Ex.no:5 13.02.2025	DICTIONARY MANIPULATION	URK24CS1189

A) AIM:

Write a python program to sort, remove and add the key in the given dictionary**.**

ALGORITHM:

STEP 1: Start the program
STEP 2: initialize the inventory dictionary with keys and values.
STEP 3: sort list stored in the “backpack” key.
STEP 4: remove the item “dagger” from “backpack”
STEP 5: increase the value of “gold” by 50.
STEP 6: print the updated inventory after each operation.
STEP 7: end the program.

PROGRAM:

```
#URK24CS1189
#inventorydict
inventory = {'gold': 500, 'pouch': ['flint', 'twine', 'gemstone'],
             'backpack': ['xylophone', 'dagger', 'bedroll']}
inventory['pocket'] = ['seashell', 'strange berry', 'lint']
inventory['backpack'].sort()
print("inventory after sorting of a key backpack", inventory, "\n")
inventory['backpack'].remove("dagger")
print("inventory after removing a value dagger in key backpack", inventory, '\n'
inventory['gold'] += 50
print("inventory after adding the value 50 to the key gold", inventory, '\n')
```

OUTPUT:

```
inventory after sorting of a key backpack {'gold': 500, 'pouch': ['flint',
    'twine', 'gemstone'], 'backpack': ['bedroll', 'dagger', 'xylophone'],
    'pocket': ['seashell', 'strange berry', 'lint']}

inventory after removing a value dagger in key backpack {'gold': 500,
    'pouch': ['flint', 'twine', 'gemstone'], 'backpack': ['bedroll',
    'xylophone'], 'pocket': ['seashell', 'strange berry', 'lint']}

inventory after adding the value 50 to the key gold {'gold': 550, 'pouch':
    ['flint', 'twine', 'gemstone'], 'backpack': ['bedroll', 'xylophone'],
    'pocket': ['seashell', 'strange berry', 'lint']}


=== Code Execution Successful ===
```

RESULT:

Thus the python program successfully verified .

B) AIM:

create a new dictionary called prices using {} format like the example above.

ALGORITHM:

STEP 1: Start the program
STEP 2: initialize two dictionaries: prices(stores item prices) and stock (stores item quantities)
STEP 3: print each item's price and stock.
STEP 4: initialize total revenue to 0.
STEP 5: calculate revenue for each item by multiplying price and stock.
STEP 6:print the revenue for each item.
STEP 7: update the total revenue.
STEP 8: print the total revenue.
STEP 9: end the program.

PROGRAM:

```
#URK24CS1189
prices = {"banana": 4, "apple": 2, "orange": 1.5, "pear": 3}
stock = {"banana": 6, "apple": 0, "orange": 32, "pear": 15}
for i in prices:
    print(f"{i} price: {prices[i]} stock: {stock[i]}")
total = 0
for i in prices:
    value = prices[i] * stock[i]
    print(f"revenue from {i}: {value}")
    total += value
print(f"total revenue: {total}")
```

OUTPUT:

```
banana price: 4 stock: 6
apple price: 2 stock: 0
orange price: 1.5 stock: 32
pear price: 3 stock: 15
revenue from banana: 24
revenue from apple: 0
revenue from orange: 48.0
revenue from pear: 45
total revenue: 117.0

=== Code Execution Successful ===
```

RESULT:

Thus the python program successfully verified.

C) AIM:

Write a python program to find the total bill and update the stock for the given list.

ALGORITHM:

STEP 1: Start the program
STEP 2: initialize groceries list,stock dictionary, and prices dictionary.
STEP 3: define function compute_bill (food):
(I) Set total=0
(II) For each item in food:
(III) If the item is in stock(stock[i]>0
(IV) Add its price to total
(V) Decrease its stock by 1.
STEP 4: call compute_bill (groceries) and store the result in total_bill.
STEP 5: print the total bill
STEP 6: print the updated stock.
STEP 7: End the program.

PROGRAM:

```
#URK24CS1189
groceries = ["banana", "orange", "apple"]
stock = {"banana": 6, "apple": 0, "orange": 32
    , "pear": 15}
prices = {"banana": 4, "apple": 2, "orange": 1
    .5, "pear": 3}
def compute_bill(food):
    total = 0
    for i in food:
        if stock[i] > 0:
            total += prices[i]
            stock[i] -= 1
    return total
total_bill = compute_bill(groceries)
print(f"total bill: {total_bill}")
print("updated stock:", stock)
```

OUTPUT:

```
total bill: 5.5
updated stock: {'banana': 5, 'apple': 0, 'orange':
    31, 'pear': 15}

=== Code Execution Successful ===
```

RESULT:
Thus the python program successfully verified.

D) AIM:

Write a python program to find the class average and the grade in letter.

ALGORITHM:

STEP 1: Start the program
STEP 2: initialize a list of students with names and their scores for homework , quizzes and tests.
STEP 3: define average (numbers) and compute the average of a list.
STEP 4: define get_average (students) and compute weighted average 10% homework,30% quizzes, 60% tests.
STEP 5: define get_letter_grade of score and assign letter grades based on score.
STEP 6: define get_grade_average of students and compute the class average.
STEP 7: print the class average and its corresponding letter grade.
STEP 8: End the program.

PROGRAM:

```
#URK24CS1189
#Grade book of teacher's student
students = [{"name": "Lloyd", "homework": [90, 97, 75, 92], "quizzes": [88, 40, 94], "tests": [75, 90]},
    {"name": "Alice", "homework": [100, 92, 98, 100], "quizzes": [82, 83, 91], "tests": [89, 97]},
    {"name": "Tyler", "homework": [0, 87, 75, 22], "quizzes": [0, 75, 78], "tests": [100, 100]}]
def average(numbers):
    return sum(numbers) / len(numbers)
def get_average(student):
    return (average(student["homework"]) * 0.1) + (average(student["quizzes"]) * 0.3) + (average
        (student["tests"]) * 0.6)
def get_letter_grade(score):
    return "A" if score >= 90 else "B" if score >= 80 else "C" if score >= 70 else "D" if score >= 60 else
        "F"
def get_class_average(students):
    return average([get_average(s) for s in students])
print(f"Class Average: {get_class_average(students):.2f}")
print(f"Class Grade: {get_letter_grade(get_class_average(students))}")
```

OUTPUT:

```
Class Average: 83.87
Class Grade: B

=== Code Execution Successful ===
```

RESULT:
Thus the python program successfully verified.