Assignment

1. **What are the types of Applications?**

**Answer :-**

An application, often called an app, is software designed to perform specific tasks for users. It can run on devices like computers, smartphones, and tablets. Examples include web browsers, games, and word processors.

**Types of Applications:**

1. **Web Applications :-** Run on web browsers and are accessed through URLs.

**Examples:** Google Docs, Facebook, Amazon.

1. **Mobile Applications:** Designed specifically for mobile devices.

**Types:** Native (built for specific platforms like iOS or Android), Hybrid (use web technologies but run inside a native container), and Web (accessed through mobile web browsers).

**Examples:** Instagram (native), Uber (hybrid), Google (web).

1. **Desktop Applications:** Installed on personal computers and run locally.

**Examples:** Microsoft Office, Adobe Photoshop, VLC Media Player.

1. **Enterprise Applications:** Large-scale software systems used to manage and integrate business operations.

**Examples:** SAP, Oracle ERP, Salesforce.

1. **Cloud Applications:** Applications that run on cloud infrastructure, allowing access from any internet-connected device.

**Examples:** Google Drive, Microsoft Azure, Dropbox.

1. **Embedded Applications:** Software designed to operate within another device or system.

**Examples:** Software in smart appliances, automotive control systems, medical devices.

1. **System Software:** Designed to provide core functions such as operating systems, disk management, utilities, hardware management, and other operational necessities.

**Examples:** Windows OS, Linux, macOS.

1. **Utility Software:** Provides system maintenance and management functions.

**Examples:** Antivirus programs, backup software, disk cleaners.

1. **Database Applications:** Software used to manage and manipulate databases.

**Examples:** MySQL, Oracle Database, Microsoft SQL Server.

1. **Multimedia Applications:** Used for creating, editing, and playing multimedia content.

**Examples:** Adobe Premiere Pro, VLC Media Player, Spotify.

1. **Educational Applications:** Designed for educational purposes to facilitate learning and teaching.

**Examples:** Duolingo, Khan Academy, Google Classroom.

1. **Game Applications:** Designed for entertainment, often with interactive elements.

**Examples:** Fortnite, Minecraft, Candy Crush Saga.

1. **Artificial Intelligence Applications:** Use AI technologies to perform tasks that usually require human intelligence.

**Examples:** Chatbots, recommendation systems, facial recognition software.

1. **Communication Applications:** Facilitate communication between users.

**Examples:** WhatsApp, Slack, Zoom.

1. **What is programming?**

**Answer :-**

Programming, also known as coding, is the process of creating a set of instructions that tell a computer how to perform a specific task. These instructions, called programs, are written in a language that the computer can understand and execute.

1. **What is Python?**

**Answer :-**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

1. **Write a Python program to check if a number is positive, negative or zero.**

**Answer :-**

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

if number > 0:

print("The number is positive.")

elif number < 0:

print("The number is negative.")

else:

print("The number is zero.")

**OUTPUT :-**

Enter a number: 10

The number is positive.

Enter a number: 0

The number is zero.

1. **Write a Python program to get the Factorial number of given numbers.**

**Answer :-**

number = int(input("Enter a number: "))

factorial = 1

if number < 0:

print("Factorial does not exist for negative numbers.")

elif number == 0:

print("The factorial of 0 is 1.")

else:

for i in range(1, number + 1):

factorial \*= i

print(f"The factorial of {number} is {factorial}.")

**OUTPUT:**

Enter a number: 5

The factorial of 5 is 120.

1. **Write a Python program to get the Fibonacci series of given range.**

**Answer :-**

range\_limit = int(input("Enter the range limit for Fibonacci series: "))

a = 0

b = 1

print("Fibonacci Series:")

print(a)

print(b)

while b < range\_limit:

a, b = b, a + b

if b < range\_limit:

print(b)

OUTPUT:-

Enter the range limit for Fibonacci series: 50

Fibonacci Series:

0

1

1

2

3

5

8

13

21

34

1. **How memory is managed in Python?**

**Answer :-**

Memory management is handled automatically by the Python interpreter through a private heap space. Here are some key aspects of memory management in Python:

1. **Automatic Memory Allocation and Deallocation:**
   * Python uses a built-in memory manager to manage memory allocation and deallocation automatically. This means developers do not need to manually allocate or deallocate memory as in lower-level languages like C or C++.
2. **Garbage Collection:**
   * Python utilizes a garbage collector to reclaim memory occupied by objects that are no longer referenced and thus are unreachable. The garbage collector periodically runs in the background to free up memory from unused objects.
3. **Reference Counting:**
   * Python uses a reference counting mechanism along with garbage collection. Each object in Python maintains a reference count, which tracks the number of references pointing to that object. When an object's reference count drops to zero, it indicates that the object is no longer in use, and the memory occupied by that object can be reclaimed.
4. **Memory Pool:**
   * Python manages memory allocation through a private heap containing various memory pools for different types of objects. This helps in efficient memory allocation and management.
5. **Memory Optimization:**
   * Python optimizes memory usage through techniques like memory sharing (for immutable objects like strings and tuples) and reuse of small integer objects (-5 to 256). This reduces the overhead of memory allocation for frequently used objects.
6. **What is the purpose continuing statement in python?**

**Answer :-**

The continue statement is used to skip the rest of the current iteration of a loop (typically a for or while loop) and continue to the next iteration. Its primary purpose is to control the flow of execution within loops.

Key purposes of the continue statement include:

1. **Skipping Iterations:** When encountering the continue statement, Python immediately stops executing the remaining code in the current loop iteration and proceeds directly to the next iteration.
2. **Selective Processing:** It allows selective processing of loop iterations based on certain conditions. For example, you might use it to skip processing certain elements in a list or performing specific actions on certain types of data.

Here's a simple example to illustrate the use of continue:

for num in range(1, 11):

if num % 2 == 0:

continue # Skip even numbers

print(num)

**OUTPUT:-**

1

3

5

7

9

1. **Write python program that swap two number with temp variable and without temp variable.**

**Answer :-**

**Swapping with a Temporary Variable**

a = float(input("Enter the first number (a): "))

b = float(input("Enter the second number (b): "))

temp = a

a = b

b = temp

print(f"After swapping, a = {a} and b = {b}")

**OUTPUT:-**

Enter the first number (a): 12

Enter the second number (b): 23

After swapping, a = 23.0 and b = 12.0

**Swapping without a Temporary Variable**

a = float(input("Enter the first number (a): "))

b = float(input("Enter the second number (b): "))

a = a + b

b = a - b

a = a - b

print(f"After swapping, a = {a} and b = {b}")

**OUTPUT:-**

Enter the first number (a): 12

Enter the second number (b): 13

After swapping, a = 13.0 and b = 12.0

**Explanation:**

This method uses arithmetic operations. By adding a and b, the sum is stored in a. Then, b is assigned the difference a - b, effectively swapping the values. Finally, a is assigned the difference a - b to complete the swap.

1. **Write a Python program to find whether a given number is even or odd, print out an appropriate message to the user.**

**Answer :-**

number = int(input("Enter a number: "))

if number % 2 == 0:

print(f"{number} is even.")

else:

print(f"{number} is odd.")

1. **Write a Python program to test whether a passed letter is a vowel or not.**

**Answer :-**

letter = input("Enter a letter: ").lower()

if letter in 'aeiou':

print(f"{letter} is a vowel.")

else:

print(f"{letter} is not a vowel.")

**OUTPUT:-**

Enter a letter: a

a is a vowel.

1. **Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.**

**Answer :-**

a = int(input("Enter the first integer: "))

b = int(input("Enter the second integer: "))

c = int(input("Enter the third integer: "))

if a == b or b == c or a == c:

sum = 0

else:

sum = a + b + c

print(f"The sum is: {sum}")

**OUTPUT:-**

Enter the first integer: 12

Enter the second integer: 13

Enter the third integer: 12

The sum is: 0

Enter the first integer: 12

Enter the second integer: 13

Enter the third integer: 14

The sum is: 39

1. **Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.**

**Answer :-**

a = int(input("Enter the first integer: "))

b = int(input("Enter the second integer: "))

if a == b or a + b == 5 or abs(a - b) == 5:

result = True

else:

result = False

print(result)

**OUTPUT:-**

Enter the first integer: 15

Enter the second integer: 15

True

1. **Write a python program to sum of the first n positive integers.**

**Answer :-**

n = int(input("Enter a positive integer: "))

sum\_n = 0

for i in range(1, n + 1):

sum\_n += i

print(f"The sum of the first {n} positive integers is: {sum\_n}")

**OUTPUT:-**

Enter a positive integer: 5

The sum of the first 5 positive integers is: 15

1. **Write a Python program to calculate the length of a string.**

**Answer :-**

user\_string = input("Enter a string: ")

string\_length = len(user\_string)

print(f"The length of the string is: {string\_length}")

**OUTPUT:-**

Enter a string: hey how

The length of the string is: 7

**Another Way :-**

user\_string = input("Enter a string: ")

string\_length = 0

for char in user\_string:

string\_length += 1

print(f"The length of the string is: {string\_length}")

**OUTPUT:-**

Enter a string: hey how

The length of the string is: 7

1. **Write a Python program to count the number of characters (character frequency) in a string**

**Answer :-**

user\_string = input("Enter a string: ")

char\_frequency = {}

for char in user\_string:

if char in char\_frequency:

char\_frequency[char] += 1

else:

char\_frequency[char] = 1

for char, frequency in char\_frequency.items():

print(f"'{char}': {frequency}")

OUTPUT:-

Enter a string: HEY HOW

'H': 2

'E': 1

'Y': 1

' ': 1

'O': 1

'W': 1

1. **What are negative indexes and why are they used?**

**Answer :-**

Negative indexes in Python are a way to index elements in sequences (such as lists, strings, and tuples) starting from the end of the sequence rather than the beginning. This feature is particularly useful for accessing elements from the end without needing to know the exact length of the sequence.

**How Negative Indexes Work:**

* The index -1 refers to the last element of the sequence.
* The index -2 refers to the second-to-last element, and so on.

For example, consider the list my\_list = [10, 20, 30, 40, 50]:

* my\_list[-1] returns 50 (the last element).
* my\_list[-2] returns 40 (the second-to-last element).

**Why Negative Indexes are Used:**

1. **Convenience**: Negative indexes provide a convenient way to access elements from the end of the sequence without calculating the offset from the start.
2. **Readability**: Using negative indexes can make the code more readable and expressive, especially when the intent is to access elements relative to the end of the sequence.
3. **Avoid Errors**: Negative indexes can help avoid errors related to calculating the length of the sequence and then accessing elements from the end.

**Example Usage:**

my\_list = [10, 20, 30, 40, 50]

# Accessing elements using negative indexes

print(my\_list[-1]) # Output: 50

print(my\_list[-2]) # Output: 40

print(my\_list[-3]) # Output: 30

**Example with Strings:**

my\_string = "Hello, World!"

# Accessing characters using negative indexes

print(my\_string[-1]) # Output: !

print(my\_string[-2]) # Output: d

print(my\_string[-3]) # Output: l

**Slicing with Negative Indexes:**

Negative indexes can also be used with slicing:

my\_list = [10, 20, 30, 40, 50]

# Slicing the last three elements

print(my\_list[-3:]) # Output: [30, 40, 50]

# Slicing from the second-to-last element to the end

print(my\_list[-2:]) # Output: [40, 50]

Negative indexes are a powerful feature in Python that enhance the flexibility and readability of code when dealing with sequences.

1. **Write a Python program to count occurrences of a substring in a string.**

**Answer :-**

main\_string = input("Enter the main string: ")

substring = input("Enter the substring to count: ")

occurrences = main\_string.count(substring)

print(f"The substring '{substring}' occurs {occurrences} times in the main string.")

**OUTPUT :-**

Enter the main string: hey how are you

Enter the substring to count: hey

The substring 'hey' occurs 1 time in the main string.

1. **Write a Python program to count the occurrences of each word in a given sentence**

**Answer :-**

sentence = input("Enter a sentence: ")

words = sentence.lower().split()

word\_frequency = {}

for word in words:

if word in word\_frequency:

word\_frequency[word] += 1

else:

word\_frequency[word] = 1

for word, frequency in word\_frequency.items():

print(f"'{word}': {frequency}")

**OUTPUT:-**

Enter a sentence: hey, How are you

'hey,': 1

'how': 1

'are': 1

'you': 1

1. **Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.**

**Answer :-**

string1 = input("Enter the first string: ")

string2 = input("Enter the second string: ")

new\_string1 = string2[:2] + string1[2:]

new\_string2 = string1[:2] + string2[2:]

result = new\_string1 + ' ' + new\_string2

print("Result:", result)

**OUTPUT:-**

Enter the first string: hey how

Enter the second string: hey fine

Result: hey how hey fine

1. **Write a Python program to add 'in' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead if the string length of the given string is less than 3, leave it unchanged.**

**Answer :-**

user\_string = input("Enter a string: ")

if len(user\_string) >= 3:

if user\_string.endswith('ing'):

user\_string += 'ly'

else:

user\_string += 'ing'

print("Result:", user\_string)

**OUTPUT :-**  
Enter a string: string

Result: stringly

1. **Write a Python function to reverses a string if its length is a multiple of 4.**

**Answer :-**

user\_string = input("Enter a string: ")

if len(user\_string) % 4 == 0:

reversed\_string = user\_string[::-1]

else:

reversed\_string = user\_string

print(reversed\_string)

**OUTPUT :-**

Enter a string: heyy

yyeh

1. **Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.**

**Answer :-**

user\_string = input("Enter a string: ")

if len(user\_string) < 2:

result = ''

else:

result = user\_string[:2] + user\_string[-2:]

print("Result:", result)

**OUTPUT:-**

Enter a string: hey how

Result: heow

1. **Write a Python function to insert a string in the middle of a string.**

**Answer :-**

def insert\_string\_middle(main\_string, insert\_string):

middle\_index = len(main\_string) // 2

new\_string = main\_string[:middle\_index] + insert\_string + main\_string[middle\_index:]

return new\_string

main\_string = input("Enter the main string: ")

insert\_string = input("Enter the string to be inserted: ")

result = insert\_string\_middle(main\_string, insert\_string)

print(result)

**OUTPUT :-**

Enter the main string: hey how are you?

Enter the string to be inserted: who are you?

hey how who are you?are you?

1. **What is List? How will you reverse a list?**

**Answer :-**

A **list** in Python is a built-in data type that is used to store multiple items in a single variable. Lists are ordered, mutable (changeable), and allow duplicate elements. They can contain items of different data types, including other lists.

**Characteristics of Lists:**

1. **Ordered**: The items have a defined order that will not change unless explicitly modified.
2. **Mutable**: The items can be changed, added, or removed after the list has been created.
3. **Allows Duplicates**: Lists can contain duplicate elements.
4. **Indexing**: Lists are indexed, and the first item has index [0], the second item has index [1], and so on.

**Example :-**

my\_list = [1, 2, 3, 'apple', 4.5, [5, 6]]

print(my\_list)

OUTPUT:-

[1, 2, 3, 'apple', 4.5, [5, 6]]

**How to Reverse a List in Python**

There are several ways to reverse a list in Python:

1. Using the reverse() Method:

This method modifies the list in place.

Example :-

my\_list = [1, 2, 3, 4, 5]

my\_list.reverse()

print(my\_list) # Output: [5, 4, 3, 2, 1]

1. Using Slicing:

This creates a new list that is the reverse of the original list.

Example :-

my\_list = [1, 2, 3, 4, 5]

reversed\_list = my\_list[::-1]

print(reversed\_list) # Output: [5, 4, 3, 2, 1]

1. Using the reversed() Function:

This returns an iterator that can be used to create a reversed list.

Example :-

my\_list = [1, 2, 3, 4, 5]

reversed\_list = list(reversed(my\_list))

print(reversed\_list) # Output: [5, 4, 3, 2, 1]

1. **How will you remove last object from a list?**

**Answer :-**

To remove the last object from a list in Python:-

1. **Using the pop() Method**

The pop() method removes and returns the last item in the list if no index is specified.

Example :-

my\_list = [1, 2, 3, 4, 5]

my\_list.pop()

print(my\_list) # Output: [1, 2, 3, 4]

1. **Using the del Statement**

The del statement can be used to delete the last item in the list by specifying its index.

Example :-

my\_list = [1, 2, 3, 4, 5]

del my\_list[-1]

print(my\_list) # Output: [1, 2, 3, 4]

1. **Using Slicing**

You can create a new list that excludes the last item by slicing. This doesn't modify the original list but creates a new one.

Example :-

my\_list = [1, 2, 3, 4, 5]

my\_list = my\_list[:-1]

print(my\_list) # Output: [1, 2, 3, 4]

1. **Suppose list1 is [2, 33, 222, 14, and 25], what is list1 [-1]?**

**Answer :-**

list1 = [2, 33, 222, 14, 25]

print(list1[-1])

**Output:** 25

1. **Differentiate between append () and extend () methods?**

**Answer :-**

|  |  |  |
| --- | --- | --- |
| **Feature** | **append()** | **extend()** |
| **Purpose** | Adds its argument as a single element to the end of the list. | Adds each element of its argument to the list. |
| **Argument Type** | Any type (object, list, tuple, etc.) | Iterable (list, tuple, set, etc.) |
| **Modification** | Increases the list length by 1, regardless of the argument's size. | Increases the list length by the number of elements in the iterable. |
| **Example** | my\_list.append([1, 2, 3]) | my\_list.extend([1, 2, 3]) |
| **Resulting List** | Original list with a single new element which is a list. | Original list with new elements from the argument added individually. |
| **Example List Before** | [a, b, c] | [a, b, c] |
| **Example List After** | [a, b, c, [1, 2, 3]] | [a, b, c, 1, 2, 3] |

1. **Write a Python function to get the largest number, smallest num and sum of all from a list.**

**Answer :-**

def list\_num(numbers):

if numbers:

largest = max(numbers)

smallest = min(numbers)

total\_sum = sum(numbers)

return largest, smallest, total\_sum

else:

return None, None, 0

my\_list = [10, 5, 17, 8, 3]

largest\_num, smallest\_num, sum\_all = analyze\_list(my\_list)

print("List:", my\_list)

print("Largest number:", largest\_num)

print("Smallest number:", smallest\_num)

print("Sum of all numbers:", sum\_all)

OUTPUT:-

List: [2, 6, 33, 200, 100, 35]

Largest number: 200

Smallest number: 2

Sum of all numbers: 376

1. **How will you compare two lists?**

**Answer :-**

To compare two lists in Python, you can use various methods depending on what comparison you want to perform whether they are equal, contain the same elements, or have elements in the same order.

1. **Equality Check (==)**:

Use the equality operator == to check if two lists are exactly equal, meaning they have the same elements in the same order.

Example :-

list1 = [1, 2, 3]

list2 = [1, 2, 3]

if list1 == list2:

print("Lists are equal")

else:

print("Lists are not equal")

OUTPUT :- Lists are equal

1. **Identity Check (is)**:

Use the identity operator is to check if two lists are the same object in memory.

Example :-

list1 = [1, 2, 3]

list2 = [1, 2, 3]

if list1 is list2:

print("Lists are identical")

else:

print("Lists are not identical")

OUTPUT :- Lists are not identical

1. **Comparison by Elements**:

Use set operations or iteration to check if two lists contain the same elements, regardless of order.

Example :-

list1 = [1, 2, 3]

list2 = [3, 2, 1]

if set(list1) == set(list2):

print("Lists have the same elements")

else:

print("Lists do not have the same elements")

OUTPUT :- Lists have the same elements

1. **Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.**

**Answer :-**

str\_list = ['abc', 'xyz', 'aba', '1221', 'aa', 'x', 'abca']

count = 0

for s in str\_list:

if len(s) >= 2 and s[0] == s[-1]:

count += 1

print("Number of strings:", count)

OUTPUT :- Number of strings: 4

1. **Write a Python program to remove duplicates from a list.**

**Answer :-**

my\_list = [1, 2, 3, 4, 2, 3, 5, 6, 1]

unique\_list = []

for item in my\_list:

if item not in unique\_list:

unique\_list.append(item)

print("Original List:", my\_list)

print("List with Duplicates Removed:", unique\_list)

OUTPUT:-

Original List: [1, 2, 3, 4, 2, 3, 5, 6, 1]

List with Duplicates Removed: [1, 2, 3, 4, 5, 6]

1. **Write a Python program to check a list is empty or not.**

**Answer :-**

my\_list = []

if len(my\_list) == 0:

print("List is empty")

else:

print("List is not empty")

OUTPUT :-

List is empty

1. **Write a Python function that takes two lists and returns true if they have at least one common member.**

**Answer :-**

def common\_member(list1, list2):

for item1 in list1:

for item2 in list2:

if item1 == item2:

return True

return False

list1 = [1, 2, 3, 4, 5]

list2 = [5, 6, 7, 8, 9]

print(have\_common\_member(list1, list2))

list3 = [11, 22, 33]

list4 = [6, 7, 8, 9]

print(have\_common\_member(list3, list4))

OUTPUT :-

True

False

1. **Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30.**

**Answer :-**

squares = [(i \*\* 2) for i in range(1, 31)]

print("First 5 elements:")

print(squares[:5])

print("\nLast 5 elements:")

print(squares[-5:])

OUTPUT:-

First 5 elements:

[1, 4, 9, 16, 25]

Last 5 elements:

[676, 729, 784, 841, 900]

1. **Write a Python function that takes a list and returns a new list with unique elements of the first list.**

**Answer :-**

def elements(input\_list):

unique\_list = []

for item in input\_list:

if item not in unique\_list:

unique\_list.append(item)

return unique\_list

my\_list = [1, 2, 2, 3, 4, 4, 5]

print(unique\_elements(my\_list))

OUTPUT :-

[1, 2, 3, 4, 5]

1. **Write a Python program to convert a list of characters into a string.**

**Answer :-**

char\_list = ['H', 'e', 'l', 'l', 'o']

char\_string = ''.join(char\_list)

print("Converted string:", char\_string)

OUTPUT :-

Converted string: Hello

1. **Write a Python program to select an item randomly from a list.**

**Answer :-**

import random

my\_list = ['apple', 'banana', 'cherry', 'date', 'elderberry']

random\_item = random.choice(my\_list)

print("Randomly selected item:", random\_item)

OUTPUT :-

Randomly selected item: cherry

1. **Write a Python program to find the second smallest number in a list.**

**Answer :-**

my\_list = [45, 23, 11, 56, 78, 2, 10]

smallest = float('inf')

second\_smallest = float('inf')

for num in my\_list:

if num < smallest:

second\_smallest = smallest

smallest = num

elif num < second\_smallest and num != smallest:

second\_smallest = num

print("Second smallest number:", second\_smallest)

OUTPUT :-

Second smallest number: 10

1. **Write a Python program to get unique values from a list**

**Answer :-**

my\_list = [1, 2, 2, 3, 4, 4, 5, 5, 6, 7, 7]

unique\_values = []

for item in my\_list:

if item not in unique\_values:

unique\_values.append(item)

print("Unique values:", unique\_values)

OUTPUT :-

Unique values: [1, 2, 3, 4, 5, 6, 7]

1. **Write a Python program to check whether a list contains a sub list**

**Answer :-**

main\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

sub\_list = [3, 4, 5]

found = False

for i in range(len(main\_list) - len(sub\_list) + 1):

if main\_list[i:i + len(sub\_list)] == sub\_list:

found = True

break

if found:

print("Sublist found in the main list")

else:

print("Sublist not found in the main list")

OUTPUT :-

Sublist found in the main list

1. **Write a Python program to split a list into different variables.**

**Answer :-**

my\_list = [1, 2, 3, 4, 5]

a, b, c, d, e = my\_list

print("Variables:", a, b, c, d, e)

OUTPUT :-

Variables: 1 2 3 4 5

1. **What is tuple? Difference between list and tuple.**

**Answer :-**

**Tuple:**

A tuple in Python is an ordered collection of items, similar to a list. However, tuples are immutable, meaning once created, their content cannot be changed.

**Syntax**: Tuples are defined using parentheses () and comma , to separate elements.

**Example**: my\_tuple = (1, 2, 3, 'a', 'b', 'c')

**List vs. Tuple:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **List** | **Tuple** |
| **Mutability** | Mutable (can be changed after creation) | Immutable (cannot be changed after creation) |
| **Syntax** | Defined using square brackets [] | Defined using parentheses () |
| **Elements** | Elements can be of different data types | Elements can be of different data types |
| **Performance** | Slightly slower for access and iteration | Slightly faster for access and iteration |
| **Use Cases** | Use when you need a collection that can be modified | Use when you want to protect data from being modified accidentally |
| **Examples** | my\_list = [1, 2, 3, 'a', 'b', 'c'] | my\_tuple = (1, 2, 3, 'a', 'b', 'c') |

1. **Write a Python program to create a tuple with different data types.**

**Answer :-**

my\_tuple = (1, 'hello', 3.14, True)

print("Tuple with different data types:", my\_tuple)

OUTPUT :-

Tuple with different data types: (1, 'hello', 3.14, True)

1. **Write a Python program to unzip a list of tuples into individual lists.**

**Answer :-**

list\_of\_tuples = [(1, 'a'), (2, 'b'), (3, 'c')]

unzipped = list(zip(\*list\_of\_tuples))

print("Unzipped lists:")

for item in unzipped:

print(list(item))

OUTPUT :-

Unzipped lists:

[1, 2, 3]

['a', 'b', 'c']

1. **Write a Python program to convert a list of tuples into a dictionary.**

**Answer :-**

list\_of\_tuples = [('a', 1), ('b', 2), ('c', 3)]

my\_dict = dict(list\_of\_tuples)

print("Dictionary:", my\_dict)

OUTPUT :-

Dictionary: {'a': 1, 'b': 2, 'c': 3}

1. **How will you create a dictionary using tuples in python?**

**Answer :-**

list\_of\_tuples = [('a', 1), ('b', 2), ('c', 3)]

my\_dict = dict(list\_of\_tuples)

print("Dictionary:", my\_dict)

OUTPUT :-

Dictionary: {'a': 1, 'b': 2, 'c': 3}

1. **Write a Python script to sort (ascending and descending) a dictionary by value.**

**Answer :-**

my\_dict = {'a': 3, 'b': 1, 'c': 2}

sorted\_dict\_asc = dict(sorted(my\_dict.items(), key=lambda item: item[1]))

print("Dictionary sorted by value (ascending):", sorted\_dict\_asc)

OUTPUT :-

Dictionary sorted by value (ascending): {'b': 1, 'c': 2, 'a': 3}

1. **Write a Python script to concatenate following dictionaries to create a new one.**

**Answer :-**

dict1 = {'a': 1, 'b': 2}

dict2 = {'c': 3, 'd': 4}

dict3 = {'e': 5, 'f': 6}

new\_dict = {}

new\_dict.update(dict1)

new\_dict.update(dict2)

new\_dict.update(dict3)

print("Concatenated dictionary using update():", new\_dict)

OUTPUT :-

Concatenated dictionary using update(): {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}

1. **Write a Python script to check if a given key already exists in a dictionary.**

**Answer :-**

my\_dict = {'a': 1, 'b': 2, 'c': 3}

key\_to\_check = 'b'

if key\_to\_check in my\_dict:

print(f"Key '{key\_to\_check}' exists in the dictionary.")

else:

print(f"Key '{key\_to\_check}' does not exist in the dictionary.")

OUTPUT :- Key 'b' exists in the dictionary.

1. **How Do You Traverse Through a Dictionary Object in Python?**

**Answer :-**

my\_dict = {'a': 1, 'b': 2, 'c': 3}

# Traversing through keys

for key in my\_dict:

print("Key:", key)

# Traversing through values

for value in my\_dict.values():

print("Value:", value)

# Traversing through values

for value in my\_dict.values():

print("Value:", value)

OUTPUT :-

Key: a

Key: b

Key: c

Value: 1

Value: 2

Value: 3

Value: 1

Value: 2

Value: 3

1. **How Do You Check the Presence of a Key in A Dictionary?**

**Answer :-**

my\_dict = {'a': 1, 'b': 2, 'c': 3}

key\_to\_check = 'b'

if key\_to\_check in my\_dict:

print(f"Key '{key\_to\_check}' exists in the dictionary.")

else:

print(f"Key '{key\_to\_check}' does not exist in the dictionary.")

OUTPUT :-

Key 'b' exists in the dictionary.

1. **Write a Python script to print a dictionary where the keys are numbers between 1 and 15.**

**Answer :-**

my\_dict = {i: i\*\*2 for i in range(1, 16)}

print("Dictionary with keys from 1 to 15 and their squares as values:")

print(my\_dict)

OUTPUT :-

Dictionary with keys from 1 to 15 and their squares as values:

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225}

1. **Write a Python program to check multiple keys exists in a dictionary**

**Answer :-**

my\_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

keys\_to\_check = ['a', 'b', 'e']

all\_keys\_exist = all(key in my\_dict for key in keys\_to\_check)

if all\_keys\_exist:

print(f"All keys {keys\_to\_check} exist in the dictionary.")

else:

print(f"Not all keys {keys\_to\_check} exist in the dictionary.")

OUTPUT :-

Not all keys ['a', 'b', 'e'] exist in the dictionary.

1. **Write a Python script to merge two Python dictionaries**

**Answer :-**

dict1 = {'a': 1, 'b': 2}

dict2 = {'c': 3, 'd': 4}

dict1.update(dict2)

print("Merged dictionary using update():", dict1)

OUTPUT :-

Merged dictionary using update(): {'a': 1, 'b': 2, 'c': 3, 'd': 4}

1. **Write a Python program to map two lists into a dictionary Sample output: Counter ({'a': 400, 'b': 400,’d’: 400, 'c': 300}).**

**Answer :-**

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

values = [400, 400, 300, 400]

mapped\_dict = dict(zip(keys, values))

print("Mapped dictionary:", mapped\_dict)

OUTPUT :-

Mapped dictionary: {'a': 400, 'b': 400, 'c': 300, 'd': 400}

1. **Write a Python program to find the highest 3 values in a dictionary**

**Answer :-**

my\_dict = {'a': 50, 'b': 100, 'c': 75, 'd': 125, 'e': 60}

highest\_values = sorted(my\_dict.values(), reverse=True)[:3]

print("The highest 3 values in the dictionary are:", highest\_values)

OUTPUT :-

The highest 3 values in the dictionary are: [125, 100, 75]

1. **Write a Python program to combine values in python list of dictionaries.**

**Sample data: [{'item': 'item1', 'amount': 400}, {'item': 'item2', 'amount': 300}, o {'item': 'item1', 'amount': 750}]**

**Expected Output:**

**Counter ({'item1': 1150, 'item2': 300})**

**Answer :-**

from collections import Counter

data = [

{'item': 'item1', 'amount': 400},

{'item': 'item2', 'amount': 300},

{'item': 'item1', 'amount': 750}

]

combined\_data = Counter()

for entry in data:

combined\_data[entry['item']] += entry['amount']

print("Combined values in the list of dictionaries:")

print(combined\_data)

OUTPUT :-

Combined values in the list of dictionaries:

Counter({'item1': 1150, 'item2': 300})

1. **Write a Python program to create a dictionary from a string.**

**Note: Track the count of the letters from the string.**

**Answer :-**

input\_string = "hello world"

char\_count = {}

for char in input\_string:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

print("Dictionary with character counts:")

print(char\_count)

OUTPUT :-

Dictionary with character counts:

{'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}

1. **Sample string:**

**'w3resource' Expected output:**

**{'3': 1,’s’: 1, 'r': 2, 'u': 1, 'w': 1, 'c': 1, 'e': 2, 'o': 1}**

**Answer :-**

input\_string = 'w3resource'

char\_count = {}

for char in input\_string:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

print("Dictionary with character counts:")

print(char\_count)

OUTPUT :-   
Dictionary with character counts: {'w': 1, '3': 1, 'r': 2, 'e': 2, 's': 1, 'o': 1, 'u': 1, 'c': 1}

1. **Write a Python function to calculate the factorial of a number (a nonnegative integer)**

**Answer :-**

def factorial(n):

# Base case: factorial of 0 is 1

if n == 0:

return 1

# Recursive case: compute factorial

else:

return n \* factorial(n - 1)

# Test the function

number = 5 # Example: calculate 5!

print(f"The factorial of {number} is:", factorial(number))

OUTPUT :-

The factorial of 5 is: 120

1. **Write a Python function to check whether a number is in a given range**

**Answer :-**

def check\_range(num, start, end):

if start <= num <= end:

return True

else:

return False

number = 10

start\_range = 5

end\_range = 15

if check\_range(number, start\_range, end\_range):

print(f"{number} is in the range [{start\_range}, {end\_range}]")

else:

print(f"{number} is not in the range [{start\_range}, {end\_range}]")

OUTPUT :-

10 is in the range [5, 15]

1. **Write a Python function to check whether a number is perfect or not.**

**Answer :-**

def is\_perfect\_number(num):

if num <= 1:

return False

sum\_of\_divisors = 0

for i in range(1, num):

if num % i == 0:

sum\_of\_divisors += i

if sum\_of\_divisors == num:

return True

else:

return False

number = 28

if is\_perfect\_number(number):

print(f"{number} is a perfect number.")

else:

print(f"{number} is not a perfect number.")

OUTPUT :- 28 is a perfect number.

1. **Write a Python function that checks whether a passed string is palindrome or not**

**Answer :-**

def is\_palindrome(s):

s = s.lower().replace(" ", "")

return s == s[::-1]

input\_string = "Madam"

if is\_palindrome(input\_string):

print(f"{input\_string} is a palindrome.")

else:

print(f"{input\_string} is not a palindrome.")

OUTPUT :- Madam is a palindrome.

1. **How Many Basic Types of Functions Are Available in Python?**

**Answer :-**

there are two basic types of functions:

1. **Built-in Functions**: These are functions that are already defined in Python and are available for use. Examples include print(), len(), type(), and range().

Example :-

print("Hello, World!") # This uses the built-in print() function

1. **User-defined Functions**: These are functions that users create to perform specific tasks. They are defined using the def keyword followed by the function name and parentheses containing any parameters.

Example :-

def greet(name):

return f"Hello, {name}!"

print(greet("Alice")) # This calls the user-defined greet() function

1. **How can you pick a random item from a list or tuple?**

**Answer :-**

import random

my\_list = [1, 2, 3, 4, 5]

random\_item = random.choice(my\_list)

print("list:",random\_item)

my\_tuple = [1, 2, 3, 4, 5]

random\_item = random.choice(my\_tuple)

print("Tuple:",random\_item)

OUTPUT:-

list: 2

Tuple: 4

1. **How can you pick a random item from a range?**

**Answer :-**

import random

range\_list = list(range(1, 11))

random\_item = random.choice(range\_list)

print(random\_item)

OUTPUT :- 3

1. **How can you get a random number in python?**

**Answer :-**

import random

random\_number = random.randint(1, 10)

print(random\_number)

OUTPUT :- 4

1. **How will you set the starting value in generating random numbers?**

**Answer :-**

1. **How will you randomize the items of a list in place?**

**Answer :-**

import random

my\_list = [1, 2, 3, 4, 5]

list1 = random.sample(my\_list, len(my\_list))

print("list:", list1)

OUTPUT :-

list: [1, 5, 4, 3, 2]

1. **What is File function in python? What are keywords to create and write file.**

**Answer :-**

1. **Write a Python program to read an entire text file.**

**Answer :-**

1. **Write a Python program to append text to a file and display the text.**

**Answer :-**

1. **Write a Python program to read first n lines of a file.**

**Answer :-**

1. **Write a Python program to read last n lines of a file.**

**Answer :-**

1. **Write a Python program to read a file line by line and store it into a list**

**Answer :-**

1. **Write a Python program to read a file line by line store it into a variable.**

**Answer :-**

1. **Write a python program to find the longest words.**

**Answer :-**

1. **Write a Python program to count the number of lines in a text file.**

**Answer :-**

1. **Write a Python program to count the frequency of words in a file.**

**Answer :-**

1. **Write a Python program to write a list to a file.**

**Answer :-**

1. **Write a Python program to copy the contents of a file to another file.**

**Answer :-**

1. **Explain Exception handling? What is an Error in Python?**

**Answer :-**

1. **How many except statements can a try-except block have? Name Some built-in exception classes:**

**Answer :-**

1. **When will the else part of try-except-else be executed?**

**Answer :-**

1. **Can one block of except statements handle multiple exception?**

**Answer :-**

1. **When is the finally block executed?**

**Answer :-**

1. **What happens when „1‟== 1 is executed?**

**Answer :-**

1. **How Do You Handle Exceptions with Try/Except/Finally in Python? Explain with coding snippets.**

**Answer :-**

1. **Write python program that user to enter only odd numbers, else will raise exception.**

**Answer :-**