📌 1. What is a String in C?

- A **string** in C is **a sequence of characters stored in a character array**.
- It is **terminated by a special null character** `'\0'` (ASCII value 0) to mark the end.
- C does **not** have a built-in string type — strings are just arrays of `char`.

📌 2. Declaring Strings

Two main ways:

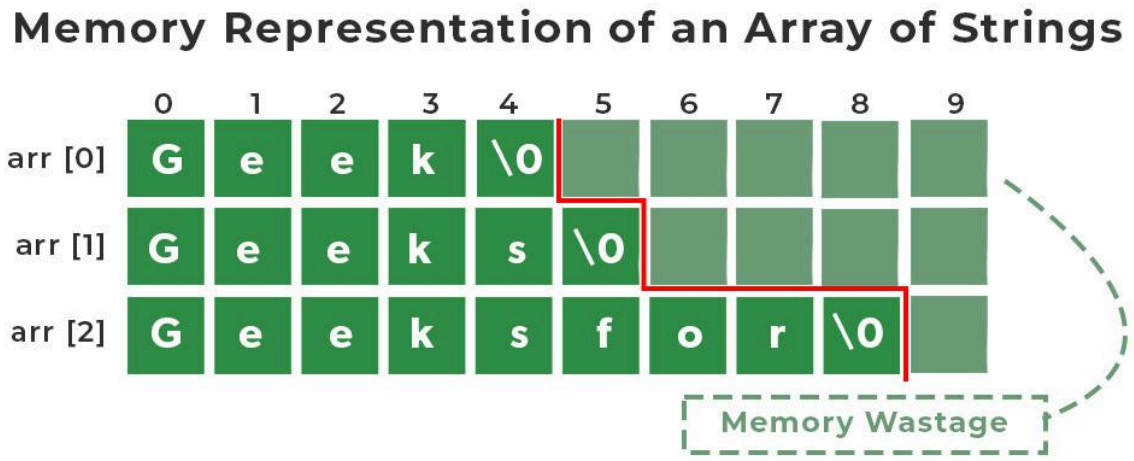
```
// Method 1: Character array
char name[10] = "Maxim";   // Automatically adds '\0'

// Method 2: String literal (pointer to const char)
char *name = "Maixm”;      // Stored in read-only memory
```

Note: `"Maxim"` in memory actually looks like:
`'M' 'a' 'x' ‘i’ '\0'`

📌 3. Initializing Strings

```
char str1[] = "Hello";          // '\0' added automatically
char str2[6] = {'H','e','l','l','o','\0'}; // manual
```

If you don’t add `'\0'`, it will not behave like a proper C string.



📌 4. Input & Output with Strings

```
#include <stdio.h>

int main() {
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);      // stops at space
    printf("Hello %s", name);
}
```

⚠ **Warning:** `scanf("%s",...)` stops at spaces. For full lines use:

```
fgets(name, sizeof(name), stdin); // reads spaces too
```

📌 5. Common String Library Functions

(from `<string.h>`)

Function	Purpose
`strlen(s)`	Length of string (without `'\0'`)
`strcpy(dest, src)`	Copy string

`strcat(dest, src)`	Append strings
`strcmp(s1, s2)`	Compare strings (`0` if equal)
`strchr(s, ch)`	Find first occurrence of char
`strstr(s, sub)`	Find substring

Example:

```
#include <stdio.h>
#include <string.h>

int main() {
    char s1[20] = "Hello";
    char s2[] = "World";
    strcat(s1, s2);       // s1 becomes "HelloWorld"
    printf("%s", s1);
}
```

📌 6. Strings vs Characters

- `'A'` → character (type `char`)
- `"A"` → string (array of chars with `'\0'`)

📌 7. Important Points

- Always leave space for the null character `'\0'` in arrays.
- String literals (`"Hello"`) are stored in **read-only memory** if declared as `char *`.
- Modifying a string literal is **undefined behavior**:

```
char *str = "Hello";
// str[0] = 'Y';  // ❌ Wrong: crash or unexpected behavior
```

- Use character arrays for modifiable strings.

📊 Strings in C vs Python

Feature	C	Python
Type	No built-in string type — strings are arrays of `char` terminated by `'\0'`	Built-in `str` type
Storage	Stored as contiguous memory of characters + null terminator	Stored as immutable Unicode objects
Declaration	`char name[] = "Ali";` or `char *name = "Ali";`	`name = "Ali"`
Mutability	Character arrays are mutable, string literals are not	Strings are immutable — you can't change them in place
Length	Calculated using `strlen(str)` from `<string.h>`	Calculated using `len(str)`
Indexing	`name[0]` returns `char`	`name[0]` returns a new string of length 1
Concatenation	Use `strcat(dest, src)` or manual loops	Use `+` operator or `join()`
Comparison	Use `strcmp(s1, s2)`	Use `==` directly
Input	`scanf("%s", str)` (no spaces) or `fgets(str, size, stdin)`	`input()` function
Null Terminator	Required at end (`'\0'`)	Not required — Python manages string boundaries automatically
Encoding	Plain ASCII or bytes (Unicode requires extra handling)	Unicode by default (UTF-8)
Library Functions	`<string.h>`: `strlen`, `strcpy`, `strcmp`, etc.	Built-in methods: `.upper()`, `.lower()`, `.find()`, `.replace()`, etc.
Memory Safety	Risk of overflow if buffer is too small	Automatic memory management, no overflow risk for normal use