Module 7) Python – Collections, functions and Modules

1. Accessing List

Question-1: Understanding how to create and access elements in a list.

Answer:

Creating a List

Here are some common methods to create a list:

Using Square Brackets

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

Using list() Constructor

a = list((1, 2, 3, 'apple', 4.5))

Creating List with Repeated Elements

# Create a list [2, 2, 2, 2, 2]

a = [2] * 5

Accessing List Elements

Elements in a list can be accessed using indexing. Python indexes start at 0, so a[0] will access the first element, while negative indexing allows us to access elements from the end of the list. Like index -1 represents the last elements of list.

a = [10, 20, 30, 40, 50]

# Access first element

print(a[0])

```
# Access last element
print(a[-1])
```

Question-2: Indexing in lists (positive and negative indexing).

Answer: Python indexes start at 0, so a[0] will access the first element, while negative indexing allows us to access elements from the end of the list. Like index -1 represents the last elements of list.

```
a = [10, 20, 30, 40, 50]

# Access first element
print(a[0])

# Access last element
print(a[-1])
```

Question-3: Slicing a list: accessing a range of elements.

Answer: Python list slicing is fundamental concept that let us easily access specific elements in a list. In this article, we'll learn the syntax and how to use both positive and negative indexing for slicing with examples.

Example: Get the items from a list starting at position 1 and ending at position 4 (exclusive).

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# Get elements from index 1 to 4 (excluded)
print(a[1:4])
```

2. List Operations

Question-1: Common list operations: concatenation, repetition, membership

Answer:

Concatenation: The + operator creates a new list by concatenating list1 and list2. This operation does not modify the original lists but returns a new combined list.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
```

print(c)

Output

[1, 2, 3, 4, 5, 6]

Repetition: In Python, we often need to add duplicate values to a list, whether for creating repeated patterns, frequency counting, or handling utility cases. In this article, there are various methods to Repeat an element in a List. * operator allows us to repeat an entire list or a list element multiple times, making it the most efficient and simplest method.

a = [1, 2, 3]

#repeated list

r = a * 3

print(r)

Output

[1, 2, 3, 1, 2, 3, 1, 2, 3]

Membership: The Python membership operators test for the membership of an object in a sequence, such as strings, lists, or tuples. Python offers two membership operators to check or validate the membership of a value. They are as follows:

Python IN Operator

The in operator is used to check if a character/substring/element exists in a sequence or not. Evaluate to True if it finds the specified element in a sequence otherwise False.

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

# using membership 'in' operator

# checking an integer in a list

print(2 in list1)

Output

```
True
```

Python NOT IN Operator

The ‘not in’ Python operator evaluates to true if it does not find the variable in the specified sequence and false otherwise.

```
# initialized some sequences
list1 = [1, 2, 3, 4, 5]
# using membership 'not in' operator
# checking an integer in a list
print(2 not in list1)
```

Output

```
False
```

Question-2: Understanding list methods like append(), insert(), remove(), pop()

Answer:

append: The append() method in Python is used to add a single item to the end of list. This method modifies the original list and does not return a new list.

```
a = [2, 5, 6, 7]
# Use append() to add the element 8
# to the end of the list
a.append(8)
print(a)
```

Output

```
[2, 5, 6, 7, 8]
```

Insert: Python List insert() method inserts an item at a specific index in a list.

```
# creating a list

fruit = ["banana","cherry","grape"]

fruit.insert(1,"apple")

print(fruit)
```

Output

```
['banana', 'apple', 'cherry', 'grape']
```

Remove: Python list remove() function removes the first occurrence of a given item from list. It make changes to the current list. It only takes one argument, element you want to remove and if that element is not present in the list, it gives ValueError.

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

a.remove("b")

print(a)
```

Output

```
['a', 'c']
```

Pop: The pop() method is used to remove an element from a list at a specified index and return that element. If no index is provided, it will remove and return the last element by default. This method is particularly useful when we need to manipulate a list dynamically, as it directly modifies the original list.

```
a = [10, 20, 30, 40]

# Remove the last element from list
```

```
a.pop()

print(a)
```

Output

```
[10, 20, 30]
```

3. Working with Lists

Question-1: Iterating over a list using loops.

Answer: Python provides several ways to iterate over list. The simplest and the most common way to iterate over a list is to use a for loop. This method allows us to access each element in the list directly.

```
a = [1, 3, 5, 7, 9]

# On each iteration val

# represents the current item/element

for val in a:

    print(val)
```

Output

```
1

3

5

7

9
```

Question-2: Sorting and reversing a list using sort(), sorted(), and reverse().

Answer:

Sort: The sort() method in Python is a built-in function that allows us to sort the elements of a list in ascending or descending order and it modifies the list in place which means there is no new list created. This method is useful when working with lists where we need to arranged the elements in a specific order, whether numerically or alphabetically.

```
a = [5, 2, 9, 1, 5, 6]

# Sort the value in increasing order

a.sort()

print(a)
```

Output

```
[1, 2, 5, 5, 6, 9]
```

Sorted: sorted() function returns a new sorted list from the elements of any iterable like (e.g., list, tuples, strings). It creates and returns a new sorted list and leaves the original iterable unchanged.

```
a = [4, 1, 3, 2]

#Using sorted function to modify list in-place

b = sorted(a)

print(b)
```

Output

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

Reverse: The reverse() method reverses the elements of the list in-place and it modify the original list without creating a new list. This method is efficient because it doesn't create a new list.

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

# Reverse the list in-place

a.reverse()

print(a)
```

Output

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

Question-3: Basic list manipulations: addition, deletion, updating, and slicing.

Answer:

Addition: To add an item to the end of the list, use the append() method.

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Output:

```
['apple', 'banana', 'cherry', 'orange']
```

Deletion: The remove() method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Output:

```
['apple', 'cherry']
```

Updating: To change the value of a specific item, refer to the index number.

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Output:

```
['apple', 'blackcurrant', 'cherry']
```

Slicing: Python list slicing is fundamental concept that let us easily access specific elements in a list.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Get elements from index 1 to 4 (excluded)

print(a[1:4])
```

Output

```
[2, 3, 4]
```

4. Tuple

Question-1: Introduction to tuples, immutability.

Answer: Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

Question-2: Creating and accessing elements in a tuple.

Answer:

Creating a Tuple

In Python Programming, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping the data sequence.

```
# Creating an empty Tuple

Tuple1 = ()

print("Initial empty Tuple: ")

print(Tuple1)


# Creating a Tuple

# with the use of string

Tuple1 = ('Geeks', 'For')

print("\nTuple with the use of String: ")

print(Tuple1)


# Creating a Tuple with
# the use of list
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))

# Creating a Tuple
# with the use of built-in function
Tuple1 = tuple('Geeks')
print("\nTuple with the use of function: ")
print(Tuple1)
```

Output:

```
Initial empty Tuple:
()

Tuple with the use of String:
('Geeks', 'For')

Tuple using List:
(1, 2, 4, 5, 6)

Tuple with the use of function:
('G', 'e', 'e', 'k', 's')
```

Accessing of Tuples

In Python Programming, Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

```
Tuple1 = tuple("Geeks")

print("\nFirst element of Tuple: ")

print(Tuple1[0])
```

Output:

```
First element of Tuple:
G
```

Question-3: Basic operations with tuples: concatenation, repetition, membership.

Answer:

Concatenation: Using + operator This is the most Pythonic and recommended method to perform this particular task. In this, we add two tuples and return the concatenated tuple. No previous tuple is changed in this process.

```
# initialize tuples

test_tup1 = (1, 3, 5)

test_tup2 = (4, 6)

# using + operator

res = test_tup1 + test_tup2

# printing result

print("The tuple after concatenation is : " + str(res))
```

Output :

```
The tuple after concatenation is : (1, 3, 5, 4, 6)
```

Repetition: Using * operator The multiplication operator can be used to construct the duplicates of a container. This also can be extended to tuples even though tuples are immutable.

```
# initialize tuple

test_tup = (1, 3)

# Repeating tuples N times

# using * operator

res = ((test_tup, ) * N)

# printing result

print("The duplicated tuple elements are : " + str(res))
```

Output

```
The duplicated tuple elements are : ((1, 3), (1, 3), (1, 3), (1, 3))
```

Membership:

Python IN Operator

The in operator is used to check if a character/substring/element exists in a sequence or not. Evaluate to True if it finds the specified element in a sequence otherwise False.

```
# initialized some sequences

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

str1 = "Hello World"

dict1 = {1: "Geeks", 2:"for", 3:"geeks"}

# using membership 'in' operator

# checking an integer in a list

print(2 in list1)
```

Output

```
True
```

Python NOT IN Operator

The ‘not in’ Python operator evaluates to true if it does not find the variable in the specified sequence and false otherwise.

```
# initialized some sequences
list1 = [1, 2, 3, 4, 5]
str1 = "Hello World"
dict1 = {1: "Geeks", 2:"for", 3:"geeks"}
# using membership 'not in' operator
# checking an integer in a list
print(2 not in list1)
```

Output

```
False
```

5. Accessing Tuples

Question-1: Accessing tuple elements using positive and negative indexing.

Answer: You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Output

```
Banana
```

Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Question-2: Slicing a tuple to access ranges of elements.

Answer: In this example, the below code initializes a list of tuples and then extracts a sublist containing tuples at index 1 and 2 (inclusive) using slicing. The result is `sliced_list` containing `[(2, 'banana'), (3, 'orange')]`, which is then printed.

```
# Initialize list of tuples

list_of_tuples = [(1, 'apple'), (2, 'banana'), (3, 'orange'), (4, 'grape')]

# Extract elements from index 1 to 2

sliced_list = list_of_tuples[1:3]

# Display list

print(sliced_list)
```

Output

```
[(2, 'banana'), (3, 'orange')]
```

6. Dictionaries

Question-1: Introduction to dictionaries: key-value pairs.

Answer: A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. While using a Dictionary, sometimes, we need to add or modify the key/value inside the dictionary.

```
dict = {'key1':'geeks', 'key2':'for'}

dict['key3'] = 'Geeks'

dict['key4'] = 'is'

dict['key5'] = 'portal'

dict['key6'] = 'Computer'

print(dict)
```

Output:

```
Current Dict is: {‘key2’: ‘for’, ‘key1’: ‘geeks’} Updated Dict is: {‘key3’: ‘Geeks’, ‘key5’: ‘portal’, ‘key6’: ‘Computer’, ‘key4’: ‘is’, ‘key1’: ‘geeks’, ‘key2’: ‘for’}
```

Question-2: Accessing, adding, updating, and deleting dictionary elements.

Answer:

Accessing: We can access a value from a dictionary by using the key within square brackets or get()method.

```
d = { "name": "Alice", 1: "Python", (1, 2): [1,2,4] }
# Access using key
print(d["name"])
# Access using get()
print(d.get("name"))
```

Output

```
Alice
Alice
```

Adding, Updating: We can add new key-value pairs or update existing keys by using assignment.

```
d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
# Adding a new key-value pair
d["age"] = 22
# Updating an existing value
d[1] = "Python dict"
print(d)
```

Output

```
{1: 'Python dict', 2: 'For', 3: 'Geeks', 'age': 22}
```

Deleting:

```
d = {1: 'Geeks', 2: 'For', 3: 'Geeks', 'age':22}
# Using del to remove an item
del d["age"]
print(d)
```

Output

```
Geeks
```

Question-3: Dictionary methods like keys(), values(), and items().

Answer:

Keys:

```
# Dictionary with three keys
Dictionary1 = {'A': 'Geeks', 'B': 'For', 'C': 'Geeks'}
# Printing keys of dictionary
print(Dictionary1.keys())
```

Output**:**

```
dict_keys(['A', 'B', 'C'])
```

Values:

```
dictionary = {"raj": 2, "striver": 3, "vikram": 4}
print(dictionary.values())
```

Output:

```
dict_values([2, 3, 4])
```

Items:

```
# Dictionary with three items
Dictionary1 = { 'A': 'Geeks', 'B': 4, 'C': 'Geeks' }
# Printing all the items of the Dictionary
print(Dictionary1.items())
```

Output:

```
dict_items([('A', 'Geeks'), ('B', 4), ('C', 'Geeks')])
```

7. Working with Dictionaries

Question-1: Iterating over a dictionary using loops

Answer:

Iterate through Value

To iterate through all values of a dictionary in Python using .values(), you can employ a for loop, accessing each value sequentially. This method allows you to process or display each individual value in the dictionary without explicitly referencing the corresponding keys.

```
# create a python dictionary

d = {"name": "Geeks", "topic": "dict", "task": "iterate"}

# loop over dict values

for val in d.values():

    print(val)
```

Iterate through keys

In Python, just looping through the dictionary provides you its keys. You can also iterate keys of a dictionary using built-in `.keys()` method.

```
# create a python dictionary

d = {"name": "Geeks", "topic": "dict", "task": "iterate"}

# default loooping gives keys

for keys in d:

    print(keys)

# looping through keys

for keys in d.keys():

  print(keys)
```

Question-2: Merging two lists into a dictionary using loops or zip().

Answer: This method uses the built-in dict() function to create a dictionary from two lists using the zip() function. The zip() function pairs the elements in the two lists together, and dict() converts the resulting tuples to key-value pairs in a dictionary.

```
# initializing lists

test_keys = ["Rash", "Kil", "Varsha"]

test_values = [1, 4, 5]

# Printing original keys-value lists

print("Original key list is : " + str(test_keys))

print("Original value list is : " + str(test_values))

# using dict() and zip() to convert lists to dictionary

res = dict(zip(test_keys, test_values))

# Printing resultant dictionary

print("Resultant dictionary is : " + str(res))
```

Output:

```
Original key list is : ['Rash', 'Kil', 'Varsha']
Original value list is : [1, 4, 5]
Resultant dictionary is : {'Rash': 1, 'Kil': 4, 'Varsha': 5}
```

Question-3: Counting occurrences of characters in a string using dictionaries.

Answer: A dictionary comprehension can also be used to count character frequencies in a concise form.

```
s = "GeeksforGeeks"

# Count characters using dictionary comprehension

freq = {c: s.count(c) for c in s}

print(freq)
```

Output

```
{'G': 2, 'e': 4, 'k': 2, 's': 2, 'f': 1, 'o': 1, 'r': 1}
```

8. Functions

Question-1: Defining functions in Python.

Answer: Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Question-2: Different types of functions: with/without parameters, with/without return values.

Answer:

Python Function with Parameters

If you have experience in C/C++ or Java then you must be thinking about the *return type* of the function and *data type* of arguments. That is possible in Python as well (specifically for Python 3.5 and above).

Python Function Syntax with Parameters

```
def function_name(parameter: data_type) -> return_type:
    """Docstring"""
    # body of the function
    return expression
```

Output:

```
The addition of 5 and 15 results 20.
```

Python Function without Parameters

```
# A simple Python function
def fun():
    print("Welcome to GFG")
# Driver code to call a function
fun()
```

Output**:**

```
Welcome to GFG
```

Question-3: Anonymous functions (lambda functions).

Answer: Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in Python.

```
s1 = 'GeeksforGeeks'
s2 = lambda func: func.upper()
print(s2(s1))
```

Output

```
GEEKSFORGEEKS
```

9. Modules

Question-1: Introduction to Python modules and importing modules.

Answer: Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

```
import mymodule
mymodule.greeting("Jonathan")
```

Question-2: Standard library modules: math, random.

Answer:

Math: Math Module is an in-built Python library made to simplify mathematical tasks in Python.

It consists of various mathematical constants and functions that can be used after importing the math module.

```
Import math
```

Random: Python Random module generates random numbers in Python. These are pseudo-random numbers means they are not truly random.

This module can be used to perform random actions such as generating random numbers, printing random a value for a list or string, etc. It is an in-built function in Python.

Question-3: Creating custom modules.

Answer: A module is simply a Python file with a .py extension that can be imported inside another Python program. The name of the Python file becomes the module name. The module contains definitions and implementation of classes, variables, and functions that can be used inside another program.

```
''' GFG.py '''

# Python program to create

# a module

# Defining a function

def Geeks():

    print("GeeksforGeeks")

# Defining a variable

location = "Noida"
```

The above example shows the creation of a simple module named GFG as the name of the above Python file is GFG.py. When this code is executed it does nothing because the function created is not invoked.

To use the above created module, create a new Python file in the same directory and import GFG module using the import statement.

```
# modules
import GFG
# Use the function created
GFG.Geeks()
# Print the variable declared
print(GFG.location)
```

Output:

```
GeeksforGeeks
Noida
```