WHERE YOU WILL LEVERAGE DICTIONARY?

You can leverage dictionaries in various scenarios where key-value pairs or efficient lookup are required. Here are some common use cases where dictionaries can be leveraged effectively:

1. Data Storage and Retrieval: Dictionaries are often used as a data structure to store and retrieve information based on keys. You can leverage dictionaries to organize and access data efficiently, especially when you need to associate values with specific identifiers or labels.
2. Configuration and Settings: Dictionaries are commonly used to store configuration parameters and settings. By leveraging a dictionary, you can easily access and modify the settings based on the corresponding keys. This allows for flexible and customizable behavior in your programs.
3. Counting and Grouping: Dictionaries are useful for counting occurrences or grouping items based on specific criteria. By using keys as identifiers and values to track counts or groupings, you can leverage dictionaries to perform efficient counting operations or organize data based on different categories.
4. Lookup Tables and Mapping: Dictionaries can serve as lookup tables or mapping structures, allowing you to map one set of values to another. By leveraging dictionaries, you can quickly retrieve corresponding values based on keys, making them useful for tasks like data transformation, code translation, or data enrichment.
5. Caching and Memoization: Dictionaries can be used to implement caching or memoization techniques, where you store computed results based on specific inputs. By leveraging a dictionary, you can store previously calculated values and retrieve them quickly, avoiding unnecessary recomputation.
6. Efficient Searching and Indexing: Dictionaries provide fast lookup by key, making them efficient for searching and indexing tasks. You can leverage dictionaries to build indexes or mappings that allow you to retrieve information quickly based on specific keys or criteria.

EXAMPLE :

Let's say you have a program that tracks the inventory of items in a store. You need to keep a record of the quantity and price of each item. In this case, you can leverage a dictionary to store the item details, using the item name as the key and a dictionary of quantity and price as the value.

```
inventory = {}
```

```
def add_item(name, quantity, price):
    if name in inventory:
        print(f"Item '{name}' already exists in the inventory.")
    else:
        inventory[name] = {'quantity': quantity, 'price': price}
        print(f"Item '{name}' added to the inventory.")

def remove_item(name):
    if name in inventory:
        del inventory[name]
        print(f"Item '{name}' removed from the inventory.")
    else:
        print(f"Item '{name}' does not exist in the inventory.")

def update_quantity(name, new_quantity):
    if name in inventory:
        inventory[name]['quantity'] = new_quantity
        print(f"Quantity updated for item '{name}'.")
    else:
        print(f"Item '{name}' does not exist in the inventory.")

def update_price(name, new_price):
    if name in inventory:
        inventory[name]['price'] = new_price
        print(f"Price updated for item '{name}'.")
    else:
        print(f"Item '{name}' does not exist in the inventory.")

def get_item_details(name):
    if name in inventory:
        item = inventory[name]
        print(f"Item: {name}")
        print(f"Quantity: {item['quantity']}")
        print(f"Price: {item['price']}")
    else:
        print(f"Item '{name}' does not exist in the inventory.")

# Add items to the inventory
add_item("Apple", 10, 0.99)
add_item("Banana", 15, 0.5)
add_item("Orange", 8, 0.75)

# Get details of an item
get_item_details("Apple")

# Update quantity of an item
update_quantity("Banana", 20)

# Remove an item from the inventory
remove_item("Orange")

# Get details of an item again
get_item_details("Orange")
```

- In this example, the **inventory** dictionary is leveraged to store the item details. The **add_item()** function adds a new item to the inventory, the **remove_item()** function removes an item, the **update_quantity()** and **update_price()** functions update the quantity and price of an item, and the **get_item_details()** function retrieves and displays the details of an item.