Notes on Proj 2:

Creating a Shopping App Using Python

Phase-End Project Problem Statement

simplilearn

Get Certified. Get Ahead.

Raised Ticket on clarification of problem statement's ID 0218 7252 on 15 Jan 24

Phase-End Project: Creating a Shopping App Using Python

Problem Scenario: You have to develop a shopping application or e-commerce application which has login and public login features on the Python platform. The applications that have been developed should also include categories, such as 3–4 for footwear, clothing, electronics, etc. It should be possible to add and update categories in the application. Additionally, it must contain a feature that allows you to add or remove items from your cart. Finally, the program needs to support a variety of payment options, including UPI and debit cards. This should be only backend implementation, and UX/UI and database connectivity are not required.

Guidelines:

- A welcome message should initially be displayed in the e-commerce application, such as "Welcome to the Demo Marketplace".
- User login and admin login should be created once the code for the welcome message has been written. For the creation of demo login and admin login, demo databases for those should be created for the user and admin verification, and session id creation.
- After databases are created, create admin and user logins. It is necessary to construct a sample product catalog with three to four product categories, such as Boots, Coats, Jackets, and Caps. The product id, name, category id, and price should all be present for each item in the dummy database of the catalog. Both users and administrators can view the catalog.
- User login rights include the ability to view cart contents, add items to carts, and remove items from carts. The user should be able to add items or delete items in the cart using session id, product id, and quantity.
- Next, the program should provide demo payment checkout options, like Net banking, PayPal, UPI, etc. After selecting the payment option, a checkout message like, "Your order is successfully placed" or "You will be shortly redirected to the portal for Unified Payment Interface to make a payment of Rs. 2000", etc., should be displayed.
- Additionally, the admin can only log in using his credentials, and if the admin attempts to log in using another set of credentials, an error notice must appear.
- Admin should not be able to interfere with any of the functions that the user can perform, as discussed above. An error should appear if the admin tries to carry out those tasks.
- Furthermore, using the previously selected attributes, the admin should be able to add new products to the catalog. Additionally, the program needs to make it possible for an existing product to be modified using an admin session id.
- The admin should then have the ability to remove any existing products from the already-generated catalog.
- Lastly, understanding the dynamic demands of the market, the admin should be able to add a new category of product and delete the prevailing category of product from the catalog.
- Users should not be able to take advantage of any of the admin's rights, as described above.

I have the following queries in problem statement comprehension. I have refered my pts to previous page

① I understand the database for login ID & password of user & admin is to be prepopulated. The word demo is creating confusion.

② Login Rights. How are we to assign rights to data? This would entail data encapsulation into a OOPs mode

In my understanding this looks like a simple problem of multiple functions. If a add to cart function is not in the work flow of a admin user, asents rights auto addressed.

③ Session ID → Rand for ok?
Or I import UUID lib.
Your guys should have explained this?

④ How is this program going to be dynamic? With multiple users logging in simul? We have only 1 client single flow. Are you to thread?
There is a lot of jargon and loose english.
I have drawn my comprehension of your intent diagramatically on next page.
Please let me know if its ok
- Are you to do this in OOPs mode with threading & encapsulation?

Design Plan of App

⇒ Run as single program without making module & seperate method.

I] Create dictionary

{products} [categories] {user creds.} {user cart}

Session ID creation Method.

Admin login fn()

Calls fns to
- amend update stocks
- update/del catagories

User login

Fn manage cart

Sub fn:
- add
- del item

UserLogin()

→ Main() Dashboard with login

No response on ticket by 17 Jan

✓ Submitted 17 Jan.

Code & notes att

Key take aways.

→ Data Structure was wrong. Should have better selected based on retrival. Use Key's & values in dic effectively

→ Lists are better if data is defined & limited like this. Code is more slick.

→ Need physical copy of cheat sheet dic, string & list function. Confusing.

→ Current program used complex structure of [{}, {}, {}] better make it {key1: {}, key2 {} ...}

→ Dashboards are laborious to type make snippets for reuse

⇒ Use AI effectively if logic is clear. It generates full code on most common use cases That may be semantically wrong but syntax & self documented code is on the spot.

→ For long program it never works. So split it function wise.

→ Great resource if logic is right.

→ Redrafting the problem statement into what AI understands got good result.

⇒ Global variable. Wasted 3 hours in debugging

for lack of understanding of this concept.

- A python function treats all variable as local even if you have declared it outside.
- Got program to work after declaring the variable as global within the function.

- List comprehension learnt on coursera is fantastic. Keep cheat sheet handy. Saved in google drive. Saves typing time in interview.
- After business logic is clear it is business time coding & debugging. Each tab is important

- Use tab guidelines.
- Python dics are great.
  Code for retrieval is easy
  provided you structure it
  well accordig to business
  logic.
- Input restrictions need to
  be thought off in advance as
  when realised during debugging
  effort multiplies
  Eg: Deleting a category
  already associated with a
  item in dictonary in
  mad cat process
- Jupyter notebook in jupyter.

Submission of write up (1)

WRITE UP: ON PROJECT II PYTHON REFRESHER
COL RAKESH PEDRAM: NOV 2023 COHORT

Approach

The business logic of the function of a shopping cart were broken down logically into step. The following distinct functions were identified.

- Authentication of user. And access rights
- Shopping cart
    - Adding items
    - Deleting items
    - Check out
- Payment
    - Gateway choice
    - Amend stock
- Admin function
    - Add Product
    - Mod Products
        - Checks to ensure data integrity
    - Rem products
    - Add categories
    - Rem categories
        - Logic to ensure items in existing catlog not linked to cat marked for del

The software was designed as a single module and both the calling method and code was in the same Jupyter block

Authentication:

Used a UUID libraray to gen session code that I stored in

two variables one each for admin and user. A list called credentials was made to enter all login id passwords to authenticate. There are seperate functions for login user and login in admin which are called from a dashboard from the main() function.

Data Storage.
This was a major problem. With lot of iterations wasting huge time deciding between either list or dictionary.
Finally used dictionaries for user cart and Main stock and list for product categories.

There are checks to ensure user enters correct data and cues are given for user to enter.
The logic has been tested with copious test scenarios and all the authentication update of data is in place.
I used a random alphabet generator to add product id to new products. Line 329

Known defi:
There is no dynamism in the program as both the user and admin cant login simultaneously.
Though session ID checks have been kept.
There are no known logic errors in the program which has been extensively tested over three days.

Takeaways:
Seemingly looked very rudimentary when I started but handling the data in dictionaries was really difficult and constant references made to syntax.
I have taken code snippets online to save the typing.
Logic and structure is largely original. I think dictionary in a dictionary is a very bad way of structuring the data. I was

just stuck with it.
This has taken thrice the effort as Project 1 and 20 manhours of testing at my skill set.
Good project I enjoyed it.

```
#Print Categories
    for index,value in enumerate(categories):
        print('Existing cat'index ',index,' ',value,end=' ')
```

Data Structure followed

This was bad way to create data set. List of dictonaries. There is no integrity of the unique value of product ID

```
products = [
    {
        "product_id": 'a',
        "product_name": "Chelsea Boots",
        "category_id": 0,
        "price": 1000,
        "stock": 30,
    },
    {
        "product_id": 'b',
        "product_name": "Winter Coat",
        "category_id": 1,
        "price": 2000,
        "stock": 30,
    },
    {
        "product_id": 'c',
        "product_name": "Denim Jacket",
        "category_id": 2,
        "price": 1500,
        "stock": 30,
    },
    {
        "product_id": 'd',
        "product_name": "Beanie",
        "category_id": 3,
        "price": 500,
        "stock": 30,
    },
]
for product in products:
    print(product['category_id'])
print([product['product_id'] for product in products])
print(products[0])
temp ={ "product_id": 5,
        "product_name": "loard",
        "category_id": 2,
        "price": 576,}
products.append(temp)
print (products)
print([product['product_name']for product in products if product['product_id'] =='a'])
for product in products:
    if product['category_id']==2: print(product['price'])
list2=[product['category_id'] for product in products]
print('list of category ID print:',list2)
# Exiting function
user_cart ={'a':20,'b':65}
for product_id in user_cart:
        for product in products:
            if product['product_id']==product_id:
                product['stock']=product['stock']-user_cart[product_id]
                print(product['stock'])
```

```
0
1
2
3
['a', 'b', 'c', 'd']
{'product_id': 'a', 'product_name': 'Chelsea Boots', 'category_id': 0, 'price': 1000, 'stock': 30}
```

```
[{'product_id': 'a', 'product_name': 'Chelsea Boots', 'category_id': 0, 'price': 1000, 'stock': 30}, {'product_id': 'b', 'product_
['Chelsea Boots']
1500
576
list of category ID print: [0, 1, 2, 3, 2]
10
-35
```

Cheat sheet for dictionary

Operations len(dictionary) - Returns the number of items in a dictionary.

for key, in dictionary - Iterates over each key in a dictionary.

for key, value in dictionary.items() - Iterates over each key,value pair in a dictionary.

if key in dictionary - Checks whether a key is in a dictionary.

dictionary[key] - Accesses a value using the associated key from a dictionary.

dictionary[key] = value - Sets a value associated with a key.

del dictionary[key] - Removes a value using the associated key from a dictionary.

Methods dictionary.get(key, default) - Returns the value corresponding to a key, or the default value if the specified key is not present.

dictionary.keys() - Returns a sequence containing the keys in a dictionary.

dictionary.values() - Returns a sequence containing the values in a dictionary.

dictionary[key].append(value) - Appends a new value for an existing key.

dictionary.update(other_dictionary) - Updates a dictionary with the items from another dictionary. Existing entries are updated; new entries are added.

dictionary.clear() - Deletes all items from a dictionary.

dictionary.copy() - Makes a copy of a dictionary.

```

ls={'0':100,'1':30,'2':34,'78':2}
for values in ls:
    print(values)
ls.get('0')
ls['0']=150
print (ls)
```

```
categories = ["Boots", "Coats", "Jackets", "Caps"]
for index,value in enumerate(categories):
    print('index ',index,' ',value,end=' ')
```

```
import random
rndstring=""
for i in range(5):
   rndstring=rndstring + chr(random.randint(ord('a'),ord('z')))
print(rndstring)
```

Ideal Data Structure

This dictonary structure is buch better. Ensure uniqueness of primary key { key:{dic1},key:{dic2}.... Accesibility to data is quick. Will save massive effort in writing code.

```
products = {
    'a':{
        "product_id": 'a',
        "product_name": "Chelsea Boots",
        "category_id": 0,
        "price": 1000,
        "stock": 30,
    },
    'b':{
        "product_id": 'b',
        "product_name": "Winter Coat",
        "category_id": 1,
        "price": 2000,
        "stock": 30,
    },
    'c':{
        "product_id": 'c',
        "product_name": "Denim Jacket",
        "category_id": 2,
        "price": 1500,
        "stock": 30,
    },
    'd':{
        "product_id": 'd',
        "product_name": "Beanie",
        "category_id": 3,
        "price": 500,
        "stock": 30,
    },
}
print(products['a']['category_id'])
```

```
0
```

Should have followed this

Tested much later. Too tired to recode.

Feed this prompts in AI

Prompts for AI

Write python code for a backend basic application for a Shopping cart. Business logic and work flow was under:

Opens app with a message 'Welcome to AI Shopping'

Make default dictonaries of catalogue {key1:{},key2:{}}, Male a list of categories=[cat1,cat2,cat3]

Create two functions for login of user and admin. This function should set session ID and check user name and password

Create a function to display catalogue

Create a function to manage cart. This should take parameter session id. It should give drop down options of 1. Add item, 2. Remove items 3. Checkout

Create function to add items to cart. This should check stock in catalogue and return to manage cart. And add item to the user cart dictonary.

Create function to remove item from cart. This should remove irem from cart if product ID exists.

Create function to check out. This should list available option as UPI, netbanking,credit and after that amend stock in cart. Reset cart dictionary and return to main menu.

Admin Function

Create separate function to add modify delete products in catalogue also add or delete categories

main() This should display a dashboard with options to either log in user or admin.

Bing code generation tried after completing my proj

```
# Import random module to generate session IDs
import random

# Define a global variable for the catalogue dictionary
catalogue = {
    "cat1": {
        "key1": {"name": "Product 1", "price": 10, "stock": 5},
        "key2": {"name": "Product 2", "price": 20, "stock": 10},
    },
```

```
    "cat2": {
        "key3": {"name": "Product 3", "price": 30, "stock": 15},
        "key4": {"name": "Product 4", "price": 40, "stock": 20},
    },
    "cat3": {
        "key5": {"name": "Product 5", "price": 50, "stock": 25},
        "key6": {"name": "Product 6", "price": 60, "stock": 30},
    },
}

# Define a global variable for the categories list
categories = ["cat1", "cat2", "cat3"]

# Define a global variable for the user cart dictionary
user_cart = {}

# Define a global variable for the session ID
session_id = None

# Define a function to login as a user or an admin
def login():
    # Ask the user to enter their username and password
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Check if the username and password are valid
    # For simplicity, we assume that the username and password are both "user" for a user and "admin" for an admin
    if username == "user" and password == "user":
        # Generate a random session ID for the user
        global session_id
        session_id = random.randint(1000, 9999)
        print(f"Welcome, user. Your session ID is {session_id}.")
        # Display the catalogue for the user
        display_catalogue()
    elif username == "admin" and password == "admin":
        # Welcome the admin and display the admin options
        print("Welcome, admin.")
        display_admin_options()
    else:
        # Display an error message and ask the user to try again
        print("Invalid username or password. Please try again.")
        login()

# Define a function to display the catalogue for the user
def display_catalogue():
    # Print the catalogue details
    print("Here is the catalogue:")
    for category, products in catalogue.items():
        print(f"{category}:")
        for key, product in products.items():
            print(f"{key}: {product['name']} - ${product['price']} - {product['stock']} in stock")
    # Display the cart options for the user
    display_cart_options()

# Define a function to display the cart options for the user
def display_cart_options():
    # Print the cart options
    print("What would you like to do?")
    print("1. Add item to cart")
    print("2. Remove item from cart")
    print("3. Checkout")
    print("4. Logout")

    # Ask the user to choose an option
    option = input("Enter your choice: ")

    # Validate the user input
    if option == "1":
        # Call the add_item function
        add_item()
    elif option == "2":
```

I didn't like this structure too

X

Global var

What I focused in my note should penalty to learn.

AI helps learning

Self documented code & good coding practices

See all comments like english. code reads

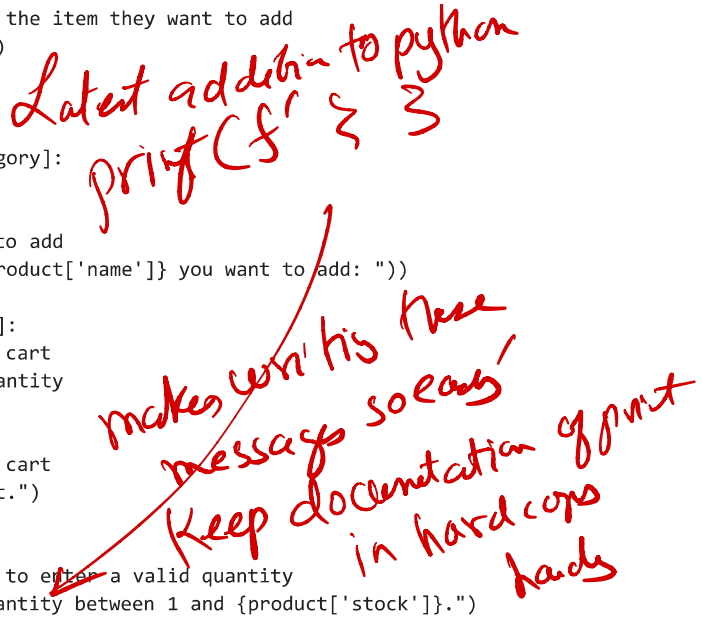
```
    elif option == "2":
        # Call the remove_item function
        remove_item()
    elif option == "3":
        # Call the checkout function
        checkout()
    elif option == "4":
        # Call the logout function
        logout()
    else:
        # Display an error message and ask the user to choose again
        print("Invalid option. Please choose again.")
        display_cart_options()

# Define a function to add an item to the cart
def add_item():
    # Ask the user to enter the category and the key of the item they want to add
    category = input("Enter the category of the item: ")
    key = input("Enter the key of the item: ")

    # Validate the user input
    if category in categories and key in catalogue[category]:
        # Get the product details from the catalogue
        product = catalogue[category][key]
        # Ask the user to enter the quantity they want to add
        quantity = int(input(f"Enter the quantity of {product['name']} you want to add: "))
        # Validate the user input
        if quantity > 0 and quantity <= product["stock"]:
            # Add the item and the quantity to the user cart
            user_cart[key] = user_cart.get(key, 0) + quantity
            # Update the stock in the catalogue
            product["stock"] -= quantity
            # Display a success message and the updated cart
            print(f"{product['name']} added to your cart.")
            display_cart()
        else:
            # Display an error message and ask the user to enter a valid quantity
            print(f"Invalid quantity. Please enter a quantity between 1 and {product['stock']}.")
            add_item()
    else:
        # Display an error message and ask the user to enter a valid category and key
        print("Invalid category or key. Please enter a valid category and key.")
        add_item()

# Define a function to remove an item from the cart
def remove_item():
    # Ask the user to enter the key of the item they want to remove
    key = input("Enter the key of the item you want to remove: ")

    # Validate the user input
    if key in user_cart:
        # Get the product details from the user cart
        product = user_cart[key]
        # Ask the user to enter the quantity they want to remove
        quantity = int(input(f"Enter the quantity of {product['name']} you want to remove: "))
        # Validate the user input
        if quantity > 0 and quantity <= product["qty"]:
            # Remove the item and the quantity from the user cart
            user_cart[key] -= quantity
            # Update the stock in the catalogue
            product["stock"] += quantity
            # Display a success message and the updated cart
            print(f"{product['name']} removed from your cart.")
            display_cart()
        else:
            # Display an error message and ask the user to enter a valid quantity
            print(f"Invalid quantity. Please enter a quantity between 1 and {product['qty']}.")
            remove_item()
    else:
        # Display an error message and ask the user to enter a valid key
        print("Invalid key. Please enter a valid key.")
```



```
151         remove_item()
152
153 # Define a function to display the cart for the user
154 def display_cart():
155     # Print the cart details
156     print("Here is your cart:")
157     for key, qty in user_cart.items():
158         # Get the product details from the catalogue
159         product = catalogue[key]
160         # Calculate the subtotal for the item
161         subtotal = product["price"] * qty
162         # Print the item details and the subtotal
163         print(f"{key}: {product['name']} - ${product['price']} - {qty} in cart - Subtotal: ${subtotal}")
164     # Display the cart options for the user
165     display_cart_options()
166
167 # Define a function to checkout the cart for the user
168 def checkout():
169     # Calculate the total price of the cart
170     total_price = 0
171     for key, qty in user_cart.items():
172         # Get the product details from the catalogue
173         product = catalogue[key]
174         # Add the subtotal for the item to the total price
175         total_price += product["price"] * qty
176     # Print the total price
177     print(f"Your total price is ${total_price}.")
178     # Ask the user to choose a payment option
179     print("How would you like to pay?")
180     print("1. UPI")
181     print("2. Netbanking")
182     print("3. Credit")
183     # Get the user input
184     option = input("Enter your choice: ")
185     # Validate the user input
186     if option == "1":
187         # Ask the user to enter their UPI ID
188         upi_id = input("Enter your UPI ID: ")
189         # Display a success message and clear the cart
190         print(f"Payment successful. Thank you for shopping with us, {upi_id}.")
191         clear_cart()
192     elif option == "2":
193         # Ask the user to enter their bank name and account number
194         bank_name = input("Enter your bank name: ")
195         account_number = input("Enter your account number: ")
196         # Display a success message and clear the cart
197         print(f"Payment successful. Thank you for shopping with us, {bank_name} {account_number}.")
198         clear_cart()
199     elif option == "3":
200         # Ask the user to enter their card number and CVV
201         card_number = input("Enter your card number: ")
202         cvv = input("Enter your CVV: ")
203         # Display a success message and clear the cart
204         print(f"Payment successful. Thank you for shopping with us, {card_number} {cvv}.")
205         clear_cart()
206     else:
207         # Display an error message and ask the user to choose again
208         print("Invalid option. Please choose again.")
209         checkout()
210
211 # Define a function to clear the cart for the user
212 def clear_cart():
213     # Reset the user cart to an empty dictionary
214     global user_cart
215     user_cart = {}
216     # Display a goodbye
217
```

I didnt cart items in cart.

Not difficult though Just need to get from product ID w/o mod data str.

This is a slick way instead of a perpetual While loop

Now you have a most likely semantically flawed code but perfect syntax. Saves huge effort. You got the framework now make it work. Fast but saves typing. generate additional functions using AI.

✓ Syntax is spot on.
Logic sound but incomplete.

✓ There are code snippets & they never work.

✓ Useful to understand structure.

✓ Also suggested good practical code on github with OOPs & encapsulation but thats too too dense.

✓ Had I used the AI code before I'd have saved time.

☒ Don't reinvent wheel, jump from others shoulders

```
#Print Categories
    for index,value in enumerate(categories):
        print('Existing cat'index ',index,' ',value,end=' ')
```

Data Structure followed

This was bad way to create data set. List of dictonaries. There is no integrity of the unique value of product ID

```
products = [
    {
        "product_id": 'a',
        "product_name": "Chelsea Boots",
        "category_id": 0,
        "price": 1000,
        "stock": 30,
    },
    {
        "product_id": 'b',
        "product_name": "Winter Coat",
        "category_id": 1,
        "price": 2000,
        "stock": 30,
    },
    {
        "product_id": 'c',
        "product_name": "Denim Jacket",
        "category_id": 2,
        "price": 1500,
        "stock": 30,
    },
    {
        "product_id": 'd',
        "product_name": "Beanie",
        "category_id": 3,
        "price": 500,
        "stock": 30,
    },
]
for product in products:
    print(product['category_id'])
print([product['product_id'] for product in products])
print(products[0])
temp ={ "product_id": 5,
        "product_name": "loard",
        "category_id": 2,
        "price": 576,}
products.append(temp)
print (products)
print([product['product_name']for product in products if product['product_id'] =='a'])
for product in products:
    if product['category_id']==2: print(product['price'])
list2=[product['category_id'] for product in products]
print('list of category ID print:',list2)
# Exiting function
user_cart ={'a':20,'b':65}
for product_id in user_cart:
        for product in products:
            if product['product_id']==product_id:
                product['stock']=product['stock']-user_cart[product_id]
                print(product['stock'])
```

```
0
1
2
3
['a', 'b', 'c', 'd']
{'product_id': 'a', 'product_name': 'Chelsea Boots', 'category_id': 0, 'price': 1000, 'stock': 30}
```

```
[{'product_id': 'a', 'product_name': 'Chelsea Boots', 'category_id': 0, 'price': 1000, 'stock': 30}, {'product_id': 'b', 'product_
['Chelsea Boots']
1500
576
list of category ID print: [0, 1, 2, 3, 2]
10
-35
```

Cheat sheet for dictionary

Operations len(dictionary) - Returns the number of items in a dictionary.

for key, in dictionary - Iterates over each key in a dictionary.

for key, value in dictionary.items() - Iterates over each key,value pair in a dictionary.

if key in dictionary - Checks whether a key is in a dictionary.

dictionary[key] - Accesses a value using the associated key from a dictionary.

dictionary[key] = value - Sets a value associated with a key.

del dictionary[key] - Removes a value using the associated key from a dictionary.

Methods dictionary.get(key, default) - Returns the value corresponding to a key, or the default value if the specified key is not present.

dictionary.keys() - Returns a sequence containing the keys in a dictionary.

dictionary.values() - Returns a sequence containing the values in a dictionary.

dictionary[key].append(value) - Appends a new value for an existing key.

dictionary.update(other_dictionary) - Updates a dictionary with the items from another dictionary. Existing entries are updated; new entries are added.

dictionary.clear() - Deletes all items from a dictionary.

dictionary.copy() - Makes a copy of a dictionary.

```

ls={'0':100,'1':30,'2':34,'78':2}
for values in ls:
    print(values)
ls.get('0')
ls['0']=150
print (ls)
```

```
categories = ["Boots", "Coats", "Jackets", "Caps"]
for index,value in enumerate(categories):
    print('index ',index,' ',value,end=' ')
```

```
import random
rndstring=""
for i in range(5):
   rndstring=rndstring + chr(random.randint(ord('a'),ord('z')))
print(rndstring)
```

```
#**Ideal Data Structure**
This dictonary structure is buch better. Ensure uniqueness of
primary key
{ key:{dic1},key:{dic2}....
Accesibility to data is quick. Will save massive effort in
writing code.
```

Ideal Data Structure

This dictonary structure is buch better. Ensure uniqueness of primary key { key:{dic1},key:{dic2}.... Accesibility to data is quick. Will save massive effort in writing code.

```
products = {
    'a':{
```

```
    'a':{
        "product_id": 'a',
        "product_name": "Chelsea Boots",
        "category_id": 0,
        "price": 1000,
        "stock": 30,
    },
    'b':{
        "product_id": 'b',
        "product_name": "Winter Coat",
        "category_id": 1,
        "price": 2000,
        "stock": 30,
    },
    'c':{
        "product_id": 'c',
        "product_name": "Denim Jacket",
        "category_id": 2,
        "price": 1500,
        "stock": 30,
    },
    'd':{
        "product_id": 'd',
        "product_name": "Beanie",
        "category_id": 3,
        "price": 500,
        "stock": 30,
    },
}
print(products['a']['category_id'])
```

```
0
```