🧠 What is a List?

Definition:
A list is a **collection of items** in a **single variable**. Lists are **ordered**, **mutable (can be changed)**, and can hold **different data types**.

1️⃣ What is *a* List?

```
# Creating lists for different apps
playlist = ["Shape of You", "Naa Ready", "Believer", "Tum Hi
Ho"]       # Spotify
favourite_foods = ["Pizza", "Burger", "Dosa", "Biryani"]
# Zomato
recent_locations = ["Home", "Airport", "Work", "Mall"]
# Uber

print("Spotify Playlist:", playlist)
print("Zomato Foods:", favourite_foods)
print("Uber Locations:", recent_locations)
```

2️⃣ List Methods

👉 append()

```
playlist.append("Oo Antava")
print("After append:", playlist)
```

👉 insert()

```
favourite_foods.insert(2, "Pani Puri")
print("After insert:", favourite_foods)
```

👉 remove()

```
recent_locations.remove("Mall")
print("After remove:", recent_locations)
```

👉 **pop()**

```
playlist.pop()
print("After pop:", playlist)
```

👉 **sort()**

```
favourite_foods.sort()
print("After sort:", favourite_foods)
```

👉 **reverse()**

```
recent_locations.reverse()
print("After reverse:", recent_locations)
```

👉 **count()**

```
print("Pizza count:", favourite_foods.count("Pizza"))
```

✅ Summary (Slide or Reel End Screen)

Concept	Method/Example
Add	append(), insert()
Remove	remove(), pop()

Access	list[index], list[start:stop]
Loop	for item in list
Check	"item" in list
Sort/Reverse	sort(), reverse()

3️⃣ Slicing

```
# Get top 2 songs
print("Top 2 Songs:", playlist[0:2])

# Get last 2 locations
print("Last 2 Locations:", recent_locations[-2:])
```

4️⃣ Iteration (Looping)

🔁 Print all food items

```
for food in favourite_foods:
    print("Craving for:", food)
```

🔁 Add artist name to each song

```
for song in playlist:
    print(song + " by Arijit")
```

5️⃣ Check if Item Exists

```
if "Dosa" in favourite_foods:
    print("Yes, Dosa is available in Zomato list!")
```

6️⃣ Update List Items (Mutability)

```
# Change "Naa Ready" to "Perfect"
playlist[1] = "Perfect"
print("Updated Playlist:", playlist)

# Change "Burger" to "Shawarma"
favourite_foods[1] = "Shawarma"
print("Updated Foods:", favourite_foods)
```

7️⃣ List with Mixed Data Types

```
order_summary = ["Biryani", 2, 199.50, True]
print("Order Summary (Zomato):", order_summary)
```

🧠 How to Print the Index of Items in a List?

☑️ Method 1: Use `enumerate()` (🔥 Recommended)

```
playlist = ["Shape of You", "Believer", "Tum Hi Ho", "Oo
Antava"]

for index, song in enumerate(playlist):
```

```
    print(f"{index} → {song}")
```

📜 Output:

```
0 → Shape of You
1 → Believer
2 → Tum Hi Ho
3 → Oo Antava
```

✅ Final Note (for video ending slide)

- 🔹 Lists are **powerful** and **flexible**.
- 🔹 They can hold any data: songs, foods, rides, prices, or even mixed info.
- 🔹 Learn them once – use them everywhere in Python apps!

🔹 What is a Tuple?

A **tuple** is just like a **list**, but it's **immutable** – meaning **you can't change** the values after creating it.

📌 Syntax:

=

```
my_tuple = (item1, item2, item3)
```

✅ Ordered
✅ Allows duplicates
❌ Cannot change (no append, remove, update)

🚖 Uber Dataset Example

Let’s say Uber stores each ride's trip summary as a tuple (because the data shouldn’t be changed once the trip is done):

```
trip_summary = ("UberGo", "Chennai", "Airport", 450.50,
"Completed")
print(trip_summary)
```

This stores:

- Ride Type
- Pickup Location
- Drop Location
- Fare
- Status

🔐 Why Tuple and Not List?

🧠 Use Tuples when:

1. Data **shouldn’t be modified** (like historical logs, coordinates, config).
2. Tuples are **faster** than lists for read-only operations.
3. Tuples can be used as **keys in dictionaries** (lists can't!).

🛠️ Basic Operations on Tuples

☑ Access by index

```
print(trip_summary[1])  # Output: Chennai
```

☑ Loop through tuple

```
for item in trip_summary:
    print(item)
```

☑ Length of tuple

```
print(len(trip_summary))  # Output: 5
```

☑ Count and Index

```
print(trip_summary.count("Completed"))  # Output: 1
print(trip_summary.index("Airport"))    # Output: 2
```

❌ Immutable Nature (Can’t Change Values)

trip_summary[1] = "Coimbatore" # ❌ Will throw TypeError

🧠 Output:

```
TypeError: 'tuple' object does not support item assignment
```

✨ Tuple vs List – Summary Table for Slide

Feature	List	Tuple
Syntax	`[1, 2, 3]`	`(1, 2, 3)`
Mutable	✅ Yes	❌ No
Ordered	✅ Yes	✅ Yes
Methods	append(), pop(), etc.	Only count(), index()
Speed	Slower	Faster
Use Case	Dynamic data	Fixed data (like logs)

🎯 Real-World Use Cases of Tuples (App Context)

App	Example Tuple Use Case
Uber	Trip summary: `("UberGo", "Chennai", "Airport", 450.50, "Completed")`
Zomato	Order receipt: `("Pizza", 2, 399.0, "Delivered")`
Spotify	Song info: `("Shape of You", "Ed Sheeran", 4.5)`

🧠 What is a Dictionary?

A **dictionary** is a collection of **key-value pairs**.
It's like a real-world address book:

- Name (Key) → Phone number (Value)
- Booking ID (Key) → Trip Details (Value)

🔑 Keys are unique
🎯 Values can be anything

📌 **Syntax**:

```
my_dict = {
    "key1": "value1",
    "key2": "value2"
}
```

🚖 Uber Dataset Example

Let's say you want to store Uber trip details using a dictionary:

```
trip = {
    "trip_id": "UB12345",
    "pickup": "Chennai Central",
    "drop": "Airport",
    "fare": 430.75,
    "driver": "Ravi",
    "status": "Completed"
}
```

🎯 Keys: `"trip_id"`, `"pickup"`, `"drop"`
📦 Values: `"UB12345"`, `"Chennai Central"`, etc.

🔍 Accessing Dictionary Values

```
print(trip["pickup"])    # Output: Chennai Central
print(trip["fare"])      # Output: 430.75
```

✅ You access the value by using the **key**.

🛠️ Dictionary Methods (Very Important)

1. `get()` – Safely access a key

```
print(trip.get("driver"))          # Output: Ravi
print(trip.get("car_model"))       # Output: None (no error!)
```

2. `keys()` – All keys

```
print(trip.keys())
# Output: dict_keys(['trip_id', 'pickup', 'drop', 'fare',
'driver', 'status'])
```

3. `values()` – All values

```
print(trip.values())
```

4. `items()` – All key-value pairs

```
for key, value in trip.items():
    print(key, ":", value)
```

5. `update()` – Change or add new key-value

```
trip.update({"car_model": "Suzuki"})
print(trip)
```

6. `pop()` – Remove a key

```
trip.pop("status")
print(trip)
```

📝 Update Value

```
trip["fare"] = 500.00
print("Updated Fare:", trip["fare"])
```

❌ Key Not Found Error (Avoid it using `get()`)

print(trip["rating"]) # ❌ KeyError if not present

🔁 Looping Through Dictionary

```
for key in trip:
    print(key, "→", trip[key])
```

Or with `items()`:

```
for key, value in trip.items():
    print(f"{key}: {value}")
```

✅ When to Use Dictionary?

✅ Use Dictionary When...

You want to store **related data as pairs**

You want to **search using a unique key**

You want fast **lookups and updates**

💡 Real-World Examples

App	Dictionary Use Case
Uber	Trip details (as shown)
Zomato	Order: `{"dish": "Pizza", "qty": 2, "price": 399}`
Spotify	Song: `{"title": "Shape of You", "artist": "Ed Sheeran", "duration": 4.5}`

🎯 Summary Table (Use as Slide or Reel Recap)

Feature	Description
Access	`dict["key"]`, `get()`

Add/Update	`dict["key"] = value`
Remove	`pop("key")`
Loop	`for key in dict:` or `for key, value in dict.items()`
Useful for	Structured info like trip/order/song etc.

❓ Does a Python `dict` Allow Duplicates?

❌ No — Dictionaries do NOT allow duplicate keys.

- Each **key** in a dictionary must be **unique**.
- If you try to add a duplicate key, the **latest value will overwrite the previous one**.

☑️ Example:

```
trip = {
    "pickup": "Chennai Central",
    "drop": "Airport",
    "pickup": "Tambaram"  # duplicate key
}

print(trip)
```

🧠 Output:

```
{'pickup': 'Tambaram', 'drop': 'Airport'}
```

➡️ The original `"pickup": "Chennai Central"` is overwritten by `"pickup": "Tambaram"`

🎯 But… Can Values Be Duplicated?

✅ **Yes!** Values **can be duplicates. No restriction.**
Only keys must be unique.

```
trip = {
    "pickup": "Chennai",
    "drop": "Chennai",       # duplicate value is allowed
    "driver": "Ravi"
}

print(trip)
```

🧠 Output:

```
{'pickup': 'Chennai', 'drop': 'Chennai', 'driver': 'Ravi'}
```

📌 Summary Slide (Perfect for YouTube)

Dict Part	Duplicate Allowed?	Notes
Keys	❌ No	Duplicate keys are overwritten

Values	✅ Yes	Any value can be repeated

❓ Can a Dictionary Key Hold Multiple Values in Python?

☑ YES — A key can hold multiple values, if the value is a list, tuple, or another collection.

But remember:
🔑 **Key = must be unique**
📦 **Value = can be anything**, even a list or tuple

🤓 Example: Uber Trip with Multiple Drop Points

```
trip = {
    "trip_id": "UB12345",
    "pickup": "Chennai",
    "drop": ["Airport", "Tambaram", "Perungudi"],  # multiple drop points
    "fare": 500
}
```

Here:

- `"drop"` key holds **a list of 3 locations**

🔍 How to Access Those Values?

```
# Access full list of drop locations
print(trip["drop"])
```

```
# Output: ['Airport', 'Tambaram', 'Perungudi']

# Access individual drop locations
print(trip["drop"][0])  # Airport
print(trip["drop"][1])  # Tambaram
```

🔁 Loop Through Multiple Values in a Key

```
for location in trip["drop"]:
    print("Dropping at:", location)
```

📜 Output:

```
Dropping at: Airport
Dropping at: Tambaram
Dropping at: Perungudi
```

☑️ Summary Table for Your YouTube Slide

Concept	Example
Multiple values for a key	Use a list or tuple
Access full values	`dict["key"]`
Access one value	`dict["key"][index]`
Loop through	`for x in dict["key"]:`

🎯 Real-World App Use Cases

App	Dictionary Key with Multiple Values
Uber	`"drop": ["Airport", "Tambaram"]`
Zomato	`"order_items": ["Pizza", "Dosa", "Coke"]`
Spotify	`"artists": ["Arijit Singh", "Sid Sriram"]`

🧠 What Do You Mean by “Multiple Dictionaries”?

You’re working with a **list of dictionaries** or **dictionary of dictionaries**.

☑️ Case 1: List of Dictionaries

(Each dictionary is one record — like multiple Uber trips)

```
trips = [
    {"trip_id": "UB001", "pickup": "Chennai", "drop": "Airport",
"fare": 430},
    {"trip_id": "UB002", "pickup": "Tambaram", "drop":
"Central", "fare": 320},
    {"trip_id": "UB003", "pickup": "T-Nagar", "drop":
"Velachery", "fare": 210}
]
```

🔍 Access Each Dictionary (Record)

```
for trip in trips:
    print("Trip ID:", trip["trip_id"])
    print("From:", trip["pickup"], "→", trip["drop"])
    print("Fare:", trip["fare"])
    print("--------")
```

☑ Case 2: Dictionary of Dictionaries

(Trip ID as key, entire record as value)

```
trip_data = {
    "UB001": {"pickup": "Chennai", "drop": "Airport", "fare":
430},
    "UB002": {"pickup": "Tambaram", "drop": "Central", "fare":
320},
    "UB003": {"pickup": "T-Nagar", "drop": "Velachery", "fare":
210}
}
```

🔍 Lookup and Loop:

```
# Lookup a specific trip using trip ID
print("UB001 Fare:", trip_data["UB001"]["fare"])

# Loop through all trips
for trip_id, details in trip_data.items():
    print("Trip:", trip_id)
    print("From:", details["pickup"], "→", details["drop"])
    print("Fare:", details["fare"])
    print("--------")
```

🎯 Use Case Comparison Table

Structure	When to Use	Example
List of Dicts	When order matters or duplicate keys allowed	Multiple trip records
Dict of Dicts	When you want to access by ID directly (fast lookup)	Trip ID → Details

---

💡 Real-World App Analogy

App	Example Use
Uber	All trip records (`list of dicts`) or Trip ID-wise data (`dict of dicts`)
Zomato	All orders by users
Spotify	Multiple playlists with song lists inside

🧠 What is a Set in Python?

A **set** is an **unordered collection** of **unique** elements.

- 🔹 It automatically removes duplicates
- 🔹 You can do cool math-like operations: union, intersection, etc.

📌 Syntax:

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

Or from a list:

```
my_set = set([1, 2, 2, 3])
```

🚖 Uber Dataset Example

Let's say you want to find **all cities** a user has taken trips to:

```
uber_cities = ["Chennai", "Bangalore", "Chennai", "Delhi",
"Bangalore"]
```

Now remove duplicates using a set:

```
unique_cities = set(uber_cities)
print(unique_cities)
```

📄 Output:

```
{'Delhi', 'Chennai', 'Bangalore'}
```

🎯 **Automatically removed duplicates!**

✅ Set Properties

Property	Value
Ordered?	❌ No
Duplicates ?	❌ No

Mutable?	✅ Yes
Indexing?	❌ No

🛠️ Set Methods & Operations

Let's say:

```
uber_user_1 = {"Chennai", "Mumbai", "Bangalore"}
uber_user_2 = {"Bangalore", "Delhi", "Hyderabad"}
```

1️⃣ `union()` – Combine all (no duplicates)

```
print(uber_user_1.union(uber_user_2))
# or uber_user_1 | uber_user_2
```

📜 Output:

```
{'Delhi', 'Mumbai', 'Hyderabad', 'Bangalore', 'Chennai'}
```

2️⃣ `intersection()` – Common cities

```
print(uber_user_1.intersection(uber_user_2))
# or uber_user_1 & uber_user_2
```

📜 Output:

```
{'Bangalore'}
```

3️⃣ `difference()` – User 1 cities not in User 2

```
print(uber_user_1.difference(uber_user_2))
# or uber_user_1 - uber_user_2
```

📄 Output:

```
{'Mumbai', 'Chennai'}
```

4️⃣ `add()` – Add a new city

```
uber_user_1.add("Coimbatore")
```

5️⃣ `remove()` – Remove a city

```
uber_user_1.remove("Mumbai")
```

🎯 Where Sets Are Used in Real Apps?

App	Example Use Case
Uber	Unique cities a user has visited
Zomato	Unique cuisines ordered

Spotify	Unique artists in liked songs
Instagram	Unique users who liked your post

❌ What Sets Can't Do

```
my_set = {"Chennai", "Delhi"}
print(my_set[0]) # ❌ Error: 'set' object is not subscriptable
```

✅ Use a `list` if you need ordered/indexed data.

📌 Summary for YouTube Slide

Operation	Method	Symbol
Union	`set1.union(set2)`	`
Intersection	`set1.intersection(set2)`	&
Difference	`set1.difference(set2)`	-
Add item	`add()`	
Remove item	`remove()`	

🔄 Can You "Update a Set in the Middle"?

❌ No direct way — because sets are unordered and do not support indexing.

So you **can't do this**:

```
my_set = {1, 2, 3}
my_set[1] = 99  # ❌ Error: 'set' object does not support item assignment
```

☑ But You Can *Remove* + *Add* (Workaround)

Since sets don't care about position, you can:

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

my_set.remove(2)   # Remove the value 2
my_set.add(99)     # Add a new value
print(my_set)      # Output: {1, 3, 99}
```

💡 This is how we "update" an element in a set.

🧠 Real-Life Example (Uber Cities)

```
uber_cities = {"Chennai", "Bangalore", "Mumbai"}

# Let's say user moved from Bangalore to Hyderabad
```

```
uber_cities.remove("Bangalore")
uber_cities.add("Hyderabad")

print(uber_cities)
# Output (unordered): {'Hyderabad', 'Mumbai', 'Chennai'}
```

✅ We “updated” Bangalore to Hyderabad — but not at any specific index (because sets have no index).

🧯 Bonus Tip: Safe Remove with `discard()`

If you're not sure the value exists:

```
uber_cities.discard("Pune")  # Won’t raise error if "Pune" not in set
```

About the Author

Gowtham SB is a **Data Engineering expert, educator, and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

Socials

📹 **YouTube** - https://www.youtube.com/@dataengineeringvideos

📸 **Instagram** - https://instagram.com/dataengineeringtamil

📸 **Instagram** - https://instagram.com/thedatatech.in

🤝 **Connect for 1:1** - https://topmate.io/dataengineering/

💼 **LinkedIn** - https://www.linkedin.com/in/sbgowtham/

🌐 **Website** - https://codewithgowtham.blogspot.com

💻 **GitHub** - http://github.com/Gowthamdataengineer

💬 **Whats App** - https://lnkd.in/g5JrHw8q

📧 **Email** - atozknowledge.com@gmail.com

📱 **All My Socials** - https://lnkd.in/gf8k3aCH