4 September Notes

Strings in Python

- Strings are **sequences of characters** enclosed in ' ', " ", or ''' '''.
- Strings are **immutable** → cannot be modified after creation.
- Supports **indexing, slicing, concatenation, repetition**.

```
s1 = "Hello"
s2 = 'World'
s3 = '''This is
a multi-line string'''
print(s1, s2)
print(s3)
```

🔹 **Indexing & Slicing**

Accessing individual elements of a sequence (like string, list, or tuple) using their position.

Index values start from 0 for the first element.

Python also supports negative indexing:

- -1 → last element
- -2 → second last element, and so on.

```
text = "Python Programming"
print(text[0])     # First char
print(text[-1])    # Last char
print(text[0:6])   # "Python"
print(text[7:])    # "Programming"
```

```
print(text[::2])    # Every 2nd char
```

🔹 Common Methods

```
s = "  Python Basics  "
print(s.lower())
print(s.upper())
print(s.strip())
print(s.replace("Python", "Java"))

words = "apple,banana,grape".split(",")
print(words)
joined = "-".join(words)
print(joined)
```

🔹 String Formatting

Why String Formatting?

- To display values (variables, numbers, strings) neatly inside text.
- Helps create readable and well-structured output.

```
name, age = "Alice", 20

print("Name: %s, Age: %d" % (name, age))       # Old style
print("Name: {}, Age: {}".format(name, age))    # format()
print(f"Name: {name}, Age: {age}")           # f-string (best)
```

1. Using % Operator (Old Style)

- Placeholders:

- %s → string
- %d → integer
- %f → float

✅ **Example:**

```
name = "Alice"
age = 25
marks = 88.5

print("Name: %s, Age: %d, Marks: %.2f" % (name, age, marks))
# Output: Name: Alice, Age: 25, Marks: 88.50
```

🔹 **2. Using str.format() Method**

- Curly braces {} are replaced with values.
- Can use **positional** or **keyword arguments**.

✅ **Example:**

```
name = "Bob"
age = 30

print("My name is {}, and I am {} years old.".format(name, age))
print("My name is {0}, and I am {1} years old.".format(name, age))
print("My name is {n}, and I am {a} years old.".format(n=name, a=age))
```

🔹 **3. Using f-Strings (Python 3.6+)**

- Prefix string with f and directly use variables inside {}.
- Most **modern and preferred** way.

✅ **Example:**

```
name = "Charlie"
```

```
age = 22
marks = 91.2356

print(f"Name: {name}, Age: {age}, Marks: {marks:.2f}")
# Output: Name: Charlie, Age: 22, Marks: 91.24
```

🔹 **4. Number Formatting with f-Strings**

```
pi = 3.14159265358979
print(f"Pi value: {pi:.2f}")   # 2 decimal places → 3.14
print(f"Pi value: {pi:.4f}")   # 4 decimal places → 3.1416

num = 12345
print(f"Number with commas: {num:,}")   # 12,345
print(f"Binary: {num:b}, Hex: {num:x}, Octal: {num:o}")
```

🔹 **5. Alignment in Formatting**

- :<10 → Left aligned (width 10)
- :>10 → Right aligned (width 10)
- :^10 → Center aligned (width 10)

✅ **Example:**

```
print(f"{'Python':<10} is left aligned")
print(f"{'Python':>10} is right aligned")
print(f"{'Python':^10} is center aligned")
```