

Task 10-4.1

Date :- 20/8/25

CAFETERIA SALES

Aim:- Record a cafeteria's snack sales for 7 days using a list; compute total and average sales, find the best/worst day, and count how many days crossed a target.

Algorithm:-

- Start
- create an empty list sales = []
- for 7 days, append integer sales to the list using append.
- compute total = sum(sales) and avg = total/7.
- find max_val = max(sales), min_val = min(sales).
- find corresponding days with index()
- count days above a target using count() on a boolean remap or with a loop.
- stop

Program:- (uses append(), index(), count());

list scenario

days = 7

sales = []

target = 500 # target sales for the day.

for s in range(8):

sample_entries = int(input("enter the seven days sales count"))

sales.append(sample_entries) # list.append()

total = sum(sales)

avg = total / days

max_val = max(sales)

min_val = min(sales)

Sample Input/output:

enter the seven days sales count : 100
enter the seven days sales count : 450
enter the seven days sales count : 1250
enter the seven days sales count : 98
enter the seven days sales count : 348
enter the seven days sales count : 900
enter the seven days sales count : 239

sales (mon. - sun) : (100, 450, 1250, 589, 98, 348, 900, 239)

Total : 3974

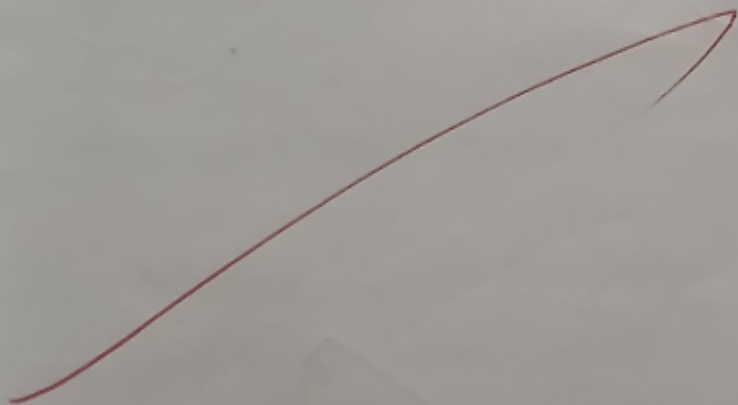
Average : 567.71

Best Day : 3 with 1250

worst day : 5 with 98

```
best-day = sales.index(max-val) + 1 # list.index()  
worst-day = sales.index(min-val) + 1
```

```
print ("Sales (Mon--Sun):", sales)  
print ("Total:", total)  
print ("Average:", round(avg, 2))  
print ("Best Day:", best-day, "with", max-val)  
print ("Worst Day:", worst-day, "with", min-val)
```



Task-4.2

TUPLE - LAB TIMETABLE

Aim :- To manage and query and immutable daily lab slot schedule using a tuple, demonstrating membership, `count()`, `index()`, and slicing.

Algorithm :-

- Start
- Define slots as a fixed tuple of integers.
- Read query hour.
- Check existence with query in slots.
- Use `count()`; if positive, use `index()` to find the first position.
- Slice into morning and afternoon
- Print results.
- Stop.

Program:-

```
#! TUPLE scenario
slots = (9, 11, 14, 16, 14) # immutable daily schedule
query = 14
exists = (query in slots)
freq = slots.count(query) # tuple.count()
first_pos = slots.index(query) + 1 if exists else "N/A" # tuple.index()

morning = slots[:2]
afternoon = slots[2:]

print("All lab slots:", slots)
print(f"Is {query}:00 present?", exists)
print(f"{query}:00 occurs", freq, "time(s)")
print("first occurrence position (1-based) :", first_pos)
print("Morning slots:", morning)
print("Afternoon slots:", afternoon)
```

sample Input/output:-

All lab slots: (9, 11, 14, 16, 14)

Is 14:00 present? True

14:00 occurs 2 times(s)

first occurrence position (1-based): 3

morning slots: (9, 11)

Afternoon slots: (14, 16, 14)



DICTIONARY - BOOKSTORE BILLING

Aim:-

to manage a like price list and bill a customer using dictionary methods and views.

Algorithm :-

- start
- Create an empty dictionary prices.
- Ask the user for the number of items in the price list
- Repeat for each item:
 - Get the item name.
 - Get the item price.
 - Add the item and price to prices.
- Ask the user for an item to update.
- If the item exists in prices, get the new price and update it.
- find the costliest item by checking each item's price.
- Ask the user for an item to remove.
- If given, remove that item from prices.
- show all available items, their prices, the costliest item, and the removed item's price.
- stop.

Program :-

prices = {}

n1 = int(input("enter number of items in price list:"))
for i in range(n1):

item = input("enter item name:")

price = float(input(f"enter price of {item}:"))

prices[item] = price

output:-

Enter the no. of item in the list: 3

Enter item name = box

Enter price of box: 15

Enter item name: pen

Enter price of pen: 10

Enter item name = pencil

Enter price of pencil: 15

Enter item to update price 'box'

Enter new price for box: 20

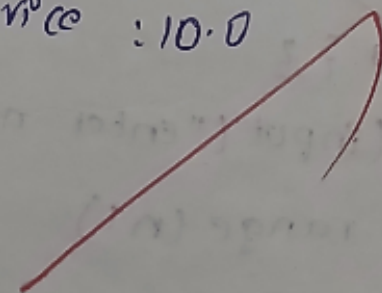
Enter an item from price list: pen

available items: ['box', 'pencil']

Prices: [20.0, 15.0]

Costliest item: box at 20.0

Removed 'pen' price: 10.0



```
# optional price revision
rev-item = input("enter item to update price (or press enter to skip):")
```

```
# if rev-item in prices:
```

```
new-price = float(input(f"enter new price for {rev-item}:"))
```

```
prices.update({rev-item: new-price}) # dict.update()
```

```
# find costliest item
```

```
costliest-item = None
```

```
max-price = 0
```

```
for item, price in prices.items():
```

```
    if price > max-price:
```

```
        max-price = price
```

```
        costliest-item = item
```

```
# Remove out-of-stock item
```

```
remove-item = input("enter an item to remove from price list (or press enter to skip):")
```

```
removed-price = None
```

```
if remove-item:
```

```
    removed-price = prices.pop(remove-item, None)
```

```
# Display results
```

```
# dict.pop
```

```
Print("\n available items:", list(prices.keys()))
```

```
Print("prices:", list(prices.keys())) # dict.keys()
```

```
Print if costliest item:
```

```
Print("costliest item:", costliest-item, "at",
```

```
if remove-item:
```

```
max-price)
```

```
Print(f"Removed {remove-item} price
```

```
(if existed): {removed-price})
```

Result: Thus, Various data types, list, Tuples and Dict python programming was used and verified successfully