PANIMALAR ENGINEERING COLLEGE, CHENNAI

23ES1206 PYTHON PROGRAMMING –
III UNIT SECOND HALF- QUESTION BANK ANSWERS
ACADEMIC YEAR – 2024-2025 / SEMESTER-II

Prepared By
DR.G.UMARANI SRIKANTH
Professor
Dept of CSE, PEC

===

PART A - 2-MARKS

6.Give the operations on tuple.

- **Indexing** – Access elements using tuple[index].
- **Slicing** – Extract sub-tuples using tuple[start:stop:step].
- **Concatenation** – Combine tuples using +.
- **Repetition** – Repeat elements using *.
- **Membership** – Check existence using in and not in.
- **Length** – Find the number of elements using len(tuple).
- **Iteration** – Loop through elements using for loop.
- **Count & Index** – Use tuple.count(value) and tuple.index(value).

---

7. How do you add or replace a value in a dictionary in Python?
Adding or Replacing a Value in a Dictionary in Python
To Add a New Key-Value Pair:

```
my_dict = {"a": 1, "b": 2}
my_dict["c"] = 3  # Adding new key-value pair
```

To Replace an Existing Value:

```
my_dict["b"] = 5  # Replacing value of key 'b'
```

---

8. How to convert a list in to tuple and tuple into a list.
Convert a List to a Tuple:

```
my_list = [1, 2, 3, 4]
my_tuple = tuple(my_list)
print(my_tuple)  # Output: (1, 2, 3, 4)
```

Convert a Tuple to a List:

```
my_tuple = (5, 6, 7, 8)
my_list = list(my_tuple)
print(my_list)  # Output: [5, 6, 7, 8]
```

---

9. Write a python program to find maximum and minimum n elements in tuple.

```
def find_max_min_n(tup, n):
    sorted_tup = sorted(tup)
    return sorted_tup[-n:], sorted_tup[:n]
# Example Usage
tup = (10, 5, 8, 3, 15, 2, 20, 7)
n = 2
max_n, min_n = find_max_min_n(tup, n)
print("Max N elements:", max_n)
print("Min N elements:", min_n)
```

<u>output</u>

```
Max N elements: [15, 20]
Min N elements: [2, 3]
```

10. Write about packing and unpacking in a tuple with example.

<u>Packing: Assigning multiple values to a tuple.</u>

```
my_tuple = 10, "Python", 3.14            # Packing
print(my_tuple)  # Output: (10, 'Python', 3.14)
```

<u>Unpacking: Extracting values from a tuple into variables.</u>

```
a, b, c = my_tuple  # Unpacking
print(a, b, c)  # Output: 10 Python 3.14
```

✅ *Packing groups values, while unpacking retrieves them separately.* 🚀

<u>PART-B (13 MARK QUESTIONS)-</u>

1. Write about tuple as return value with example program.

In Python, a function can return multiple values using a tuple. This is a powerful feature that allows bundling different return values into a single compound object.

<u>Why Tuples for Return Values?</u>

- Python does not support returning multiple values directly like some other languages.
- Tuples allow grouping of multiple values into a single return object.
- Tuples are immutable, making them safe to use as return types.
- tuple can be returned from a function when multiple values need to be returned. Since tuples are immutable, they are a safe way to return multiple values without modification.

<u>Logic</u>

 A function calculate(a, b) is defined to perform **three operations**:

 Inside the function:

- sum_ = a + b → computes the sum.
- diff = a - b → computes the difference.
- product = a * b → computes the product.
- These three values are **returned as a tuple**: (sum_, diff, product).

 The function call calculate(10, 5) returns (15, 5, 50) and stores it in result.

 The values are **accessed by indexing** the tuple:

- result[0] → Sum
- result[1] → Difference
- result[2] → Product

 The program **prints** the full tuple and each value separately.

program

```
def calculate(a, b):
   sum_ = a + b
   diff = a - b
   product = a * b
   return sum_, diff, product  # Returns a tuple
# Function call
result = calculate(10, 5)
# Output the results
print("Returned tuple:", result)
print("Sum:", result[0])
print("Difference:", result[1])
print("Product:", result[2])
```

output

```
Returned tuple: (15, 5, 50)
Sum: 15
Difference: 5
Product: 50
```

---

7. Explain about linear search and write a python program to perform linear search on a list.

Linear search or sequential search is the simplest searching algorithm used to find an element in a list or array. It checks each element in sequence until the target element is found or the end of the list is reached.

Working Principle

- Start from the first element of the array.
- Compare each element with the key (element to be searched).
- If a match is found, return the index of that element.
- If the end of the array is reached and no match is found, return -1.

Logic

```
set i = 0
repeat following steps while i < n
                if arr[i] == key, return i
                increment i by 1
Return -1 (element not found)
```

Example (with Array)

Array: [10, 25, 30, 45, 50]

Key to search: 30

- Step 1: 10 ≠ 30
- Step 2: 25 ≠ 30
- Step 3: 30 = 30 → Element found at index 2

Python Code

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Return the index if found
    return -1  # Return -1 if not found
#enter the total elements
n=int(input("enter the total elements "))
#set list called lst as empty
lst=[]
#get an element and append it with lst
for i in range(n):
    element=int(input("enter the element "))
    lst.append(element)
#get target value to be searched
target=int(input("enter the target element "))
index=linear_search(lst,target)
if (index!= -1) :
    print(index)
else:
    print("element not found")
```

output

```
enter the total elements 5
enter the element 66
enter the element 77
enter the element 99
enter the element 44
enter the element 55
enter the target element 44
3
enter the total elements 5
enter the element 99
enter the element 88
enter the element 77
enter the element 66
enter the element 55
enter the target element 100
element not found
```

8. **(i)Write a python program to create a histogram from a given list of integers.**

Histogram

A **histogram** is a graphical representation of the **distribution of numerical data** using **bars**. Each bar represents the **frequency** (count) of data points that fall within a specific range or interval (called a **bin**).

✅ **Example:**

If you have test scores of students, a histogram shows how many students scored within ranges like 0–10, 11–20, etc.

✅ **Key Features:**

- X-axis: Intervals (bins) of values
- Y-axis: Frequency (count) of values in each bin
- Useful for identifying data distribution (normal, skewed, etc.)

Python code

```python
#create histogram
def histogram(data):
   for i in data:
      print("*" *i )

size=int(input("enter the size "))
lst=[]
for i in range(size):
  element=int(input("enter the element "))
  lst.append(element)
histogram(lst)
```

output

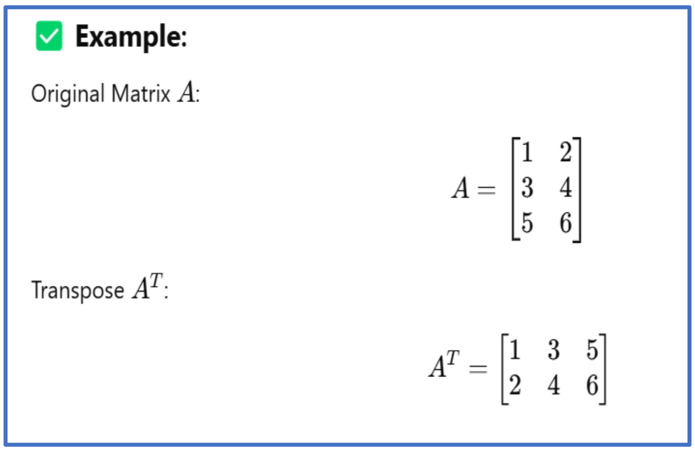
```
enter the size 5
enter the element 3
enter the element 2
enter the element 7
enter the element 5
enter the element 9
***
**
*******
*****
*********
```

(ii)Write a python program for transpose of a given matrix.

The **transpose** of a matrix is obtained by **interchanging its rows and columns**.

✅ **Definition:**

If **A** is an m×n matrix, then its **transpose** A^T is an n×m matrix where the element at position (i, j) in **A** becomes the element at (j, i) in A^T.



✅ **Example:**

Original Matrix A:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Transpose A^T:

$$A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Python code

```
M = 3 #no of rows
N = 4  #no of columns
A = [ [1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]]
#initialise Matrix B with zeros for N rows and  M cols using list comprehension
B = [[0 for x in range(M)] for y in range(N)]  #here N is the rows and M cols
#find transpose of A matrix
for i in range(M):
    for j in range(N):
          B[j][i] = A[i][j]
# print the B matrix
print("Result matrix is")
for i in range(N):
    for j in range(M):
         print(B[i][j], " ", end='')
    print()
```

output

```
Result matrix is
1  2  3
1  2  3
1  2  3
1  2  3
```

----

9. (i)Write a program to interchange the first and last element in a list.

Logic

1.Define a function swapfirstlast(mylist).

2.Inside the function:

- Check if the length of mylist is greater than 1.
- If yes:
    - Swap the first element mylist[0] with the last element mylist[-1].

3.Return the modified list.

4.In the main program:

- Define a list numbers with some elements.
- Call the function swapfirstlast(numbers) and store the result in swapped list.

5.Print "List after swapping:" followed by swapped list.

Python code

```
def swapfirstlast(mylist):
    if len(mylist) > 1:  # Ensure the list has at least two elements
        mylist[0], mylist[-1] = mylist[-1], mylist[0]  # Swap first and last element
    return mylist
numbers = [10, 20, 30, 40, 50]
swapped_list = swapfirstlast(numbers)
print("List after swapping:", swapped_list)
```

output

```
List after swapping: [50, 20, 30, 40, 10]
```

(ii)Write a python program to sort a dictionary by key or value.

Logic : sort by key

1. Create a dictionary d with integer keys and string values.
2. Extract all the keys from the dictionary using d.keys().
3. Use the sorted() function to sort the keys in ascending order.
4. Store the result in a variable sorted_dict .
5. Print the sorted list of keys.

Python code

```
#sort dict by key
d = {3: "three", 1: "one", 2: "two", 5: "five", 4: "four"}
# Sorting and printing
sorted_dict = sorted(d.keys())
print(sorted_dict)
```

output

```
[1, 2, 3, 4, 5]
```

<u>#sort dict by value</u>

<u>Logic</u>

1. Create a dictionary **d** with keys and string values.
2. Access all key-value pairs using d.items().
3. Use sorted() function with a lambda function as the key argument to sort by value.
4. lambda item: item[1] extracts the value from each (key, value) pair for comparison.
5. Convert the sorted list of tuples back into a dictionary using dict().
6. Store the result in sorted_dict.

<u>Python code</u>

```
#sort dict by value
d = {3: "three", 1: "one", 2: "two", 5: "five", 4: "four"}
sorted_dict = dict(sorted(d.items(), key=lambda item: item[1]))
print(sorted_dict)
```

<u>output #dictionary sorted using values</u>

```
{5: 'five', 4: 'four', 1: 'one', 3: 'three', 2: 'two'}
```

----

10. i) How can you determine the size of a set in Python?

<u>Logic</u>

1. initialize an empty set my_set.
2. repeat the following steps until user enters 0:
   a. prompt the user to enter an element.
   b. Convert the input to integer and store it in variable ele.
   c. if ele == 0, then exit the loop.
   d. else, add ele to the set my_set using the add() method.
3. Display the elements of the set my_set.
4. Compute the size of the set using len(my_set).

<u>program</u>

```
# Defining a set
my_set=set()
while True:
      ele= int(input("enter the element  "))
      if ele==0:
          break
      my_set.add(ele)
print(my_set)
print("The size of the set is:  ", len(my_set))
```

<u>output</u>

```
enter the element  23
enter the element  56
enter the element  44
enter the element  88
enter the element  99
enter the element  0
{99, 44, 23, 56, 88}
The size of the set is:   5
```

ii) Discuss about calendar program supported by python.

Calendar Module in Python

Python provides a built-in **calendar** module to work with dates and calendars. It supports various functions to display calendars, check leap years, find weekdays, and more.

Key Features of the calendar Module

1. **Display a Calendar** – Print a monthly or yearly calendar.
2. **Find Leap Years** – Check if a year is a leap year.
3. **Get Weekday of a Date** – Determine the weekday for a specific date.
4. **Leap Days Between Years-** leapdays(y1, y2) function returns the number of leap years in the range from year y1 to y2 (excluding y2).
5. **First Weekday Setting** – Modify the starting day of the week.
6. **Iterate Over Months and Days** – Retrieve month/week-based data.

Example Programs

1. Display a Calendar for a Given Year

example

```
import calendar
year = 2025
print(calendar.calendar(year))
```

📝 *Displays the full-year calendar for 2025.*

---

2. Display a Specific Month

example

```
import calendar
print(calendar.month(2024, 3))  # March 2024
```

📝 *Displays only March 2024.*

---

3. Check if a Year is a Leap Year

example

```
import calendar
print(calendar.isleap(2024))  # Output: True (Leap Year)
print(calendar.isleap(2023))  # Output: False
```

4.Leap Days Between Years: The leapdays(y1, y2) function returns the number of leap years in the range from year y1 to y2 (excluding y2).

example

```
import calendar
print(calendar.leapdays(2000, 2025))  # Output: 7
```

5. Find the Weekday of a Given Date
example
```
import calendar
day_of_week = calendar.weekday(2024, 3, 30)  # (Year, Month, Day)
print(day_of_week)  # 5 (Saturday)
```

---

6. Get the First Weekday and Number of Days in a Month
example
```
import calendar
first_day, num_days = calendar.monthrange(2024, 3)
print("First day:", first_day, "Number of days:", num_days)
```
output
```
First day: 4 Number of days: 31
```

PART-C

2(a). How can lists, tuples, and dictionaries handle mixed data types? Provide examples for each.

In Python, **lists**, **tuples**, and **dictionaries** are versatile data structures that can store elements of **mixed data types**. This flexibility allows you to combine integers, strings, floats, and even other complex objects within a single collection. Here's how each structure accommodates mixed data types:

Lists

- **Definition**: An ordered, mutable collection that allows duplicate elements.
- **Mixed Data Types**: Lists can contain elements of varying data types, including other lists or complex objects.

Example:
```
my_list = [42, "hello", 3.14, [1, 2, 3], {"key": "value"}]
print(my_list)
```
Output:
```
 [42, 'hello', 3.14, [1, 2, 3], {'key': 'value'}]
```
In this example, my_list contains an integer, a string, a float, another list, and a dictionary. This demonstrates the list's ability to hold heterogeneous data types.

Tuples

- **Definition**: An ordered, immutable collection that allows duplicate elements.
- **Mixed Data Types**: Tuples can also store elements of different data types.

Example:
```
my_tuple = (1, "apple", 3.5, [10, 20], {"key": "value"})
print(my_tuple)
```
Output:
```
 (1, 'apple', 3.5, [10, 20], {'key': 'value'})
```

Here, my_tuple includes an integer, a string, a float, a list, and a dictionary, showcasing that tuples can hold mixed data types.

Dictionaries

- **Definition**: An unordered collection of key-value pairs where keys are unique
- **Mixed Data Types**: Keys must be of an immutable data type (like strings, numbers, or tuples), but values can be of any data type, allowing for a mix of different types.

Example:

```
my_dict = {
    "name": "Alice",
    "age": 30,
    "is_member": True,
    "scores": [85, 90, 78],
    "profile": {"height": 165, "weight": 68}
}
print(my_dict)
```

Output:

```
{
    'name': 'Alice',
    'age': 30,
    'is_member': True,
    'scores': [85, 90, 78],
    'profile': {'height': 165, 'weight': 68}
}
```

In this dictionary, keys are strings, and values include a string, an integer, a boolean, a list, and another dictionary, illustrating the capacity to handle mixed data types.

Key Points:

- **Lists**: Ordered and mutable; can contain duplicate and mixed data types.
- **Tuples**: Ordered and immutable; can contain duplicate and mixed data types.
- **Dictionaries**: Unordered; keys must be immutable and unique, but values can be of any data type, allowing for mixed types.

This flexibility in Python's data structures enables the creation of complex and heterogeneous collections, facilitating diverse data management tasks.

---

b. Discuss about tuple assignment and write a python program for swapping of two numbers using tuple assignment.

Tuple Assignment:

-Tuple assignment allows assigning values to multiple variables simultaneously using tuples.

-It enables unpacking of a tuple into individual variables in a single line.

-This is useful in swapping values, returning multiple values from a function, etc.

Syntax:

a, b = (10, 20)   # Tuple unpacking

The values 10 and 20 are assigned to a and b respectively.

Swapping Two Numbers Using Tuple Assignment:

Instead of using a temporary variable, you can swap two variables directly:

```
a, b = b, a
```

This uses tuple packing on the right and unpacking on the left.

✅ **Python Program: Swapping Using Tuple Assignment**

```
# Input two numbers
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
print("Before swapping:")
print("a =", a)
print("b =", b)
# Swapping using tuple assignment
a, b = b, a
print("After swapping:")
print("a =", a)
print("b =", b)
```

Output

```
Enter first number: 5
Enter second number: 10
Before swapping:
a = 5
b = 10
After swapping:
a = 10
b = 5
```
