_prompt()
        return prompt_yn(prompt)
    except Exception as e:
        print(e)
        return prompt_yn(prompt)
    return char


def exit_prompt(prompt="Do you wish to quit?"):
    char = prompt_yn(prompt)
    if char == "y":
        exit(0)
```