1. **What a string is**
2. **Indexing (accessing characters by position)**
3. **Slicing (taking parts of a string)**

🔹 1. What is a String?

- In Python, a **string** is just text written inside **quotes** (`"  "` or `'  '`).

Example:

```
 chai_type = "Ginger Chai"
customer_name = "Priya"
```

-
- Anything inside quotes is considered a string: `"chai"`, `"123"`, `"Hello"`.

💡 Strings are **immutable** → once created, they cannot be changed. If you "modify" them, Python actually creates a new string in memory.

🔹 2. Using Strings

You can combine strings with variables using **formatted strings (f-strings)**:

```
chai_type = "Ginger Chai"
customer_name = "Priya"

print(f"Order for {customer_name}, {chai_type} please!")
```

👉 Output:

```
Order for Priya, Ginger Chai please!
```

This shows how you can **insert variables into text**.

🔹 3. Indexing (positions in a string)

- Each character in a string has a **position number (index)**.
- Index starts from **0**, not 1.

Example:

```
description = "aromatic and bold"
print(description[0])   # 'a'  (1st letter)
print(description[1])   # 'r'  (2nd letter)
print(description[-1])  # 'd'  (last letter, using negative index)
```

🔹 4. Slicing (taking parts of a string)

You can extract substrings with the format:

```
string[start:end:step]
```

- `start` → where to begin
- `end` → stop before this index (not included)
- `step` → jump size (optional)

Example:

```
description = "aromatic and bold"

print(description[0:8])     # "aromatic"  (0 to 7)
print(description[12:])     # "bold" (from index 12 to end)
print(description[::2])     # every 2nd character → "aomtcn old"
print(description[::-1])    # reverse string → "dlob dna citamora"
```

⚡ Key point: **last index is not included** → `0:8` gives indices `0-7`.

🔹 5. Tricks with Slicing

- `[::-1]` → reverses a string.
- `[:end]` → from start to `end-1`.
- `[start:]` → from start index to the end.
- `string[::step]` → skips characters.

📝 Summary:

- Strings = text in quotes.
- They are immutable.
- **Indexing** lets you pick a single character (`text[0]`).
- **Slicing** lets you pick a range (`text[2:5]`).
- `[::-1]` is a shortcut for reversing.

✨ Example combining all:

```
chai_type = "Ginger Chai"
description = "aromatic and bold"

print(chai_type.upper())          # "GINGER CHAI"
print(description[0:8])           # "aromatic"
print(description[-4:])           # "bold"
print(description[::-1])          # "dlob dna citamora"
```

🔹 1. What is Encoding?

- **Encoding** is the process of converting characters (letters, emojis, symbols) into **numbers (binary 0s and 1s)** so that a computer can store and process them.

- Computers only understand binary (0s & 1s).

- So `"chai"` → gets converted into numbers (using rules of an encoding system).

Example:

- `"A"` in **ASCII** → 65

- `"a"` in **ASCII** → 97

So inside memory, `"A"` is actually stored as `01000001`.

🔹 2. ASCII vs Unicode

- **ASCII** (old system): could only represent **128 characters** (English letters, digits, punctuation).
  → Not enough for other languages like Hindi (चाय), Chinese (茶), Arabic, emojis (☕).

- **Unicode** (modern system): goal is to represent **all characters in all languages**.
  Example:

  - `"च"` (cha in Hindi) → Unicode code point `U+091A`

  - `"茶"` (tea in Chinese) → Unicode code point `U+8336`

So Unicode = a giant universal mapping table for all human languages + symbols.

🔹 3. What is UTF-8?

- Unicode defines **code points** (like “this symbol = U+091A”).
- But how do we store those code points in memory? That’s where **encodings** come in.

👉 UTF-8 is one **encoding scheme** for Unicode.

- It uses **1 to 4 bytes** per character:
  - English letters → 1 byte
  - European letters with accents → 2 bytes
  - Asian languages (Chinese, Hindi, etc.) → 3 bytes
  - Rare symbols/emojis → 4 bytes

That’s why UTF-8 is **efficient**: English text stays small, but it can still handle every character in the world.

🔹 4. Why is UTF-8 Necessary?

- Without encoding, `"चाय"` or `"茶"` would be meaningless to a computer.
- If you open a file with the wrong encoding, you’ll see **garbage characters** (like `Ã¤` instead of `ä`).
- UTF-8 is the **default standard** today because:
  - It supports all languages.
  - It’s backward-compatible with ASCII.
  - It avoids storage waste (English stays light).

🔹 5. Example in Python

```
text = "चाय"   # "Chai" in Hindi
```

```
# Encode string into bytes (UTF-8)
encoded = text.encode("utf-8")
print(encoded)
# b'\xe0\xa4\x9a\xe0\xa4\xbe\xe0\xa4\xaf'

# Decode back into string
decoded = encoded.decode("utf-8")
print(decoded)
# चाय
```

So:

- `.encode("utf-8")` → string → bytes
- `.decode("utf-8")` → bytes → string

✅ **In simple words**:

- Encoding = language → numbers (so computers understand).
- Unicode = dictionary of all world characters.
- UTF-8 = most common way to actually store Unicode in memory/files.
- Without it → we couldn't reliably share text across languages & platforms.