MANIPAL
UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

Name	**Juli kumari**
Roll Number	**2214509725**
Course Name	**Python Programming**
Course Code	**DCA3104**
Semester	**5th semester**
Session	**2024**

<u>Set-I</u>

Question 1 = a) Discuss different datatypes used in Python Programming.

Answer = Python has a wide range of built-in data types to effectively handle various data types. These data types fall under the following general categories: Boolean, mapping, set, numerical, and sequence.

(a) Numeric type
- int – It used to store intergers. For example – 10,-5,6.
- float – It used to store decimal numbers. For example – 3.14,-0.5.

(b) Sequence type
- str – It holds text data, such as "Hello" and "Python". String cannot be changed.
- list – It holds a sorted group of objects that can be of many kinds and are changeable. For example – [1,"hello" 3.24].
- tuple – It is similar to lists but immutable. For example – (1,"Python",4,-23).

(c) Set type
- Set – A collection of unique items that are not in order. For example = {1,2,3}

(d) Mapping type
- dict – A collection of key – values pairs . For example – {"name":"Alice","age":25} .

(e) Boolean type
- Bool – It represents True or False.

(f) None type
- None type – It represents the absence of a value. For example – None.

Python does not require explicit type definitions for variables because it is dynamically typed. Because of its adaptability, Python may be used for a wide range of data manipulation tasks, making it both powerful and easy to use.

b) Explain the concept of operator precedence with example.

Answer – Operator precedence dictates the sequence in which operators are evaluated inside an expression. Python resolves complex expressions according to a certain hierarchy. Higher precedence operators are assessed before lower precedence operators.

Levels of Precedence: High to Low

1) Parenthesis () : When expressing the evaluation order explicitly, parenthesis has the highest precedence. For instance, result = (3 + 2) * 4 → result = 20
2) Exponentiation ** : For instance, result = 2 ** 3 ** 2 → result = 2 ** 9 = 512 (Right-to-left association)
3) Unary Operators +, -, ~ : For instance, result = -5 + 3 → result = -2
4) Operators for multiplication *, /, //, % : For instance, result = 10 / 2 * 3 → result = 15
5) The Additive Operators + and - : For instance, result = 10 - 2 + 3 → result = 11.
6) Comparison operators such as <, >, ==,!=, and others : For instance, result = 5 > 3 -> result = True
7) Operators of Logic and, or, not : And has a higher precedence than or, while o not has a higher precedence than and. For instance, result = not True or False and True → result = False

8) Assignment Operators =, +=, -= : The least important operators are assignment operators.

Operator Precedence Example

```
Result = 2*3-4/2+10

#Procedure

#1. First, multiply and divide – 2*3=6,4/2=2.

#2.Subtacting and adding – 10+6-2 = 14

#final score – 14.
```

Writing expressions that are efficient, error-free, and unambiguous requires an understanding of operator precedence. To ensure accurate evaluation, utilise brackets whenever possible.

Question 2 = a) Discuss the use of else statement with for and while loop. Example with an example.

Answer - Python's for and while loops can be utilised using the else statement. When the while loop's state changes to False or the loop has completed iterating through its sequence, the else block is run. The else block is skipped, though, if the loop is ended early using the break statement.

Use with for loop - A for loop's else block only runs if all of the loop's iterations are finished without a break.

For Example,

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

for num in numbers:

   if num == 0:  # Check for zero

      print("Zero found!")

      break

else:

   print("All numbers are non-zero.")

# Output: All numbers are non-zero.
```

Use with while loop - If there is no break to end the loop, the else block in a while loop runs when the loop condition changes to False.

For Example,

```
count = 3

while count > 0:

   print(f"Count is {count}")
```

```
    count -= 1
else:
    print("Loop completed.")
# Output:
# Count is 2
# Count is 1
# Loop completed.
```

Take note - For post-loop operations that should only take place after the loop naturally ends the else block is helpful.

b) Explain the use of following string functions :- upper(), lower() , isdigit() , isalpha() , split() , join() with example.

Answer - Python comes with a number of built-in string functions for working with and querying strings. Here are a few popular techniques:

1) upper() - All of the string's characters are changed to uppercase.
   For instance,
   ```
   text = "hello world"
   result = text.upper()
   print(result) # Output: "HELLO WORLD"
   ```

2) lower() - Changes all of the string's characters to lowercase.
   For instance,
   ```
   text = "HELLO WORLD"
   result = text.lower()
   print(result) # Output: "hello world"
   ```

3) isdigit() - Determines whether the string is made up entirely of numeric characters. Returns True if every character is a number; False otherwise.
   For instance,
   ```
   text = "12345"
   (text.isdigit()) print # Output: "123a" is the true text.
   (text.isdigit()) print # Output: Untrue
   ```

4) isalpha() - Determines whether the string is made up entirely of alphabetic characters. If every character is alphabetic, the return is True; if not, it is False.
   For instance,
   ```
   text = "Hello"
   print(text.isalpha()) # Output: True
   text = "Hello123"
   print(text.isalpha()) # Output: False
   ```

5) split() - Uses a provided delimiter (whitespace is the default) to divide a string into a list of substrings.
   For instance,
   text = "Python is fun"
   words = text.split()
   print(words) # Output: ["Python", "is", "fun"]
   text = "a,b,c"
   parts = text.split(",")
   print(parts) # Output: ["a", "b", "c"]

6) join() - Uses a provided delimiter to unite elements of a list (or any iterable) into a single string.
   For Instance,
   words = ["Python", "is", "fun"]
   sentence = " ".join(words)
   print(sentence) # Output: "Python is fun"
   letters = ["a", "b", "c"]
   result = "-".join(letters)
   print(result) # Output: "a-b-c"

Question 3 = a) How to delete an element from list? Explain different cases with examples.
Answer - Python has multiple methods for removing items from a list. The precise requirement—such as deleting by value, by index, or eliminating several elements—determines the approach that is taken.

1) Using Del Statement - Making use of the del statement - Cuts a group of elements or removes a member by its index.
   As an illustration
   - Delete by Index
     numbers = [10, 20, 30, 40]
     del numbers[1]
     print(numbers) # Output: [10, 30, 40]
   - Delete a Range
     numbers = [10, 20, 30, 40]
     del numbers[1:3]
     print(numbers) # Output: [10, 40]

2) Using remove() method - Eliminates the value's initial occurrence.
   As an illustration
   fruits = ['apple', 'banana', 'cherry', 'banana']
   fruits.remove('banana')
   print(fruits) # Output: ['apple', 'cherry', 'banana']

3) Using pop() method - Returns the element at a given index after removing it; by default, this is the last element.
   As an illustration
   - Remove Last Element
     numbers = [10, 20, 30, 40]

```
last = numbers.pop()
print(numbers)  # Output: [10, 20, 30]
print(last)     # Output: 40
```

- Remove by Index

```
numbers = [10, 20, 30, 40]
removed = numbers.pop(1)
print(numbers)  # Output: [10, 30, 40]
print(removed)  # Output: 20
```

4) Using Slicing - Make a new list that does not include certain items.
   As an illustration

```
numbers = [10, 20, 30, 40]
numbers = numbers[:1] + numbers[2:]
print(numbers)  # Output: [10, 30, 40]
```

5) Clear Entire List - Clear() is used to eliminate every element.
   As an illustration

```
numbers = [10, 20, 30]
numbers.clear()
print(numbers)  # Output: []
```

b) Explain differences between list, tuple, set and dictionary.

Answer – Python comes with several built-in data structures for handling and organising data. The most often used ones are dictionary, set, tuple, and list. Because of its distinct qualities, each is appropriate for particular use scenarios. This is a thorough comparison:

1) List
   - Definition - A list is a collection that can hold a variety of elements and is ordered, changeable, and indexed.
   - Essential Features
     - Ordered - Keeps the elements in their proper arrangement.
     - Mutable - It is possible to add, modify, or remove elements.
     - Duplicates - Accepts duplicate values.
     - Indexing -Elements are accessible based on their location.
   - Syntax - my_list = [1, 2, 3, 4, 5]
   - Use Case - Ideal for keeping a well-organised collection of objects when regular updates are required, and duplicates are permitted.
2) Tuple
   - Definition - An ordered, unchangeable collection that can hold a variety of elements is called a tuple.
   - Essential Features
     - Ordered - keeps the elements in their proper arrangement.
     - Immutable - Elements cannot be altered or eliminated after they have been defined.
     - Duplicates - Accepts duplicate values.

        - Indexing - Allows for access items through indexing.
    - Syntax - my_tuple = (1, 2, 3, 4, 5)
    - Use Case - Perfect for fixed data sets where element integrity needs to be maintained.

3) Set
    - Definition - An unordered, changeable collection of distinct elements is called a set.
    - Essential Features
        - Unordered - Does not keep the elements in their proper arrangement.
        - Mutable - Individual elements cannot be changed, however elements can be added or removed.
        - No Duplicates - Guarantees that every component is distinct.
        - No indexing - Does not allow for slicing or indexing.
    - Syntax -my_set = {1, 2, 3, 4, 5}
    - Use Case - Ideal for guaranteeing that data stays unique or for operations like union, intersection, and difference.

4) Dictionary
    - Definition - An unordered, changeable collection of key-value pairs is called a dictionary.
    - Essential Features
        - Key-Value Pairs - Uses value pairs as the key to store data.
        - Keys -Must be distinct and unchangeable (such as tuples, strings, or numbers).
        - Values - May have duplicates and be changeable.
        - No indexing - Use keys, not positions, to access items.
    - Syntax - my_dict = {"name": "Alice", "age": 25, "city": "New York"}
    - Use Case - Perfect for situations where identifiers (keys) are required to access or edit data.

<u>Set-II</u>

Question 4 = a) What is lambda function? Why is it used?

Answer - Python lambda functions are anonymous functions, meaning they don't have names. Simple actions can be carried out in a clear and compact manner with lambda functions.

A lambda function has the following syntax-
lambda arguments: a expression
arguments - The function takes in this list of parameters.
expression - The computation or action that the function carries out and outputs.

Example of a Lambda Function

square = lambda x: x * x

print(square(5)) # Output: 25

The Use of Lambda Functions

1) Conciseness - Simple functions can be defined in a single line of code thanks to lambda functions' conciseness. This is especially helpful for small businesses.
2) Anonymous Role - When you need a function temporarily, such as when sorting, filtering, or applying a function to components of an iterable, lambda functions are helpful because they don't require naming.
3) Programming that is functional - Functional programming paradigms, in which functions are supplied as arguments to higher-order functions like map(), filter(), and reduce(), make extensive use of lambda functions.

Lambda example in map()

```
numbers = [1, 2, 3, 4]
squared = map(lambda x: x ** 2, numbers)
print(list(squared)) # Output: [1, 4, 9, 16]
```

b) Explain differences between remove(), discard( ) and pop( ) method for deleting elements from set.

Answer - Sets are collections of distinct, unordered entities in Python. There are various ways to eliminate items from a set, but they behave differently when it comes to handling missing elements and element removal.

1) The remove() method

- To eliminate a particular element from the set, use the remove() method.
- A Key Error is raised if the element cannot be located.
- It doesn't return a value; instead, it changes the original set.

  Example
  ```
  fruits = {'apple', 'banana', 'cherry'}
  fruits.remove('banana')
  print(fruits)  # Output: {'apple', 'cherry'}
  ```
  If an element that is not part of the set is attempted to be removed:
  ```
  fruits.remove('mango')  # Raises KeyError
  ```

2) discard() method

- The discard() method, like remove(), eliminates a particular element from the set.
- Discard() does not raise an error if the element cannot be located, which is the main distinction. It just does nothing.
- It does not return a value, just like delete(), but it does alter the original set.

  Example
  ```
  fruits = {'apple', 'banana', 'cherry'}
  fruits.discard('banana')
  print(fruits)  # Output: {'apple', 'cherry'}
  ```
  If you try to discard an element that is not in the set, no error occurs:
  ```
  fruits.discard('mango')  # No error, set remains unchanged
  print(fruits)  # Output: {'apple', 'cherry'}
  ```

3) pop() method

- An arbitrary element from the set is removed and returned using the pop() method.
- Because sets are not ordered, the element that is eliminated is not always the same and may vary from one instance to the next.
- Pop() raises a KeyError if the set is empty.

  Example
  ```
  fruits = {'apple', 'banana', 'cherry'}
  removed_item = fruits.pop()
  print(fruits)  # Output: A set with one element removed (arbitrary)
  print(removed_item)  # Output: The removed element (e.g., 'banana')
  ```
  If the set is empty:
  ```
  empty_set = set()
  empty_set.pop()  # Raises KeyError
  ```

How to Use Each

- To make sure an element is in the set and to generate an error if it is absent, use delete().
- To safely remove an element without caring about whether it exists or not, use discard().
- If you wish to delete and recover any element without caring which one is removed, use pop().

Question 5 = What is exception handling? Write a Python program which would throw exception if the value entered by user is less than zero or zero otherwise print it.

Answer – Python's exception handling mechanism enables programmers to manage problems (also known as exceptions) that could arise while a program is running. Using the try, except, else, and finally blocks, Python offers a method for handling mistakes gracefully rather to allowing the program to crash. By identifying and handling faults in a controlled manner, the primary objective is to guarantee that a program continues to run even in the event of an error.

Exception handling components include

1) The try block - Includes the code that has the potential to raise an exception. The software can try running some code that can cause an error thanks to the try block.
2) Except block - Includes the code to deal with the exception in the event that the try block raises one. What to do in the event of an error is specified by the unless block.
3) Else block - Carries out if the try block did not generate an exception.
4) Finally block - Regardless of whether an exception happens or not, this block gets executed. For housekeeping tasks (such deleting files or releasing resources), it is helpful.

Program Example

Utilising Exception Handling to Manage User Input
The Python program that follows shows how to respond when a user enters zero or a negative value. An exception is raised if the entered value is less than or equal to zero. If not, the user-inputted value is printed.

```
def get_user_input():

    try:

        # Taking input from the user

        value = float(input("Enter a number: "))

        # Check if the value is less than or equal to zero

        if value <= 0:

            # Raise an exception if the number is less than or equal to zero

            raise ValueError("The value cannot be less than or equal to zero.")

    except ValueError as e:

        # Handle the exception if the value is negative or zero

        print(f"Error: {e}")

    else:

        # This block executes if no exception occurs

        print(f"You entered: {value}")

    finally:

        # This block always executes

        print("Execution completed.")

# Calling the function to run the program

get_user_input()
```

Justification

1. Information Gathering - To gather user input, utilise the input() function. Float() is used to transform the input into a float. This is because we also want to handle decimal values that the user may enter.
2. Making an Exception - The program uses the raise keyword to raise a ValueError with the relevant error message if the entered value is less than or equal to zero. This halts additional try block execution.
3. Managing Exceptions - Unless block outputs a message stating that the input was invalid (either zero or negative) after catching the raised ValueError.
4. Else Block - The else block is run, and the entered value is written if there is no exception (that is, if the input is a positive number).

5. Finally Block - Whether or whether there is an exception, the finally block always executes. Cleaning is one of its frequent uses. Here, it merely prints the words "Execution completed."

Question 6 = a) Explain the use of pandas, NumPy and matplotlib libraries in detail.
Answer = Three of the most widely used libraries in Python's extensive ecosystem for data analysis and visualisation are Pandas, NumPy, and Matplotlib. Each has a distinct function in effectively managing and evaluating data.

1. Pandas
    - Goal - Mostly used for structured data operations including data wrangling, cleaning, and preparation, Pandas is a robust data manipulation library.
    - Important attributes
        - Data Structures - There are two primary data structures offered by Panda
            - For one-dimensional data, use a series.
            - DataFrame for tabular, two-dimensional data, similar to SQL or Excel tables.
        - Data Cleaning- Easily deal with discrepancies, duplication, and missing data.
        - Data Conversion - Execute tasks like as pivoting, reshaping, joining, and combining datasets.
        - Analysis of Data - Facilitates statistical calculations, grouping, and filtering.
    - Use Case - Pandas is perfect for preparing database datasets, time-series data, and CSV files for analysis.

2. NumPy
    - Goal -Python or NumPy, is a library for scientific computing and numerical calculations.
    - Important attributes
        - Array Management -Gives access to the ndarray object, which is significantly quicker and uses less memory than Python lists.
        - Operations in Mathematics - It provides resources for Fourier transformations, random number generation, linear algebra, and other topics.
        - Performance - Because NumPy is based on C, it does numerical calculations much more quickly than Python loops.
        - Integration - frequently serves as the backend for other libraries, such as Scikit-learn and Pandas.
    - Use Case - Helpful for carrying out mathematical calculations on huge datasets or matrices, such determining the standard deviation, mean, median, and matrix operations.

3. Matplotlib

- Goal - A data visualisation library called Matplotlib may be used to create animated, interactive, and static charts.
- Important attributes
    - Simple Plots -Make bar charts, pie charts, histograms, scatter plots, and line plots.
    - Personalisation - Offers a wide range of plot customisation choices, including axes, labels, legends, and styles.
    - Integration - Works well with NumPy and Pandas for direct data visualisation.
- Use Case - Frequently employed to visually represent distributions, trends, or correlations within a dataset.

b) What are CRUD operations?

Answer - The four fundamental actions carried out on databases or data storage systems are represented by the acronym CRUD, which stands for Create, Read, Update, and Delete. Effective data management and manipulation depend on these processes.

1. Create
    - Goal - Update a database or data structure with new information.
    - An example in SQL - INSERT INTO students (id, name, age) VALUES (1, 'John', 20);
    - Additional Use Cases - Creating data in Python entails adding items to a NumPy array or inserting rows in a Pandas DataFrame.
2. Read
    - Goal - Get information out of a data structure or database.
    - An example in SQL - SELECT * FROM students WHERE age > 18;
    - Additional Use Cases - Accessing components in a NumPy array or reading data from a CSV file into a Pandas DataFrame.
3. Update
    - Goal -Make changes to data that already exists in a database or data structure.
    - An example in SQL - UPDATE students SET age = 21 WHERE id = 1;
    - Additional Use Cases - Updating data in Pandas may require changing values in particular rows or columns.
4. Delete
    - Goal -Take information out of a data structure or database.
    - An example in SQL - DELETE FROM students WHERE age < 18;
    - Additional Use Cases - Removing items from a NumPy array or rows or columns from a Pandas DataFrame.