CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test - II

Sub:	Introduction to Python, Data and Control Systems							Code:	22MBABA303
Date:	16-04-2025	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MBA
SET- II									

1.a. Explain how arrays are different from lists in Python

Feature	Python List	Array (from `array` module or `NumPy`)
Data Type	Can hold **different data types** (e.g., int, float, str)	Can hold **only one data type** (e.g., all integers or all floats)
Flexibility	Very flexible and dynamic	Less flexible, but more efficient for numerical data
Performance	Slower for large numerical data	Faster and more memory-efficient for large numerical operations
Functionality	General-purpose, not optimized for math	Optimized for **numerical and scientific computing**
Built-in Support	Native to Python (`[]`)	Requires importing `array` or `numpy`
Example	`[1, 'hello', 3.14]`	`array.array('i', [1, 2, 3])` or `np.array([1, 2, 3])`

1.b. Assume a string of first 10 letters of English alphabet, using the slice operation perform the following:

i) Print first 3 letters from the list

```
alphabet = "abcdefghij"
first_three = alphabet[:3]
print("First 3 letters:", first_three)
```

ii) Print from 5th letter to end of the list

```
alphabet = "abcdefghij"
from_fifth_onwards = alphabet[4:]
print("From 5th letter to end:", from_fifth_onwards)
```

iii) Print any three letters from the middle

```
alphabet = "abcdefghij"
middle_three = alphabet[3:6]
print("Three letters from the middle:", middle_three)
```

1.c. Analyze different types of Collections available in Python

Python provides several built-in **collection data types** to store and organize data efficiently. Each type serves a different purpose based on the structure, mutability, and access pattern.

List

- **Ordered**, **mutable**, allows **duplicate values**.
- Stores items in a sequence.
- Elements can be accessed via index.

my_list = [1, 2, 3, 4]

Tuple

- **Ordered**, **immutable**, allows **duplicates**.
- Faster than lists and can be used as dictionary keys if they contain only immutable data.

my_tuple = (1, 2, 3)

Set

- **Unordered**, **mutable**, **no duplicate values**.
- Ideal for membership testing and removing duplicates.

my_set = {1, 2, 3, 3}

Dictionary

- **Unordered (ordered since Python 3.7)**, **mutable**, **key-value pairs**.
- Keys must be unique and immutable; values can be of any type.

my_dict = {"State": "Karnataka", "City": “Bangalore”}

2.a. Differentiate parameters and arguments in a function

Concept	Parameter	Argument
Definition	A **variable** in the function **definition**	A **value** passed to the function **when it is called**
When used	While **defining** a function	While **calling** a function
Purpose	Acts as a **placeholder** for input	Provides **actual data** to the function
Location	Inside the parentheses of a function definition	Inside the parentheses during function call

2.b. Explain the methods of Dictionary, fromkeys(), items() and pop() using an example

Method	**Purpose**
fromkeys()	Creates a new dictionary with given keys and a default value
items()	Returns a list-like view of key-value pairs
pop()	Removes and returns the value of the specified key

1. fromkeys()

Creates a new dictionary with specified **keys** and a **common default value**.

Example:

```
keys = ['a', 'b', 'c']

new_dict = dict.fromkeys(keys, 0)

print(new_dict)
```

Output:

{'a': 0, 'b': 0, 'c': 0}

2. items()

Returns a **view object** that displays a list of the dictionary's **(key, value)** pairs.

Example:

```
my_dict = {'name': 'KNK', 'city': 'BLR'}

print(my_dict.items())
```

Output:

dict_items([('name', 'KNK'), ('age', 'BLR')])

You can loop through items() like this:

```
    for key, value in my_dict.items():

    print(key, "->", value)
```

3. pop()

Removes the item with the **specified key** and returns its value. If the key is not found, it throws a KeyError unless a **default value** is provided.

Example:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
value = my_dict.pop('b')
print("Popped value:", value)
print("Updated dict:", my_dict)
```

Output:

Popped value: 2

Updated dict: {'a': 1, 'c': 3}

2.c. Evaluate the effectiveness of two different functions for adding two numbers, one with local variables, one with global variables

As local variables

```
def add_numbers():
    a = 10  # local variable
    b = 20  # local variable
    result = a + b
    print("Sum:", result)
add_numbers()
```

As global Variables

```
# Global variables
x = 15
y = 25
def add_numbers():
    result = x + y  # using global variables
    print("Sum:", result)
add_numbers()
```

3.a. Use datetime module to build a program that gets the current date and prints that day of the week

```
import datetime
# Get current date
current_date = datetime.date.today()
# Get the day of the week (e.g., Monday, Tuesday, etc.)
day_of_week = current_date.strftime("%A")
print("Today's date is:", current_date)
print("Day of the week is:", day_of_week)
```

3.b. Analyze the significance of exception handling on par with the other types of errors

Aspect	Errors	Exceptions
Definition	Issues in code that are usually **syntax-related** or beyond control	Issues during **runtime** that can be **handled**
When Occurs	**Before execution** (compile-time) or serious runtime faults	**During execution** (runtime)
Can be handled?	❌ No, they typically crash the program	✅ Yes, using `try` - `except` blocks
Examples	SyntaxError, IndentationError, MemoryError	ZeroDivisionError, ValueError, FileNotFoundError

3c. Explain different Python packages along with their importance

Package	Purpose	Importance
`NumPy`	Numerical operations	Fast and efficient array operations
`Pandas`	Data analysis	Easy handling of tabular data
`Matplotlib`	Visualization	Build charts and graphs
`Scikit-learn`	Machine learning	Simple ML model creation
`TensorFlow`	Deep learning	Build complex neural networks
`Flask/Django`	Web development	Build web apps and APIs
`Requests`	HTTP requests	API communication
`BeautifulSoup`	Web scraping	Extract data from web pages
`Datetime`	Date/time operations	Useful in time-based data
`OS/Sys`	System operations	File handling, system interaction

4. Develop a Python program for

- **Accepting two numbers and find the quotient**
- **Handle the zero-division error using exception handling.**

- Accepts two numbers from the user.
- Finds the quotient (division result).
- Handles zero-division errors using try-except.

```
# Program to find the quotient and handle ZeroDivisionError
try:
    # Accepting input from the user
    num1 = float(input("Enter the numerator: "))
    num2 = float(input("Enter the denominator: "))
    # Performing division
    quotient = num1 / num2
    print("Quotient is:", quotient)
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")
except ValueError:
    print("Error: Please enter valid numbers.")
```