**Day 4 Session 3: Complete Python Strings Learning Guide**

**What is a String?**

A **string** is simply text - any collection of letters, numbers, symbols, or spaces that you want to treat as text rather than as a mathematical value or command.

Think of strings as anything you might write in a sentence, like:

* Names: "Alice", "John"
* Messages: "Hello World"
* Addresses: "123 Main Street"
* Even numbers when treated as text: "42", "3.14"

**1. Strings Introduction**

In Python, you create strings by wrapping text in **quotes**. Python accepts both single quotes and double quotes:

# Using double quotes

name = "Alice"

message = "Hello World"

# Using single quotes

name = 'Alice'

message = 'Hello World'

# Both work exactly the same way

print(name) # Output: Alice

print(message) # Output: Hello World

**Why Use Quotes?**

Quotes tell Python "treat this as text, not as code." Without quotes, Python thinks you're referring to a variable:

# Without quotes - Python looks for a variable named Alice

name = Alice # This causes an error if Alice isn't defined

# With quotes - Python treats it as text

name = "Alice" # This works perfectly

**#Practices**

# a simple string stored in a variable 'text'

text = "Hello Universe"

# define the 'text' in another variable name 'upper\_text'

# print the text message in upper case

upper\_text = text.upper()

print (upper\_text)

HELLO UNIVERSE

# define the 'text' in another variable name 'lower\_text'

# print the text message in lower case

lower\_text = text.lower()

print (lower\_text)

hello universe

# let again define three variables

myvar1 = "a"

myvar2 = 2

myvar3 = "2"

# print myvar1 type

type(myvar1)

str

# print myvar2 type

type(myvar2)

int

# print myvar3 type

type(myvar3)

str

# first define a variable and take an input and print that variable

myvar4 = input("Enter your choice ")

myvar4

'4'

# the ourput has come as a string

# now if i do addition any number with the myvar4, it will show an error

# before adding i have to convert the string as an integer or as a float

myvar5 = int(myvar4) + 2

myvar5

6

# i can again convert the final output 'myvar5' as a string and print the same

myvar6 = str(myvar5)

myvar6

'6'

# now we can add another string with the string 'myvar6'

# and the output will be a string

myvar7 = myvar6 + " number of seasons"

myvar7

'6 number of seasons'

# if we use print function, we also get the same same output

print (myvar7)

6 number of seasons

**2. Print Function**

The print() function displays strings (and other data) on the screen:

# Basic printing

print("Hello World") # Output: Hello World

print('Python is awesome') # Output: Python is awesome

# Printing variables

name = "Sarah"

print(name) # Output: Sarah

# Printing multiple items

age = 25

print("My name is", name, "and I am", age, "years old")

# Output: My name is Sarah and I am 25 years old

# Printing with separator

print("apple", "banana", "cherry", sep=", ")

# Output: apple, banana, cherry

# Printing without newline

print("Hello", end=" ")

print("World")

# Output: Hello World

**#Practices**

# a function is a code where i can pass some parameters

# the print() function is an example

# here in jupyter we can print any operation as we have seen above

# but in real code we need to print any parameter through print() function

# while we print any string by print() function it will print it

# if i want to print the spaces, commas, questions marks will be same as the input string in the output

print("hello! my name is prasenjit, i am from west bengal")

hello! my name is prasenjit, i am from west bengal

**3. Multiple Line String**

For longer text that spans multiple lines, use triple quotes:

# Using triple double quotes

long\_text = """This is a long string

that spans multiple lines.

You can write as much as you want

without worrying about line breaks."""

print(long\_text)

# Output:

# This is a long string

# that spans multiple lines.

# You can write as much as you want

# without worrying about line breaks.

# Using triple single quotes

poem = '''Roses are red,

Violets are blue,

Python is awesome,

And so are you!'''

print(poem)

# Output:

# Roses are red,

# Violets are blue,

# Python is awesome,

# And so are you!

# Alternative method using \n (newline character)

multiline = "Line 1\nLine 2\nLine 3"

print(multiline)

# Output:

# Line 1

# Line 2

# Line 3

**#Practices**

# now multiple line string

# if i want to print multiple lines, there are two different ways

# first one by using 'new line' operator

# single 'new line' means next line, multiple times 'new lines' operator means multiple empty new lines/paragraphs

# here i will use two 'new line' operatorstwo times d ouble 'new line

print("hello! what are you doing?\n\nhi! I am studying")

hello! what are you doing?

hi! I am studying

# we will use triple quotes for multiline string

print("""hello! what is your name? what are you doing?

hi! myname is prasenjit boral

i am studying""")

hello! what is your name? what are you doing?

hi! myname is prasenjit boral

i am studying

# Using \n for more new lines

print("Name: Prasenjit Boral\nRole: Project Manager\nCompany: Precision Electronics Limited")

Name: Prasenjit Boral

Role: Project Manager

Company: Precision Electronics Limited

# using more example on triple quotes (""" or ''') for multiline strings

# it is best for printing block text or long paragraphs

print("""Dear Team,

Please find the status update below:

- Task A: Completed

- Task B: In Progress

- Task C: Pending

Regards,

Prasenjit""")

Dear Team,

Please find the status update below:

- Task A: Completed

- Task B: In Progress

- Task C: Pending

Regards,

Prasenjit

# joining multiple strings e.g., with +

# it is good when variables are involved

name = "Prasenjit"

project = "Python Brushup"

print("Hello " + name + ",\nWelcome to the " + project + " program.")

Hello Prasenjit,

Welcome to the Python Brushup program.

# joining multiple strings e.g., with f-strings

# it is recommended in advance level

print(f"Hello {name},\nWelcome to the {project} program.")

Hello Prasenjit,

Welcome to the Python Brushup program.

**4. Get Character by Index**

You can access individual characters in a string using their position (index). Python starts counting from 0:

word = "Python"

# 012345 (index positions)

# Accessing individual characters

print(word[0]) # Output: P (first character)

print(word[1]) # Output: y (second character)

print(word[2]) # Output: t (third character)

print(word[5]) # Output: n (last character)

# Real-world example

name = "Alice"

first\_letter = name[0]

print("Your name starts with:", first\_letter) # Output: Your name starts with: A

# Getting the last character

email = "user@gmail.com"

last\_char = email[len(email) - 1] # or email[-1]

print("Last character:", last\_char) # Output: Last character: m

**#Practices**

# in the backend, strings in python are stored as sequences of Unicode characters

# once a string is created, it cannot be changed.

# any modification (like s += "new") creates a new string object in memory.

# python3 uses Unicode (UTF-8/UTF-16/UTF-32) to represent characters.

# internally, python uses an efficient strategy called "Flexible String Representation" (PEP 393).

# # strings are stored as a sequence of characters indexed from 0 to (length of the variable - 1)

myvar8 = "I like classes in Bengali"

myvar8 [0]

'I'

myvar8 [1]

' '

myvar8 [2]

'l'

# printing all the characters in myvar8

for i in range(len(myvar8)):

print(f"Index {i}: {myvar8[i]}")

Index 0: I

Index 1:

Index 2: l

Index 3: i

Index 4: k

Index 5: e

Index 6:

Index 7: c

Index 8: l

Index 9: a

Index 10: s

Index 11: s

Index 12: e

Index 13: s

Index 14:

Index 15: i

Index 16: n

Index 17:

Index 18: B

Index 19: e

Index 20: n

Index 21: g

Index 22: a

Index 23: l

Index 24: i

| **Code** | **Output** | **Reason** |
| --- | --- | --- |
| myvar[0] | 'I' | First character |
| myvar[1] | ' ' | Space after 'I' |
| myvar[2] | 'l' | Start of "like" |
| myvar[3] | 'i' | Second letter of "like" |
| myvar[4] | 'k' | Third letter of "like" |
| myvar[5] | 'e' | Last letter of "like" |
| myvar[6] | ' ' | Space before "classes" |
| myvar[7] | 'c' | First letter of "classes" |
| myvar[8] | 'l' | Second letter of "classes" |
| myvar[9] | 'a' | Third letter of "classes" |
| myvar[10] | 's' | Fourth letter of "classes" |
| myvar[11] | 's' | Fifth letter of "classes" |
| myvar[12] | 'e' | Sixth letter of "classes" |
| myvar[13] | 's' | Last letter of "classes" |
| myvar[14] | ' ' | Space before "in" |
| myvar[15] | 'i' | First letter of "in" |
| myvar[16] | 'n' | Second letter of "in" |
| myvar[17] | ' ' | Space before "Bengali" |
| myvar[18] | 'B' | First letter of "Bengali" |
| myvar[19] | 'e' | Second letter of "Bengali" |
| myvar[20] | 'n' | Third letter of "Bengali" |
| myvar[21] | 'g' | Fourth letter of "Bengali" |
| myvar[22] | 'a' | Fifth letter of "Bengali" |
| myvar[23] | 'l' | Sixth letter of "Bengali" |
| myvar[24] | 'i' | Last letter of "Bengali" |

**5. String Slicing**

String slicing lets you extract portions of a string using the format string[start:end]:

text = "Hello World"

# 01234567890 (index positions)

# Basic slicing

print(text[0:5]) # Output: Hello (characters 0 to 4)

print(text[6:11]) # Output: World (characters 6 to 10)

print(text[0:5]) # Output: Hello

# Slicing shortcuts

print(text[:5]) # Output: Hello (from start to index 4)

print(text[6:]) # Output: World (from index 6 to end)

print(text[:]) # Output: Hello World (entire string)

# Step slicing

print(text[::2]) # Output: HloWrd (every 2nd character)

print(text[1::2]) # Output: el ol (every 2nd character starting from index 1)

# Practical examples

sentence = "Python Programming"

first\_word = sentence[:6] # Output: Python

second\_word = sentence[7:] # Output: Programming

print(f"First word: {first\_word}")

print(f"Second word: {second\_word}")

# Extracting file extension

filename = "document.pdf"

extension = filename[-3:] # Output: pdf

print(f"File extension: {extension}")

**#Practices**

# string slicing

# string slicing allows us to extract a part of the string (a substring) using the syntax "string[start : stop]"

myvar9 = "This is Python"

myvar9 [0:3]

'Thi'

myvar8 [0:4]

'This'

myvar9 [0:8]

'This is '

myvar9 [5:10]

'is Py'

**Example String:**

myvar = "I like classes in Bengali"

**Indexing for reference:**

| **Character** | **I** |  | **l** | **i** | **k** | **e** |  | **c** | **l** | **a** | **s** | **s** | **e** | **s** |  | **i** | **n** |  | **B** | **e** | **n** | **g** | **a** | **l** | **i** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |

**Common Slicing Examples:**

| **Code** | **Result** | **Explanation** |
| --- | --- | --- |
| myvar[0:1] | 'I' | Characters from index 0 **up to** (not including) 1 |
| myvar[2:6] | 'like' | Index 2 to 5 |
| myvar[7:14] | 'classes' | Index 7 to 13 |
| myvar[18:] | 'Bengali' | From index 18 to end |
| myvar[:6] | 'I like' | From start to index 5 |
| myvar[-8:] | 'Bengali' | Last 8 characters |
| myvar[::2] | 'Ilk lsesi egli' | Every second character |
| myvar[::-1] | 'ilagneB ni sessalc ekil I' | Reversed string |

**Notes:**

* Negative indices count from the end (-1 is last character).
* start is inclusive; stop is exclusive.
* step can be used to skip characters or reverse the string.

**6. Negative Indexing**

Python allows you to count from the end of the string using negative numbers:

word = "Python"

# -6-5-4-3-2-1 (negative index positions)

# 0 1 2 3 4 5 (positive index positions)

# Accessing characters from the end

print(word[-1]) # Output: n (last character)

print(word[-2]) # Output: o (second to last)

print(word[-6]) # Output: P (first character, same as word[0])

# Negative slicing

text = "Hello World"

print(text[-5:]) # Output: World (last 5 characters)

print(text[:-6]) # Output: Hello (all except last 6 characters)

print(text[-5:-1]) # Output: Worl (from -5 to -2)

# Practical example

phone = "555-123-4567"

area\_code = phone[:3] # Output: 555

last\_four = phone[-4:] # Output: 4567

print(f"Area code: {area\_code}")

print(f"Last four digits: {last\_four}")

**#Practices**

# negetive indexing

# in python, negative indexing allows us to access characters from the end of the string

# -1 is the last character

# -2 is the second last

# and so on

myvar10 = " I like classes in Bengali"

myvar10 [-1]

| **Character** | **' '** | **I** |  | **l** | **i** | **k** | **e** |  | **c** | **l** | **a** | **s** | **s** | **e** | **s** |  | **i** | **n** |  | **B** | **e** | **n** | **g** | **a** | **l** | **i** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| **Negative** | -26 | -25 | -24 | -23 | -22 | -21 | -20 | -19 | -18 | -17 | -16 | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

**Examples of Negative Indexing:**

| **Code** | **Result** | **Reason** |
| --- | --- | --- |
| myvar10[-1] | 'i' | Last character |
| myvar10[-2] | 'l' | Second last character |
| myvar10[-8] | ' ' | Space before "Bengali" |
| myvar10[-7:] | 'Bengali' | Last word using negative slicing |
| myvar10[-13:-7] | 's in ' | Part between "classes" and "Bengali" |
| myvar10[::-1] | 'ilagneB ni sessalc ekil I ' | Reversed string |

**Part 1: Diagram with Both Positive & Negative Indexes**

myvar = "I like Bangla"

**Part 1: Diagram with Both Positive & Negative Indexes**

**Indexing Map**

| **Character** | **I** | **' '** | **l** | **i** | **k** | **e** | **' '** | **B** | **a** | **n** | **g** | **l** | **a** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pos Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Neg Index | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

**Part 2: Slicing Practice Using Negative Indexes**

| **Code** | **Result** | **Explanation** |
| --- | --- | --- |
| myvar[-1] | 'a' | Last character |
| myvar[-6:] | 'Bangla' | Slice from -6 (B) to end |
| myvar[-11:-8] | 'lik' | Substring from 'l' to 'k' |
| myvar[-13:-12] | 'I' | First character using negative index |
| myvar[::-1] | 'algnaB ekil I' | Entire string reversed |

**Part 3: Using Negative Indexes in Loops & Conditions**

**Loop with Negative Indexing**

# Print characters using negative indexes

for i in range(-1, -len(myvar)-1, -1):

print(f"Index {i}: {myvar[i]}")

This prints:

Index -1: a

Index -2: l

Index -3: g

...

Index -13: I

**Step-by-Step Explanation of the above example**

myvar = "I like Bangla"

Length: len(myvar) = 13

**Syntax:** range(start, stop, step)

**Values in this case:**

| **Parameter** | **Value** | **Meaning** |
| --- | --- | --- |
| start | -1 | Start from the **last character** (a) |
| stop | -len(myvar) - 1 → -13 - 1 → -14 | Go until just **before** the first character |
| step | -1 | Step **backwards** one index at a time |

**What does range(-1, -14, -1) generate?**

It creates this sequence of indexes: -1, -2, -3, ..., -13

Which correspond to the string: 'a', 'l', 'g', 'n', 'a', 'B', ' ', 'e', 'k', 'i', 'l', ' ', 'I'

**Full Loop Behavior**

for i in range(-1, -len(myvar)-1, -1):

print(f"Index {i}: {myvar[i]}")

This will print:

Index -1: a

Index -2: l

Index -3: g

Index -4: n

Index -5: a

Index -6: B

Index -7:

Index -8: e

Index -9: k

Index -10: i

Index -11: l

Index -12:

Index -13: I

**Why -len(myvar)-1?**

* Because Python slicing/loop ranges are **exclusive of the stop value**.
* -len(myvar) is -13, which points to 'I'.
* We go to -14 to **include -13** in the loop.

**Condition with Negative Index**

if myvar[-1] == 'a':

print("Ends with 'a'")

**Real-World Uses**

| **Use Case** | **Example** |
| --- | --- |
| Check last character | if myvar[-1] == "." |
| Get last word | myvar.split()[-1] → 'Bangla' |
| Trim last 3 chars | myvar[:-3] → 'I like Ban' |
| Reverse string | reversed\_str = myvar[::-1] |

**7. String Functions**

**len() - Get String Length**

text = "Hello World"

length = len(text)

print(f"The text has {length} characters") # Output: The text has 11 characters

name = "Alice"

if len(name) > 3:

print("Long name") # Output: Long name

# Checking empty strings

user\_input = ""

if len(user\_input) == 0:

print("No input provided") # Output: No input provided

**#Practices**

# length function len()

# it will print the total length of the variable

myvar11= "I love India"

len(myvar11)

12

**strip() - Remove Whitespace**

# Remove spaces from beginning and end

messy\_text = " Hello World "

clean\_text = messy\_text.strip()

print(f"Before: '{messy\_text}'") # Output: Before: ' Hello World '

print(f"After: '{clean\_text}'") # Output: After: 'Hello World'

# Remove specific characters

text = "xxxHello Worldxxx"

cleaned = text.strip('x')

print(cleaned) # Output: Hello World

# Real-world example: cleaning user input

user\_name = input("Enter your name: ").strip() # Removes extra spaces

print(f"Hello, {user\_name}!")

**#Practices**

# strip function strip()

# it will remove spaces from beginning and end of the variable

myvar12= " I love India "

myvar12.strip()

'I love India'

**lower() - Convert to Lowercase**

text = "HELLO WORLD"

lowercase = text.lower()

print(lowercase) # Output: hello world

# Case-insensitive comparison

user\_input = "YES"

if user\_input.lower() == "yes":

print("User agreed") # Output: User agreed

# Email processing

email = "USER@GMAIL.COM"

clean\_email = email.lower()

print(clean\_email) # Output: user@gmail.com

**upper() - Convert to Uppercase**

text = "hello world"

uppercase = text.upper()

print(uppercase) # Output: HELLO WORLD

# Shouting effect

message = "attention please"

print(message.upper() + "!") # Output: ATTENTION PLEASE!

**replace() - Replace Text**

text = "Hello World"

new\_text = text.replace("World", "Python")

print(new\_text) # Output: Hello Python

# Replace multiple occurrences

sentence = "I love cats. Cats are amazing. Cats are cute."

new\_sentence = sentence.replace("cats", "dogs").replace("Cats", "Dogs")

print(new\_sentence) # Output: I love dogs. Dogs are amazing. Dogs are cute.

# Remove characters by replacing with empty string

phone = "555-123-4567"

digits\_only = phone.replace("-", "")

print(digits\_only) # Output: 5551234567

# Limit replacements

text = "banana banana banana"

limited = text.replace("banana", "apple", 2) # Replace only first 2

print(limited) # Output: apple apple banana

**#Practices**

# replace a word or character in a string, we use the .replace()

myvar13 = "I know English"

myvar13 = myvar13.replace("English", "Bengali")

print(myvar13)

I know Bengali

# replace a character

text = "data patterns"

text = text.replace("a","@")

print(text)

d@t@ p@tterns

# Replace multiple words using chaining

department = "design and engineering"

department = department.replace("design", "research").replace("engineering", "development")

print(department)

research and development

**split() - Split String into List**

sentence = "Hello World Python"

words = sentence.split()

print(words) # Output: ['Hello', 'World', 'Python']

# Split by specific character

data = "apple,banana,cherry"

fruits = data.split(",")

print(fruits) # Output: ['apple', 'banana', 'cherry']

# Split with limit

text = "one-two-three-four"

parts = text.split("-", 2) # Split only twice

print(parts) # Output: ['one', 'two', 'three-four']

# Processing CSV data

csv\_line = "John,25,Engineer,New York"

name, age, job, city = csv\_line.split(",")

print(f"Name: {name}, Age: {age}, Job: {job}, City: {city}")

# Output: Name: John, Age: 25, Job: Engineer, City: New York

# Split lines

multiline = "Line 1\nLine 2\nLine 3"

lines = multiline.split("\n")

print(lines) # Output: ['Line 1', 'Line 2', 'Line 3']

**# Creating acronyms**

company = "international business machines"

words = company.split()

acronym = ""

for word in words:

acronym += word[0].upper()

print(acronym) # Output: IBM

**Step-by-Step Explanation**

**1. company = "international business machines"**

This is your input string — the full name of a company.

**2. words = company.split()**

* The .split() method breaks the string into a list of words by default using **spaces**.
* So now:

*words = ['international', 'business', 'machines']*

**3. acronym = ""**

* Initializes an empty string to store the resulting acronym.

**4. Loop:**

*for word in words:*

*acronym += word[0].upper()*

* Loops through each word in the list.
* word[0] takes the **first letter** of the word.
* .upper() converts it to **uppercase**.
* += adds it to the growing acronym string.

**Iteration Breakdown:**

| **Iteration** | **word** | **word[0]** | **word[0].upper()** | **acronym** |
| --- | --- | --- | --- | --- |
| 1 | "international" | 'i' | 'I' | 'I' |
| 2 | "business" | 'b' | 'B' | 'IB' |
| 3 | "machines" | 'm' | 'M' | 'IBM' |

**5. print(acronym)**

* Prints the final acronym: **IBM**

**Output:**

IBM

**8. Substring in String (Checking if Text Exists)**

Use the in keyword to check if one string exists inside another:

text = "Hello World"

# Basic checking

if "World" in text:

print("Found 'World' in the text") # Output: Found 'World' in the text

if "Python" in text:

print("Found Python")

else:

print("Python not found") # Output: Python not found

# Case-sensitive checking

sentence = "I love Python programming"

if "python" in sentence:

print("Found python")

else:

print("Not found (case matters)") # Output: Not found (case matters)

# Case-insensitive checking

if "python" in sentence.lower():

print("Found python (case-insensitive)") # Output: Found python (case-insensitive)

# Practical examples

email = "user@gmail.com"

if "@" in email and "." in email:

print("Valid email format") # Output: Valid email format

# Checking file extensions

filename = "document.pdf"

if filename.endswith(".pdf"):

print("This is a PDF file") # Output: This is a PDF file

# Word filtering

comment = "This is a great product"

banned\_words = ["bad", "terrible", "awful"]

contains\_banned = any(word in comment.lower() for word in banned\_words)

if not contains\_banned:

print("Comment approved") # Output: Comment approved

**9. String Concatenation (Joining Strings)**

Multiple ways to combine strings:

# Using + operator

first\_name = "John"

last\_name = "Doe"

full\_name = first\_name + " " + last\_name

print(full\_name) # Output: John Doe

**#Practices**

# Using += operator

message = "Hello"

message += " World"

message += "!"

print(message) # Output: Hello World!

**#Practices**

# Using join() method (efficient for multiple strings)

words = ["Python", "is", "awesome"]

sentence = " ".join(words)

print(sentence) # Output: Python is awesome

# Join with different separators

fruits = ["apple", "banana", "cherry"]

comma\_separated = ", ".join(fruits)

print(comma\_separated) # Output: apple, banana, cherry

dash\_separated = " - ".join(fruits)

print(dash\_separated) # Output: apple - banana - cherry

# Practical example: building file paths

folder = "documents"

subfolder = "projects"

filename = "report.txt"

full\_path = folder + "/" + subfolder + "/" + filename

print(full\_path) # Output: documents/projects/report.txt

# Building URLs

base\_url = "https://api.example.com"

endpoint = "users"

user\_id = "123"

full\_url = base\_url + "/" + endpoint + "/" + user\_id

print(full\_url) # Output: https://api.example.com/users/123

**10. String Format**

**Using .format() method**

name = "Alice"

age = 25

# Basic formatting

message = "My name is {} and I am {} years old".format(name, age)

print(message) # Output: My name is Alice and I am 25 years old

# Numbered placeholders

text = "I have {0} apples and {1} oranges. {0} are red.".format(5, 3)

print(text) # Output: I have 5 apples and 3 oranges. 5 are red.

# Named placeholders

info = "Name: {name}, Age: {age}, City: {city}".format(

name="Bob",

age=30,

city="New York"

)

print(info) # Output: Name: Bob, Age: 30, City: New York

# Formatting numbers

price = 19.99

quantity = 3

total = price \* quantity

bill = "Price: ${:.2f}, Quantity: {}, Total: ${:.2f}".format(price, quantity, total)

print(bill) # Output: Price: $19.99, Quantity: 3, Total: $59.97

**Using f-strings (Python 3.6+) - Recommended**

name = "Charlie"

age = 28

height = 5.9

# Basic f-string

message = f"My name is {name} and I am {age} years old"

print(message) # Output: My name is Charlie and I am 28 years old

# Expressions inside f-strings

print(f"Next year I will be {age + 1} years old") # Output: Next year I will be 29 years old

# Formatting numbers

price = 123.456

print(f"Price: ${price:.2f}") # Output: Price: $123.46

# Multiple variables

product = "laptop"

cost = 999.99

discount = 0.1

final\_price = cost \* (1 - discount)

print(f"The {product} costs ${cost:.2f}, but with {discount:.0%} off, you pay ${final\_price:.2f}")

# Output: The laptop costs $999.99, but with 10% off, you pay $899.99

# Advanced example from the image

txt = "My name is John, and I am {}"

age = 25

print(txt.format(age)) # Output: My name is John, and I am 25

# Even better with f-strings

name = "John"

age = 25

print(f"My name is {name}, and I am {age}") # Output: My name is John, and I am 25

**Key Differences: String Numbers vs Regular Numbers**

This is crucial to understand:

# These are NUMBERS (can do math)

age = 25

price = 19.99

total = age + price # Result: 44.99

# These are STRINGS (text that looks like numbers)

age\_text = "25"

price\_text = "19.99"

total\_text = age\_text + price\_text # Result: "2519.99" (concatenation, not addition)

# Converting strings to numbers

age\_number = int(age\_text) # Convert to integer

price\_number = float(price\_text) # Convert to float

real\_total = age\_number + price\_number # Result: 44.99

# Converting numbers to strings

age = 25

age\_string = str(age) # Result: "25"

**Complete Practical Example**

# User information system

print("=== User Registration System ===")

# Get user input

first\_name = input("Enter your first name: ").strip().title()

last\_name = input("Enter your last name: ").strip().title()

email = input("Enter your email: ").strip().lower()

age\_input = input("Enter your age: ").strip()

# Process the data

full\_name = first\_name + " " + last\_name

age = int(age\_input)

username = (first\_name[0] + last\_name).lower()

# Validate email

is\_valid\_email = "@" in email and "." in email

# Create user profile

profile = f"""

--- USER PROFILE ---

Full Name: {full\_name}

Email: {email}

Age: {age}

Username: {username}

Email Valid: {is\_valid\_email}

Name Length: {len(full\_name)} characters

Adult: {age >= 18}

"""

print(profile)

# Example output:

# --- USER PROFILE ---

# Full Name: John Smith

# Email: john.smith@gmail.com

# Age: 25

# Username: jsmith

# Email Valid: True

# Name Length: 10 characters

# Adult: True

**Summary**

**Strings** are text wrapped in quotes and are fundamental to Python programming. Key points to remember:

1. **Always use quotes** - Single 'text' or double "text"
2. **Indexing starts at 0** - First character is at position 0
3. **Slicing extracts portions** - Use [start:end] syntax
4. **Many built-in methods** - len(), strip(), lower(), upper(), replace(), split()
5. **Check content with in** - Easy way to find substrings
6. **Multiple ways to combine** - +, +=, join(), .format(), f-strings
7. **String vs numbers** - Remember the difference between "5" and 5

Strings are one of the most frequently used data types in Python, so mastering these concepts will make you a more effective programmer!