**Python – Collections, functions and Modules**

**Q1. Understanding how to create and access elements in a list.**

**Q2. Indexing in lists (positive and negative indexing).**

**Ans.** A list in Python is a data structure that stores an ordered collection of items. These items can be:

* Numbers
* Strings
* Boolean
* Other lists
* Or a mix of different types

To create a list, you define a sequence of items grouped together. These items are separated by commas and enclosed in square brackets [ ].

Every item in a list has a index. The first item is at position 0, The second is at position 1, And so on...

Python also allows negative indexing, which means you can count from the end. -1 means the last number, -2 means the second to last, etc.

We uses list to store multiple values in one variable, to group related data, to easily retrieve, modify, or loop through items.

**Q3. Slicing a list: accessing a range of elements.**

Ans. Slicing is like taking a slice from a cake. you don’t want the whole list, just a part of it.

When you slice a list, you choose:

* Where to start
* Where to stop
* (Optionally) How big each step should be
* When slicing, you use this pattern:

List [start : stop]

Start = index where slicing begins

Stop = index where slicing ends

* You can add a third number to control how many items to skip:

list[start : stop : step]

**Q4. Common list operations: concatenation, repetition, membership.**

Ans.

Concatenation means joining two lists into one—like putting two rows of boxes end to end.

List A + List B = New List have all items from both A & B

Ex. List A: ["apple", "banana"]

List B: ["cherry", "date"]

Concatenating gives :

["apple", "banana", "cherry", "date"]

Repetition means repeating the contents of a list a number of times—like making multiple copies.

List × 3 = List repeated 3 times

List : [“Vraj” , “Mohit”]

Repeating it 2 times gives:

[“Vraj” , “Mohit” , “Vraj” , “Mohit”]

Membership checks whether an item is **in** a list or **not in** a list.

Ex. List: ["cat", "dog", "bird"]

Check: “Is 'dog' in the list?” = Yes

Check: “Is 'elephant' in the list?” = No

**Q5. Understanding list methods like append(), insert(), remove(), pop().**

Ans.

1. append(): Add to the End

Imagine a List of 1 to 10 adding one more num 11 at the back.

append() adds one item to the end of the list.

2. insert(): Add at a Specific Position

Imagine a list of 1 to 10 not have 5 and then add 5 to their specific space.

insert(index, item) lets you place a new item at a specific position in the list.

The other items move to make space.

3. remove(): Delete by Value

Imagine finding a specific num 5 in a list and taking it out.

remove(value) looks through the list and deletes the first matching item.

If the item isn’t there, Python complains.

4. pop(): Remove by Position (and Return It)

Like in list of 1 to 10 remove from 5 num index

pop(index) removes the item at a specific position and gives it back.

If no index is given, it pops the last item by default.

**Q6. Iterating over a list using loops.**

Ans. Iterating just means going through each item in a list, one at a time, and doing something with each item.

A loop helps you repeat an action for every item in a list, automatically—without writing separate code for each one.

A for loop lets you look at each item directly in order.

“For each item in this list, do this…”

It goes item by item—from the first to the last.

**Q7. Sorting and reversing a list using sort(), sorted(), and reverse().**

Ans. Sorting means arranging items in a specific order:

Alphabetical (A to Z)

Numerical (smallest to largest)

sort() changes the original list permanently.

sorted() leaves your original list unchanged and gives you a new, sorted copy.

Reversing means flipping the list so the last item becomes first, and so on.

reverse() just flips the list as it is, without sorting.

**Q8. Basic list manipulations: addition, deletion, updating, and slicing.**

Ans. Addition (Adding Items to a List)

You can add items using:

append() : adds to the end

insert() : adds at a specific position

+ (concatenation) : joins two lists into a new one

Deletion (Removing Items from a List)

You can delete items using:

remove(value) : removes a specific item

pop(index) : removes item at a specific position

del : deletes an item or even the entire list

Updating (Changing Items in a List)

You can change an item directly by assigning a new value to a specific index.

Slicing (Getting a Sublist)

You can extract a part of the list using:

list[start : stop]

Gets items from start up to (but not including) stop

list[start : stop : step]

**Q9. Introduction to tuples, immutability.**

Ans. A tuple is a type of data structure in Python—very similar to a list.

Just like a list, a tuple:

Stores multiple values (numbers, strings, etc.)

Keeps those values in order

Allows duplicates

Tuples Are Immutable

Immutability means:

"Once created, it cannot be changed."

You can’t add, remove, or change items in a tuple.

You can only access the data, not modify it.

We Use Tuple:

You want to protect the data from being changed by mistake.

Tuples are slightly faster and use less memory than lists.

**Q10. Creating and accessing elements in a tuple.**

Ans: A tuple is like a list, but it's immutable—you can’t change it after it's made.

It’s a collection of items, written with round brackets:

(“Vraj”, “Mohit”)

Access Elements in a Tuple

Tuples are ordered, so you can use index numbers

tuple\_name[index]

name: (“Vraj”, “Mohit”)

name[0] = Vraj

name[1] = Mohit

**Q11. Basic operations with tuples: concatenation, repetition, membership.**

Ans. Concatenation :

tuple1 = (1, 2, 3)

tuple2 = (4, 5)

result = tuple1 + tuple2

print(result)

output: (1,2,3,4,5)

repetition :

tuple1 = (1, 2)

result = tuple1 \* 3

print(result) # Output: (1, 2, 1, 2, 1, 2)

Membership :

tuple1 = (10, 20, 30)

print(20 in tuple1) Output: True

print(40 not in tuple1) Output: True

**Q12. Accessing tuple elements using positive and negative indexing.**

Ans. Positive Indexing :

my\_tuple = ('a', 'b', 'c', 'd')

print(my\_tuple[0]) Output: 'a'

print(my\_tuple[2]) Output: 'c'

Negative Indexing :

print(my\_tuple[-1]) Output: 'd'

print(my\_tuple[-3]) Output: 'b'

**Q13. Slicing a tuple to access ranges of elements.**

Ans.

my\_tuple = ('a', 'b', 'c', 'd', 'e', 'f')

print(my\_tuple[1:4]) Output: ('b', 'c', 'd')

print(my\_tuple[:3]) Output: ('a', 'b', 'c')

print(my\_tuple[3:]) Output: ('d', 'e', 'f')

print(my\_tuple[-4:-1]) Output: ('c', 'd', 'e')

print(my\_tuple[::2]) Output: ('a', 'c', 'e')

print(my\_tuple[::-1]) Output: ('f', 'e', 'd', 'c', 'b', 'a')

**Q14. Introduction to dictionaries: key-value pairs.**

Ans. A dictionary is a built-in data type in Python used to store key-value pairs. It’s unordered, mutable, and indexed by keys.

dict = {

"name": "Vraj",

"age": 20,

"city": "Ahmedabad"

}

**Q15. Accessing, adding, updating, and deleting dictionary elements.**

Ans. person = {"name": "Vraj", "age": 20}

print(person["name"]) Output: Vraj

person["city"] = "Ahmedabad"

print(person) # {'name': 'Vraj', 'age': 20, 'city': 'Ahmedabad'}

person["age"] = 21

print(person["age"]) Output: 21

del person["city"]

person.clear()

**Q16. Dictionary methods like keys(), values(), and items().**

Ans. student = {

"name": "Vraj",

"age": 20,

"grade": "A"

}

print(student.keys())

Output: dict\_keys(['name', 'age', 'grade'])

Convert to list if needed:

print(list(student.keys()))

print(student.values())

Output: dict\_values(['John', 20, 'A'])

print(list(student.values()))

print(student.items())

Output: dict\_items([('name', 'John'), ('age', 20), ('grade', 'A')])

for key, value in student.items():

print(f"{key}: {value}")