❖ What is Data Structure in Python?

1. **Data Structure** is a way of organizing and managing data efficiently in a computer.
2. It helps in performing operations like searching, sorting, insertion, and deletion.
3. Python provides built-in data structures to store and process data easily.
4. Choosing the right data structure improves the performance of a program.

◆ Four Common Data Structures in Python

1. **List (list)** → Ordered, mutable collection used to store multiple values.
   Example: [1, 2, 3]
2. **Tuple (tuple)** → Ordered, immutable collection for fixed data.
   Example: (10, 20, 30)
3. **Dictionary (dict)** → Stores key-value pairs for fast data retrieval.
   Example: {'name': 'John', 'age': 25}
4. **Set (set)** → Unordered collection of unique elements.
   Example: {1, 2, 3, 4}

◆ Mutable and Immutable Data Structures in Python

1. **Mutable**
   - Can be modified after creation.
   - Allows adding, removing, or changing elements.
   - Example: list, dict, set
   - Code:

```
my_list = [1, 2, 3]

my_list.append(4)  # List is modified → [1, 2, 3, 4]
```

2. **Immutable**
   - Cannot be changed after creation.
   - Any modification creates a new object.
   - Example: tuple, string, frozenset
   - Code:

```
my_tuple = (10, 20, 30)

# my_tuple[1] = 50 # ❌ Error: Tuples cannot be modified
```

💡 **मराठी टीप (Marathi Tip)**

- **Mutable म्हणजे बदलता येणारे, Immutable म्हणजे कायमस्वरूपी बदल न होणारे!**
- **List आणि Dictionary कधीही संपादित करता येतात, पण Tuple आणि String कायम स्थिर असतात.**
- **Study tips – List बदलते, Tuple स्थिर असते, Set वेगळे असते, आणि Dictionary जलद शोधासाठी वापरते!**

3.1 Lists in Python (Most IMP remember all 3.1)

1 What is a List?

1. A **list** is an ordered collection of elements that can be **modified** (mutable) after creation.
2. Lists allow storage of **multiple data types** (e.g., integers, strings, booleans) in one variable.
3. They are defined using **square brackets []**, with each element separated by a comma.
4. Lists support various operations such as **accessing, updating, deleting, and iterating** through elements.

Real-Life Analogy:

Think of a list like a **shopping list**—you can add, remove, or change items on your shopping list as needed.

Example:

```
my_list = [10, "hello", 3.5, True]
print(my_list)  # Output: [10, "hello", 3.5, True]
```

---

2 How to Access Elements in a List?

Definition:

1. Elements in a list are accessed using **indexing**, where the first element has an index of 0.
2. **Positive indexing** starts from 0 and goes up to n-1 (for n elements).
3. **Negative indexing** starts from -1 (last element) and goes backwards.
4. **Slicing** lets you extract a subset of the list using [start:end:step].

Example:

```
my_list = [10, 20, 30, 40, 50, 60]
print(my_list[1])     # Output: 20 (Positive indexing)
print(my_list[-1])    # Output: 60 (Negative indexing)
print(my_list[1:4])   # Output: [20, 30, 40] (Slicing)
print(my_list[::2])   # Output: [10, 30, 50] (Step slicing)
```

Simple Diagram for Indexing & Slicing:

```
List:   [10, 20, 30, 40, 50, 60]
Index:   0   1   2   3   4   5   (Positive)
        -6  -5  -4  -3  -2  -1   (Negative)
```

Real-Life Analogy:

Accessing a list is like looking up items in a **menu** where each dish is numbered, and slicing is like choosing a range of dishes (e.g., appetizers from position 2 to 4).

---

3 How to Delete Values in a List

Definition:

1. Elements in a list can be deleted using various methods.
2. The **del** statement removes elements by index.
3. The **remove()** method deletes the first occurrence of a value.
4. The **pop()** method removes and returns the element at a specified index (or the last element by default).

Example:

```
my_list = [10, 20, 30, 40, 50]
del my_list[2]        # List becomes [10, 20, 40, 50]
my_list.remove(40)    # List becomes [10, 20, 50]
popped = my_list.pop() # List becomes [10, 20]; popped equals 50
```

---

4 How to Update a List

Definition:

1. Updating a list means **changing its content** without creating a new list.
2. Elements can be updated by assigning a new value using an index.
3. A range of elements can be updated via **slicing assignment**.
4. Methods like **append(), extend(), and insert()** help update lists by adding elements.

Example:

```
my_list = [10, 20, 30, 40]
my_list[1] = 25          # Updates index 1 → [10, 25, 30, 40]
my_list[2:4] = [35, 45]  # Updates indices 2 and 3 → [10, 25, 35, 45]
```

---

5 Six Basic List Operations

Definition :

1. **Concatenation (+)** – Combines two lists into one.
2. **Repetition (*)** – Repeats the list a specified number of times.
3. **Membership (in)** – Checks if an element is present in the list.
4. **Length (len())** – Returns the number of elements in the list.
5. **Iteration** – Looping through elements using a **for loop**.
6. **Sorting (sort())** – Arranges elements in ascending or descending order.

Example:

```
list1 = [1, 2, 3]
list2 = [4, 5]
print(list1 + list2)   # Concatenation: [1, 2, 3, 4, 5]
print(list1 * 2)       # Repetition: [1, 2, 3, 1, 2, 3]
print(2 in list1)      # Membership: True
print(len(list1))      # Length: 3
```

---

6 Six Methods of Lists

Definition:

1. **append()** – Adds an element to the end of the list.
2. **extend()** – Adds all elements from another list to the end.
3. **insert()** – Inserts an element at a specified index.
4. **remove()** – Removes the first occurrence of a specified value.
5. **pop()** – Removes and returns an element (last by default).
6. **clear()** – Removes all elements, leaving an empty list.

Example:

```
my_list = [10, 20]
my_list.append(30)        # [10, 20, 30]
my_list.insert(1, 15)     # [10, 15, 20, 30]
```

---

7 Six Built-in List Functions

Definition:

1. **len()** – Returns the number of elements in the list.
2. **max()** – Returns the largest element.
3. **min()** – Returns the smallest element.
4. **sum()** – Returns the sum of the list elements.
5. **sorted()** – Returns a sorted copy of the list.
6. **list()** – Converts an iterable to a list.

Example:

```
numbers = [5, 3, 8, 1]
print(len(numbers))   # Output: 4
print(max(numbers))   # Output: 8
print(min(numbers))   # Output: 1
print(sum(numbers))   # Output: 17
print(sorted(numbers))# Output: [1, 3, 5, 8]
```

---

8 Two Ways to Add Elements to a List

Definition:

1. **append()** adds a single element to the end of the list.
2. **extend()** adds multiple elements (an iterable) to the end of the list.
3. **insert()** allows insertion of an element at a specific index.
4. These methods allow modification of the list **in place** without creating a new list.

Example:

```
my_list = [10, 20]
my_list.append(30)      # Result: [10, 20, 30]
my_list.extend([40, 50])  # Result: [10, 20, 30, 40, 50]
```

---

9 What is Indexing and Slicing?

- **Indexing in Python Lists**

  Definition:

  1. **Indexing** is used to access individual elements in a list.
  2. It can be **positive** (starting from 0) or **negative** (starting from -1 for the last element).
  3. Lists support direct **index-based access** using list[index].
  4. Attempting to access an index that does not exist results in an **IndexError**.

  Example:

```
my_list = ["apple", "banana", "cherry"]
print(my_list[0])    # Output: apple (positive index)
print(my_list[-1])   # Output: cherry (negative index)
```

- **Slicing in Python Lists**

  Definition

  1. **Slicing** extracts a portion of a list using [start:end:step].
  2. The **start** index is **included**, but the **end** index is **excluded**.
  3. If start is omitted, it defaults to the **first** element (0).
  4. If end is omitted, it defaults to the **last** element.

  Example:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
print(numbers[2:5])    # Output: [2, 3, 4]  (Elements at indices 2,3,4)
print(numbers[:4])     # Output: [0, 1, 2, 3]  (Start from index 0)
print(numbers[3:])     # Output: [3, 4, 5, 6, 7]  (End at last index)
print(numbers[::2])    # Output: [0, 2, 4, 6]  (Step of 2)
```

  Real-Life Analogy:

  - **Indexing** is like picking a single book from a bookshelf using its position.
  - **Slicing** is like taking a section of books from the shelf (e.g., books from position 3 to 6).

मराठीत एक सोपा टीप (Marathi Tip)

- **List म्हणजे बदलता येणारा डेटा संग्रह!**
- **Indexing आणि Slicing** मधून एखादे घटक मिळवणे म्हणजे मेन्यूमधून पदार्थ निवडणे.
- **Append** म्हणजे नविन वस्तू शेवटी घालणे, **Extend** म्हणजे एकापेक्षा जास्त वस्तू एकत्र घालणे.
- **Mutable म्हणजे बदलता येणारे, Immutable म्हणजे कायमस्वरूपी!**

"Lists मध्ये विविध ऑपरेशन्स सहज करता येतात, त्यामुळे आपला कोड क्लीन आणि कार्यक्षम राहतो. लक्षात ठेवा, List बदलता येण्यामुळे त्यात सुधारणा करणे सोपे जाते!"

List vs Tuples (Most IMP)

Feature	List	Tuple
Definition	A list is a **mutable** ordered collection of elements.	A tuple is an **immutable** ordered collection of elements.
Syntax	Created using **square brackets** `[ ]`	Created using **parentheses** `( )`
Mutability	**Mutable** → Elements can be **added, removed, or changed.**	**Immutable** → Elements **cannot** be changed after creation.
Performance	**Slower** because operations like append/remove take more time.	**Faster** due to its immutability and optimized memory usage.
Usage	Used when the data **needs to be modified frequently.**	Used when the data **should remain constant.**
Memory	**Consumes more memory** due to additional overhead for modifications.	**Consumes less memory** as it's fixed and optimized.
Methods	Supports **many built-in methods** like `append()`, `remove()`, `sort()`, etc.	Supports **fewer methods** like `count()` and `index()`.
Looping Speed	Slower due to dynamic nature.	Faster due to fixed size and immutability.
Example	`list1 = [10, 20, 30]`	`tuple1 = (10, 20, 30)`

3.2 Tuples in Python

(Defination and built in tuple function are IMP , read the rest theory only in 3.2)

1 What is a Tuple?

1. A **tuple** is an **ordered** collection of elements that is **immutable** (cannot be changed after creation).
2. Tuples allow storage of **multiple data types** (e.g., integers, strings, floats) in a single variable.
3. They are defined using **parentheses ()**, with each element separated by a comma.
4. Since tuples are **immutable**, they are faster and more memory-efficient than lists.

Real-Life Analogy:

Think of a tuple like a **passenger train ticket**—once issued, you cannot change the passenger's name or seat number.

Example:

```
my_tuple = (10, "hello", 3.5, True)
print(my_tuple)  # Output: (10, "hello", 3.5, True)
```

---

2 How to Access Elements in a Tuple?

Definition:

1. Elements in a tuple are accessed using **indexing**, starting from 0 for the first element.
2. **Positive indexing** starts from 0 and moves forward.
3. **Negative indexing** starts from -1 (last element) and moves backward.
4. **Slicing** can be used to extract a portion of a tuple using [start:end:step].

Example:

```
my_tuple = (10, 20, 30, 40, 50, 60)
print(my_tuple[1])     # Output: 20 (Positive indexing)
print(my_tuple[-1])    # Output: 60 (Negative indexing)
print(my_tuple[1:4])   # Output: (20, 30, 40) (Slicing)
print(my_tuple[::2])   # Output: (10, 30, 50) (Step slicing)
```

Real-Life Analogy:

Accessing a tuple is like picking a **seat in a movie theater**, where each seat has a fixed number.

---

3 How to Delete Values in a Tuple?

Definition:

1. Tuples are **immutable**, meaning you cannot delete individual elements.
2. However, you can delete an **entire tuple** using del tuple_name.
3. To remove specific elements, you must convert the tuple into a **list**, delete the element, and convert it back.
4. Tuples are often used when data should remain **unchanged** for safety.

Example (Deleting Entire Tuple):

```
my_tuple = (10, 20, 30)
del my_tuple  # The entire tuple is deleted
```

Example (Removing an Element by Converting to List):

```
my_tuple = (10, 20, 30)
temp_list = list(my_tuple)  # Convert to list
temp_list.remove(20)        # Remove element
my_tuple = tuple(temp_list)  # Convert back to tuple
print(my_tuple)  # Output: (10, 30)
```

---

4 How to Update a Tuple?

Definition:

1. Since tuples are **immutable**, you cannot modify them directly.
2. To update a tuple, convert it into a **list**, modify the elements, and convert it back.
3. You can concatenate two tuples to create a **new** updated tuple.
4. Tuples provide safety by preventing accidental modifications.

Example (Using List Conversion):

```
my_tuple = (10, 20, 30)
temp_list = list(my_tuple)  # Convert to list
temp_list[1] = 50  # Change element at index 1
my_tuple = tuple(temp_list)  # Convert back to tuple
print(my_tuple)  # Output: (10, 50, 30)
```

Example (Using Concatenation):

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5)
```

```
new_tuple = tuple1 + tuple2  # Concatenation
print(new_tuple)  # Output: (1, 2, 3, 4, 5)
```

---

5 Six Basic Tuple Operations

Definition:

1. + (Concatenation) - Joins two tuples together.
2. * (Repetition) - Repeats a tuple multiple times.
3. in (Membership) - Checks if an element exists in a tuple.
4. len() - Returns the number of elements in a tuple.
5. **Iteration** - Tuples can be looped using for loops.
6. count() - Counts occurrences of an element in a tuple.

Example:

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5)
# 1. Concatenation
print(tuple1 + tuple2)  # Output: (1, 2, 3, 4, 5)
# 2. Repetition
print(tuple1 * 2)  # Output: (1, 2, 3, 1, 2, 3)
# 3. Membership
print(3 in tuple1)  # Output: True
# 4. Length
print(len(tuple1))  # Output: 3
# 5. Iteration
for item in tuple1:
    print(item)  # Output: 1 2 3
# 6. Count
print(tuple1.count(2))  # Output: 1
```

---

6 Six Methods of Tuples

1. **count()** → Returns the number of times a value appears in a tuple.
2. **index()** → Returns the index of the first occurrence of a value.
3. **sorted()** → Returns a sorted list of tuple elements.
4. **len()** → Returns the total number of elements in the tuple.
5. **min()** → Returns the smallest element in the tuple.
6. **max()** → Returns the largest element in the tuple.

Example:

```
my_tuple = (10, 20, 30, 20, 40)
# count() - Counts occurrences of an element
print(my_tuple.count(20))  # Output: 2
# index() - Returns index of first occurrence
print(my_tuple.index(30))  # Output: 2
# sorted() - Returns a sorted list of tuple elements
print(sorted(my_tuple))  # Output: [10, 20, 20, 30, 40]
# len() - Returns length of tuple
print(len(my_tuple))  # Output: 5
# min() - Returns smallest element
print(min(my_tuple))  # Output: 10
# max() - Returns largest element
print(max(my_tuple))  # Output: 40
```

---

7 Six Built-in Tuple Functions

1. **tuple()** → Converts a list or iterable into a tuple.
2. **len()** → Returns the length of the tuple.
3. **max()** → Returns the maximum element in a tuple.
4. **min()** → Returns the minimum element in a tuple.
5. **sum()** → Returns the sum of all numeric elements in a tuple.
6. **sorted()** → Returns a sorted list of tuple elements.

Example:

```
numbers = (5, 10, 15, 20)
# tuple() - Converts list to tuple
my_list = [1, 2, 3]
print(tuple(my_list))  # Output: (1, 2, 3)
# len() - Returns tuple length
print(len(numbers))  # Output: 4
# max() - Returns largest element
print(max(numbers))  # Output: 20
# min() - Returns smallest element
print(min(numbers))  # Output: 5
# sum() - Returns sum of all elements
print(sum(numbers))  # Output: 50
# sorted() - Sorts elements (returns a list)
print(sorted(numbers))  # Output: [5, 10, 15, 20]
```

मराठीत एक सोपा टीप (Marathi Tip)

- ट्यूपल्स (Tuples) हे लिस्टसारखे असतात पण **Immutable असतात**.
- जर तुम्हाला **ट्यूपल्समधील डेटा बदलायचा असेल**, तर तो **लिस्टमध्ये Convert करा, बदल करा आणि परत ट्यूपलमध्ये Convert करा!**
- **del** keyword वापरून **संपूर्ण** ट्यूपल **delete** करता येतो, पण त्यातील घटक वेगळे काढता येत नाहीत.
- **Multiple data types एकत्र** साठवण्यासाठी ट्यूपल्स खूप उपयोगी आहेत!

3.3 Sets in Python (Most IMP remember all 3.3)

1 What is a Set?

1. A **set** is an **unordered collection** of unique elements (no duplicate values allowed).
2. Sets are defined using **curly braces {}** or the **set() constructor**.
3. They support **mathematical operations** like union, intersection, and difference.
4. Sets are **mutable** (elements can be added or removed) but only contain **immutable** data types (like numbers, strings, and tuples).

Real-Life Analogy:

A **set of unique colors** in a painting (no duplicate colors exist).

Example:

```
my_set = {1, 2, 3, 4, 5}

print(my_set)  # Output: {1, 2, 3, 4, 5}
```

---

2 How to Create a Set?

Definition:

1. A **set** is created using curly brackets {} or the set() function.
2. **Elements in a set must be unique**; duplicates are automatically removed.
3. Sets can store **integers, strings, and tuples**, but not mutable types like lists.
4. Empty {} creates a dictionary, so an empty set must be created using set().

Example: *(Creating a set with 5 elements)*

```
fruits = {"apple", "banana", "cherry", "mango", "grape"}

print(fruits)  # Output: {'banana', 'apple', 'mango', 'cherry', 'grape'}
```

Creating an Empty Set (Correct Way)

```
empty_set = set()

print(type(empty_set))  # Output: <class 'set'>
```

---

3 How to Access Elements in a Set?

Definition:

1. Sets **do not support indexing** because they are unordered.
2. You cannot access elements directly using an index (like in lists or tuples).
3. You can **iterate over a set** using a loop to access elements.
4. You can check for the presence of an element using the in keyword.

Example:

```
my_set = {"car", "bike", "bus", "train"}
for item in my_set:
    print(item)  # Prints each element (order is random)


print("bus" in my_set)  # Output: True (checks if "bus" exists)
```

---

4 How to Update a Set?

Definition:

1. Use add() to insert a **single element** into the set.
2. Use update() to insert **multiple elements** into the set.
3. Duplicate values are automatically removed.
4. Only immutable types can be added to a set.

Example:

```
numbers = {1, 2, 3}
numbers.add(4)
print(numbers)  # Output: {1, 2, 3, 4}


numbers.update([5, 6, 7])
print(numbers)  # Output: {1, 2, 3, 4, 5, 6, 7}
```

---

5 How to Delete Values in a Set?

Definition:

1. Use remove() or discard() to delete a specific element.
2. remove() raises an **error** if the element doesn't exist.
3. discard() does **not** raise an error if the element is missing.
4. Use pop() to remove a **random** element.

Example:

```
my_set = {10, 20, 30, 40}


my_set.remove(20)  # Removes 20
print(my_set)  # Output: {10, 30, 40}


my_set.discard(50)  # Does nothing, as 50 is not in the set
```

---

6 Six Basic Set Operations

1. **Union (union())**: Combines two sets and returns all unique elements.
2. **Intersection (intersection())**: Returns only the common elements between two sets.
3. **Difference (difference())**: Returns elements that exist in one set but not in the other.
4. **Symmetric Difference (symmetric_difference())**: Returns elements that are not present in both sets.
5. **Subset (issubset())**: Checks if one set is a subset of another (all elements of the first set exist in the second).
6. **Superset (issuperset())**: Checks if one set contains all elements of another.

Example:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}


print(A.union(B))  # Output: {1, 2, 3, 4, 5, 6}
print(A.intersection(B))  # Output: {3, 4}
print(A.difference(B))  # Output: {1, 2}
```

---

7 Six Built-in Set Functions

1. **add(value)** – Adds a single element to the set.
2. **remove(value)** – Removes a specific element; raises an error if it does not exist.
3. **discard(value)** – Removes a specific element; does not raise an error if missing.
4. **pop()** – Removes and returns a **random element** from the set.
5. **clear()** – Removes all elements, making the set empty.
6. **copy()** – Returns a shallow copy of the set.

Example:

```
my_set = {10, 20, 30, 40}
my_set.add(50)
print(my_set)  # Output: {10, 20, 30, 40, 50}
my_set.remove(20)
print(my_set)  # Output: {10, 30, 40, 50}
my_set.discard(100)  # No error, even though 100 is not present
random_element = my_set.pop()
print(random_element)  # Output: Random element from the set
my_set.clear()
print(my_set)  # Output: set()
```

मराठीत एक सोपा टीप (Marathi Tip)

- "सेट्स म्हणजे कपाटात **ठेवलेले वेगवेगळे प्रकारचे वस्त्रांचे संच**, जिथे कोणतीही वस्तू **पुन्हा ठेवली जात नाही!**
- सेट म्हणजे **म्युझियम** – प्रत्येक वस्तू **वेगळी असते** आणि एकाच वस्तूची **नक्कल ठेवली जात नाही!**

3.4 Dictionaries in Python (Most IMP remember all 3.4)

1 What is a Dictionary in Python?

1. A **dictionary** is a collection of **key-value pairs**, where each key is unique.
2. It is an **unordered, mutable** data structure used for fast data retrieval.
3. Keys in a dictionary must be **immutable types** (e.g., strings, numbers, tuples).
4. It is defined using {} with keys and values separated by :.

Example:

```
student = {"name": "Rahul", "age": 20, "city": "Mumbai"}
print(student)
# Output: {'name': 'Rahul', 'age': 20, 'city': 'Mumbai'}
```

---

2 How to Create a Dictionary ?

1. Dictionaries are created using **curly brackets {}** with key-value pairs.
2. The keys should be **unique**, and values can be of **any data type**.
3. You can also create a dictionary using the **dict()** constructor.
4. Keys are used to **access values efficiently**.

Example:

```
car = {
  "brand": "Toyota",
  "model": "Camry",
  "year": 2022,
  "color": "White",
  "price": 2500000
}
print(car)
# Output: {'brand': 'Toyota', 'model': 'Camry', 'year': 2022, 'color': 'White', 'price': 2500000}
```

---

3 How to Access Elements in a Dictionary?

1. Elements are accessed using **keys**, not indexes like lists or tuples.
2. The syntax is **dictionary[key]** to get the corresponding value.
3. The **.get(key)** method can also be used to avoid errors if the key does not exist.
4. Using **loops**, all key-value pairs can be accessed.

Example:

```
student = {"name": "Amit", "age": 21, "course": "Python"}
print(student["name"])  # Output: Amit


# Using get() to avoid error
print(student.get("age"))  # Output: 21


# Accessing all keys and values
for key, value in student.items():
   print(key, ":", value)
```

---

4 How to Update a Dictionary?

1. Dictionaries can be updated by assigning a **new value** to an existing key.
2. The .update() method allows adding **multiple key-value pairs**.
3. If the key **exists**, the value is updated. If not, a new key-value pair is added.
4. Dictionaries are **mutable**, so changes are made directly in memory.

Example:

```
student = {"name": "Amit", "age": 21, "course": "Python"}
# Updating a value
student["age"] = 22
print(student)
# Output: {'name': 'Amit', 'age': 22, 'course': 'Python'}
# Adding new key-value pair
student.update({"grade": "A", "city": "Pune"})
print(student)
# Output: {'name': 'Amit', 'age': 22, 'course': 'Python', 'grade': 'A', 'city': 'Pune'}
```

5 How to Delete Values in a Dictionary?

1. The del keyword removes a specific key-value pair.
2. The .pop(key) method removes and **returns** a value.
3. The .popitem() method removes the **last inserted** key-value pair.
4. The .clear() method deletes **all elements** from the dictionary.

Example:

```
student = {"name": "Amit", "age": 22, "course": "Python", "grade": "A"}
# Using del
del student["grade"]
print(student)  # Output: {'name': 'Amit', 'age': 22, 'course': 'Python'}
# Using pop()
removed_value = student.pop("course")
print(removed_value)  # Output: Python
# Using clear()
student.clear()
print(student)  # Output: {}
```

6 Six Basic Dictionary Operations

1. **Adding elements** → dict[key] = value
2. **Updating elements** → dict.update({key: value})
3. **Deleting elements** → del dict[key] or dict.pop(key)
4. **Checking key existence** → "key" in dict
5. **Getting all keys** → dict.keys()
6. **Getting all values** → dict.values()

7 Six Built-in Dictionary Functions

1. **len(dict)** – Returns the number of key-value pairs.
2. **dict.keys()** – Returns all dictionary keys.
3. **dict.values()** – Returns all values in the dictionary.
4. **dict.items()** – Returns all key-value pairs as tuples.
5. **dict.get(key)** – Retrieves a value without error if the key is missing.
6. **dict.copy()** – Creates a copy of the dictionary.

Example:

```
student = {"name": "Amit", "age": 22, "course": "Python"}


print(len(student))  # Output: 3
print(student.keys())  # Output: dict_keys(['name', 'age', 'course'])
print(student.values())  # Output: dict_values(['Amit', 22, 'Python'])
```

---

8 Six Dictionary Methods

1. **clear()** – Removes all dictionary elements.
2. **pop(key)** – Removes a specific key and returns its value.
3. **popitem()** – Removes and returns the **last inserted** key-value pair.
4. **setdefault(key, value)** – Returns the value of a key, or sets a default value if key doesn't exist.
5. **update({key: value})** – Updates the dictionary with new key-value pairs.
6. **copy()** – Returns a **shallow copy** of the dictionary.

Example:

```
student = {"name": "Amit", "age": 22}
student.setdefault("grade", "A")
print(student)
# Output: {'name': 'Amit', 'age': 22, 'grade': 'A'}


student.update({"city": "Pune"})
print(student)
# Output: {'name': 'Amit', 'age': 22, 'grade': 'A', 'city': 'Pune'}
```

मराठीत एक सोपा टीप (Marathi Tip)

- **"Dictionary म्हणजे एक खास स्टोरेज आहे जिथे प्रत्येक माहितीला एक विशिष्ट कळ (Key) दिली जाते. त्यामुळे माहिती पटकन मिळते आणि बदलता पण येते. हे तंत्र सॉफ्टवेअर मध्ये डेटा स्टोअर करण्यासाठी खूप महत्त्वाचे आहे!"**

List vs Dictionary (Most IMP)

Feature	List	Dictionary
Definition	A list is an **ordered collection** of elements that can be accessed using an **index**.	A dictionary is an **unordered collection** of key-value pairs where elements are accessed using a **key**.
Syntax	Created using **square brackets** `[ ]`	Created using **curly braces** `{ }`
Accessing Elements	Elements are accessed using an **index** (e.g., `list1[0]`).	Elements are accessed using a **key** (e.g., `dict1["name"]`).
Mutability	**Mutable** → Elements can be **added, removed, or modified**.	**Mutable** → Values can be **modified**, but **keys must be unique and immutable**.
Ordering	**Maintains order** of elements (since Python 3.7+).	**Maintains order** (since Python 3.7+), but keys are unique.
Duplicates	Allows duplicate values.	**Does not allow duplicate keys**, but values can be duplicated.
Performance	Faster for **sequential access**.	Faster for **searching and retrieving data** using keys.
Usage	Used when the data is a **simple collection of elements**.	Used when the data is **structured as key-value pairs**.
Example	`list1 = [10, 20, 30]`	`dict1 = {"name": "Amit", "age": 20}`

❖ Summer 2022

1. Give two differences between list and tuple. **(2marks)**
2. Explain four Buit-in tuple functions python with example. **(4marks)**
3. Explain indexing and slicing in list with example. **(4marks)**
4. Write a program to create dictionary of students that includes their ROLL NO. and NAME.
   i) Add three students in above dictionary
   ii) Update name = 'Shreyas' of ROLL NO = 2
   iii) Delete information of ROLL NO = 1. **(4marks)**
5. Write the output of the following :
   i) >>> a = [2, 5, 1, 3, 6, 9, 7]
   >>> a [2 : 6] = [2, 4, 9, 0]
   >>> print (a)
   ii) >>> b = ["Hello" , "Good"]
   >>> b. append ("python")
   >>> print (b)
   iii) >>> t1 = [3, 5, 6, 7]
   >>> print (t1 [2])
   >>> print (t1 [–1])
   >>> print (t1 [2 :])
   >>> print (t1 [:]). **(6marks)**

❖ Winter 2022

1. Describe Tuples in Python. **(2marks)**
2. Write any four methods of dictionary.. **(4marks)**
3. Write basis operations of list.. **(4marks)**
4. Compare list and dictionary. (Any 4 points). **(4marks)**
5. Write python program to perform following operations on Set (Instead of Tuple)
   i) Create set
   ii) Access set Element
   iii) Update set
   iv) Delete set. **(6marks)**
6. Explain mutable and immutable data structures. **(6marks)**

❖ Summer 2023

1. Write down the output of the following Python code

   >>>indices=['zero','one','two','three','four','five']

   i) >>>indices[:4]

   ii) >>>indices[:-2]. **(2marks)**

2. Explain creating Dictionary and accessing Dictionary Elements with example. **(4marks)**
3. Write a python program to input any two tuples and interchange the tuple variables.. **(4marks)**
4. Differentiate between list and Tuple. **(4marks)**
5. Write a Python Program to accept values from user in a list and find the largest number and smallest number in a list. **(6marks)**
6. Explain any six set function with example.. **(6marks)**

❖ Winter 2023

1. Explain two ways to add objects / elements to list. **(2marks)**
2. Explain four built-in list functions. **(4marks)**
3. T = ('spam', 'Spam', 'SPAM!', 'SaPm')

   print (T [2])

   print (T [-2])

   print (T [2:])

   print (List (T). **(4marks)**
4. Explain different functions or ways to remove key : value pair from Dictionary **(4marks)**
5. Explain any four set operations with example **(6marks)**
6. List and explain any four built-in functions on set. **(6marks)**

❖ Summer 2024

1. Enlist any four data structures used in python. **(2marks)**
2. Write syntax for a method to sort a list.. **(2marks)**
3. Write python program to perform following operations on set.

   i) Create set of five elements

   ii) Access set elements

   iii) Update set by adding one element

   iv) Remove one element from set.. **(4marks)**
4. What is the output of the following program?

```
dict1 = {'Google' : 1, 'Facebook' : 2, 'Microsoft' : 3}
dict2 = {'GFG' : 1, 'Microsoft' : 2, 'Youtube' : 3}
dict1·update(dict2);
for key, values in dictl·items( ):
        print (key, values).
```

   (4marks)
5. Compare list and tuple (any four points). **(4marks)**
6. Write the output for the following if the variable course = "Python"

```
>>> course [ : 3 ]
>>> course [ 3 : ]
>>> course [ 2 : 2 ]
>>> course [ : ]
>>> course [ -1 ]
>>> course [ 1 ]
```

   (6marks)

❖ Winter 2024

1. Compare list and Tuple.. **(2marks)**
2. Describe any four methods of list in Python. **(4marks)**
3. How to create dictionary in Python ? Write any three methods of dictionary . **(4marks)**
4. Write the output for the following if the variable fruit = "banana".

```
>> fruit [:3]
>> fruit [3:]
>> fruit [3:3]
>> fruit [:].
```

   (4marks)
5. Explain basic operations performed on set with suitable example . **(6marks)**