Author: PyShaala
Date: August 17, 2025
YouTube Channel: PyShaala

What are Strings?

In Python, a **string** is a sequence of characters. Strings are used to represent text data and are one of the most common data types you'll work with.

Key Characteristics:

- **Ordered:** Characters in a string have a defined order, and you can access them by their position (index).
- **Immutable:** Once a string is created, it cannot be changed. Any operation that seems to modify a string actually creates a *new* string.
- **Sequence:** Strings are sequences, similar to lists and tuples, allowing you to iterate over their characters.
- **Created using quotes:** Strings can be enclosed in single quotes (`'...'`), double quotes (`"..."`), or triple quotes ('''...''' or """..."""). Triple quotes are useful for multiline strings.

Common Uses:

Strings are fundamental for handling textual information in Python, used for:

- Storing names, messages, and other text-based data.
- Working with file paths and URLs.
- Representing and manipulating data read from files or user input.
- Formatting output.

Syntax

Creating Strings

Strings can be created using single, double, or triple quotes.

In [1]:
```
# Using single quotes
single_quoted_string = 'Hello'
print(single_quoted_string)
print(type(single_quoted_string))
```

```
Hello
<class 'str'>
```

In [2]:
```
# Using double quotes
double_quoted_string = "World"
print(double_quoted_string)
print(type(double_quoted_string))
```

```
World
<class 'str'>
```

In [3]:
```
# Using triple single quotes for multiline strings
multiline_string_single = '''This is a
multiline
string using triple single quotes.'''
print(multiline_string_single)
```

```
This is a
multiline
string using triple single quotes.
```

In [4]:
```
# Using triple double quotes for multiline strings
multiline_string_double = """This is another
multiline
string using triple double quotes."""
print(multiline_string_double)
print(type(multiline_string_double))
```

```
This is another
multiline
string using triple double quotes.
<class 'str'>
```

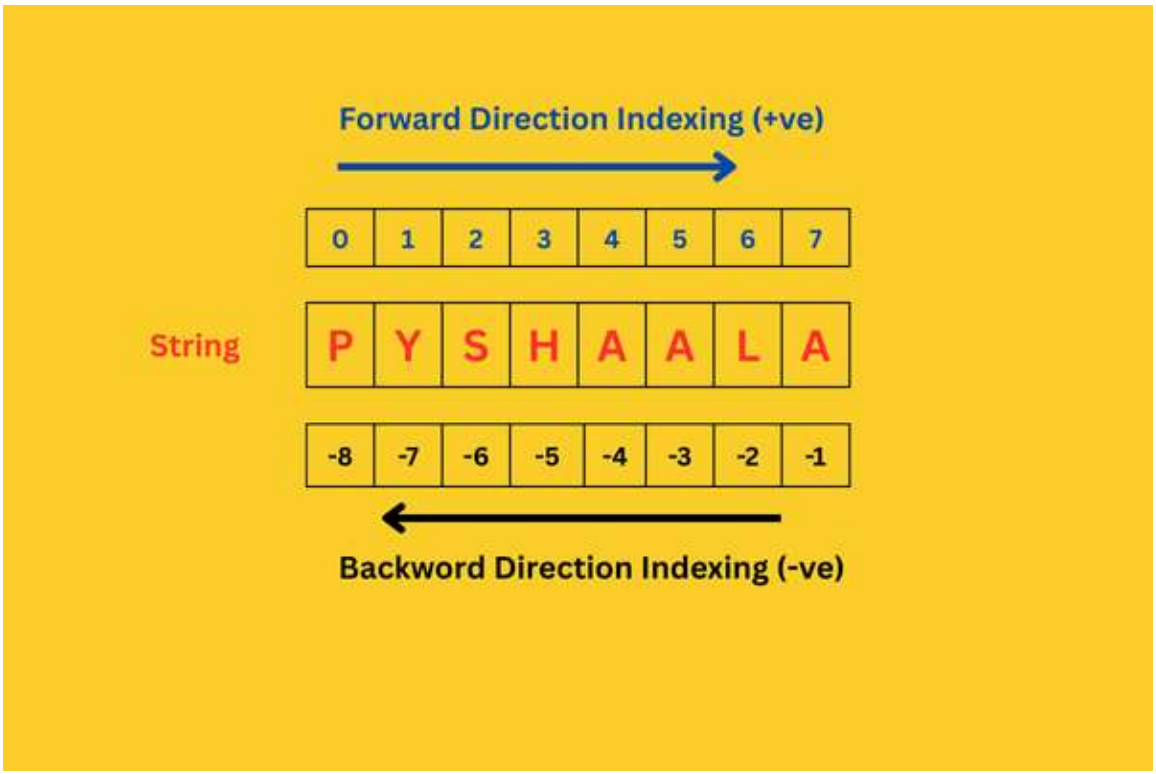
In [5]:
```
# Including quotes within a string
string_with_quotes1 = "He said, 'Hello!'"

string_with_quotes2 = 'She said, "Hi!"'
string_with_quotes3 = 'It\'s a sunny day.' # Using escape character
print(string_with_quotes1)
print(string_with_quotes2)
print(string_with_quotes3)
```

```
He said, 'Hello!'
She said, "Hi!"
It's a sunny day.
```

🟢 Accessing String Elements

You can access individual characters or a range of characters in a string using **indexing** and **slicing**, similar to lists.

- **Indexing:** Accessing a single character using its position (index).
  - Positive indexing starts from `0` for the first character.
  - Negative indexing starts from `-1` for the last character.



In [6]:
```
string = "PYSHAALA"
# Accessing elements using positive indexing
# print(string[3])

# Accessing elements using negative indexing
print(string[-4])
```

A

- **Slicing:** Accessing a sequence of characters using a range of indices. The syntax is `string[start:stop:step]`.
  - `start`: From where we have to consider slice (inclusive). Defaults to the beginning of the string.
  - `stop`: We have to terminate the slice(substring) at the stop-1 (exclusive). Defaults to the end of the string.
  - `step`: The step size for slicing. Defaults to `1`.

- **Behaviour of Slice operator:**

`string[start:stop:step]`

`step` : value either `+ve` or `-ve` .

- if +ve then it should be forward direction(left to right) and we have to consider start to end-1.
- if -ve then it should be backward direction(right to left) and we have to consider start to end+1

In [7]:
```
my_string = 'pyshaala'
print(f' 0  1  2  3  4  5  6  7')
print(f' p  y  s  h  a  a  l  a')
print(f'-8 -7 -6 -5 -4 -3 -2 -1')
```

```
 0  1  2  3  4  5  6  7
 p  y  s  h  a  a  l  a
-8 -7 -6 -5 -4 -3 -2 -1
```

`Note: stop`

- In the backward direction if stop value is -1 then result is always empty.
- In the forward direction if stop value is 0 then result is always empty.

In [8]:
```
s = "pyshaala"
# In the backward direction if stop value is -1 then result is always empty.
print(s[4:-1:-1])  # empty
# In the forward direction if stop value is 0 then result is always empty.
print(s[2:0:1])    # empty
```

📌 **Quick Rule:**

- Backward (- step) → stop = -1 → always empty.
- Forward (+ step) → stop = 0 → always empty.

In [32]:
```
my_string = 'pyshaala'
print(f' 0  1  2  3  4  5  6  7')
print(f' p  y  s  h  a  a  l  a')
print(f'-8 -7 -6 -5 -4 -3 -2 -1')
# Accessing elements using slicing
my_string = 'pyshaala'
print("Characters from index 0 to 6 (exclusive):", my_string[0:6])
print("Characters from index 7 to the end:", my_string[7:])
print("Characters from the beginning to index 6 (exclusive):", my_string[:6])
print("Every second character:", my_string[::-2])
print("Reverse the string:", my_string[::-1])

# Trying to modify a string (will cause an error)
try:
    my_string[0] = 'J'
except TypeError as e:
    print(f"Error when trying to modify string: {e}")
```

```
 0  1  2  3  4  5  6  7
 p  y  s  h  a  a  l  a
-8 -7 -6 -5 -4 -3 -2 -1
Characters from index 0 to 6 (exclusive): pyshaa
Characters from index 7 to the end: a
Characters from the beginning to index 6 (exclusive): pyshaa
Every second character: aahy
Reverse the string: alaahsyp
Error when trying to modify string: 'str' object does not support item assignment
```

🌍 Real-Life Examples

Strings are used in countless real-world programming scenarios to handle and process text data. Here are a few examples demonstrating common uses of strings.

In [10]:
```
# Example 1: Email Address Parsing

# This example demonstrates splitting a string (email address) into parts.

email = "user.name@example.com"
print(f"Original email: {email}")

# Split the email address at the "@" symbol
parts = email.split("@")
print(f'Parts:{parts}')
username = parts[0]
domain = parts[1]

print(f"Username: {username}")
print(f"Domain: {domain}")
```

```
# Check if the domain ends with ".com"
if domain.endswith(".com"):
    print("Domain ends with .com")
else:
    print("Domain does not end with .com")

# Replace the domain
new_email = email.replace("example.com", "newdomain.org")
print(f"New email after replacing domain: {new_email}")
```

```
Original email: user.name@example.com
Parts:['user.name', 'example.com']
Username: user.name
Domain: example.com
Domain ends with .com
New email after replacing domain: user.name@newdomain.org
```

🔁 String Formatting (f-strings, .format(), % formatting)

Formatting inserts variables into strings:

- **% formatting**: Old style, like C's printf.
- **.format()**: More flexible, positional or named.
- **f-strings**: Modern (Python 3.6+), fastest and readable.

Use f-strings for new code.

In [33]:
```
name = "PyShaala"
subs = 1000

# % formatting
print("Welcome to %s! Subs: %d" % (name, subs))  # 'Welcome to PyShaala! Subs: 1000

# .format()
print("formart2 :Welcome to {}! Subs: {}".format(name, subs))  # Same as above
print("Welcome to {channel}! Subs: {count}".format(channel=name, count=subs))  # Na

# f-strings
print(f"Welcome to {name}! Subs: {subs}")  # Same
print(f"Subs doubled: {subs * 2}")        # With expressions
```

```
Welcome to PyShaala! Subs: 1000
formart2 :Welcome to PyShaala! Subs: 1000
Welcome to PyShaala! Subs: 1000
Welcome to PyShaala! Subs: 1000
Subs doubled: 2000
```

In [12]:
```
# Example 2: Formatting Data for Output

# This example shows how to format string output using f-strings and rounding.

product = "Laptop"
price = 1200.50
```

```
quantity = 3
total = price * quantity

print("--- Order Summary ---")

# Using f-strings to format output
print(f"Product: {product}")
print(f"Price per item: ${price:.2f}") # Format price to 2 decimal places
print(f"Quantity: {quantity}")
print(f"Total Cost: ${total:.2f}") # Format total to 2 decimal places

# Another formatting example
city = "New York"
temp = 72.567
print(f"The temperature in {city} is {temp:.1f}°F.") # Round temperature to 1 decim
```

```
--- Order Summary ---
Product: Laptop
Price per item: $1200.50
Quantity: 3
Total Cost: $3601.50
The temperature in New York is 72.6°F.
```

In [13]:
```
# Example 3: Processing User Input

# This example demonstrates cleaning and validating user input.

user_input = input("Enter your name (or leave blank): ")
print(f'User Input:{user_input}')
# Remove leading/trailing whitespace
cleaned_input = user_input.strip()
print(f"Cleaned input: '{cleaned_input}'")

# Check if the input is empty after stripping whitespace
if not cleaned_input:
    print("Name was not entered.")
else:
    # Capitalize the first letter of each word
    formatted_name = cleaned_input.title()
    print(f"Formatted name: {formatted_name}")

    # Check if the name contains only alphabetic characters
    if formatted_name.replace(" ", "").isalpha():
         print("Name contains only alphabetic characters.")
    else:
         print("Name contains non-alphabetic characters.")
```

```
User Input:pyshaala
Cleaned input: 'pyshaala'
Formatted name: Pyshaala
Name contains only alphabetic characters.
```

📝 String Operations (Concatenation, Repetition, Membership)

Common operations:

- **Concatenation**: Using `+` to join strings.
- **Repetition**: Using `*` to repeat strings.
- **Membership**: `in` or `not in` to check if a substring exists.

These are efficient for building dynamic text, like generating reports.

In [34]:
```
# Concatenation
str1 = "Hello"
str2 = 'Pyshaala!'
greeting = str1 + ' ' + str2

print(str1 + ' ' + str2)  # 'Hello PyShaala!'

# Repetition
stars = "*" * 5
print(stars)  # '*****'

# Membership
print("Py" in greeting)    # True
print("Java" not in greeting)  # True
```

```
Hello Pyshaala!
*****
True
True
```

⚙️ String Methods (Functions)

Python strings come with a rich set of built-in methods that allow you to perform various operations, such as changing case, searching, replacing, splitting, and joining. These methods return *new* strings because strings are immutable.

Here are some of the most commonly used and useful string methods:

- `.lower()` : Converts the string to lowercase.
- `.upper()` : Converts the string to uppercase.
- `.strip()` : Removes leading and trailing whitespace.
- `.replace(old, new)` : Returns a new string with all occurrences of `old` replaced by `new` .
- `.split(separator)` : Splits the string into a list of substrings using the specified separator. If no separator is given, it splits by whitespace.
- `.join(iterable)` : Concatenates the elements of an iterable (like a list of strings) into a single string, using the string the method is called on as the separator.
- `.find(substring)` : Returns the lowest index in the string where the substring is found. Returns -1 if not found.
- `.index(substring)` : Similar to `find()` , but raises a `ValueError` if the substring is not found.

- `.startswith(prefix)` : Returns `True` if the string starts with the specified prefix.
- `.endswith(suffix)` : Returns `True` if the string ends with the specified suffix.
- `.isalpha()` : Returns `True` if all characters in the string are alphabetic.
- `.isdigit()` : Returns `True` if all characters in the string are digits.
- `.isalnum()` : Returns `True` if all characters are alphanumeric (letters or numbers).
- `.isspace()` : Returns `True` if the string contains only whitespace characters.
- `.title()` : Returns a titlecased version of the string, where words start with an uppercase character and the rest are lowercase.
- `.count(substring)` : Returns the number of occurrences of a substring in the string.
- `.len()` : (Note: `len()` is a built-in function, not a string method, but commonly used with strings) Returns the length of the string.

In [15]:
```
# lower(): Converts the string to lowercase.
# upper(): Converts the string to uppercase.
# title(): Returns a titlecased version of the string, where words start with an up
# Examples of common string methods

my_string = "  Hello, Python World!  "
print(f"Original string: '{my_string}'")

# Case conversion
print(f"Lowercase: '{my_string.lower()}'")
print(f"Uppercase: '{my_string.upper()}'")
print(f"Titlecase: '{my_string.title()}'")
```

```
Original string: '  Hello, Python World!  '
Lowercase: '  hello, python world!  '
Uppercase: '  HELLO, PYTHON WORLD!  '
Titlecase: '  Hello, Python World!  '
```

In [16]:
```
# strip(): Removes leading and trailing whitespace.
# Stripping whitespace
stripped_string = my_string.strip()
print(f"Stripped: '{stripped_string}'")
```

```
Stripped: 'Hello, Python World!'
```

In [17]:
```
# replace(old, new): Returns a new string with all occurrences of `old` replaced by
# Replacing substrings
stripped_string = "Python new my Python"

replaced_string = stripped_string.replace("Python", "Java")
print(f"Replaced: '{replaced_string}'")
```

```
Replaced: 'Java new my Java'
```

In [18]:
```
# split(separator): Splits the string into a list of substrings using the specified
# Splitting strings
sentence = "Python is easy to learn"
words = sentence.split() # Split by whitespace
print(f"Words (split by space): {words}")

csv_data = "apple,banana,cherry,date"
```

```
fruits = csv_data.split(",") # Split by comma
print(f"Fruits (split by comma): {fruits}")
```

```
Words (split by space): ['Python', 'is', 'easy', 'to', 'learn']
Fruits (split by comma): ['apple', 'banana', 'cherry', 'date']
```

In [19]:
```
# join(iterable): Concatenates the elements of an iterable (like a list of strings)
# Joining strings
words_list = ["This", "is", "a", "list", "of", "words."]
joined_sentence = " ".join(words_list) # Join with space
print(f"Joined with space: '{joined_sentence}'")

hyphenated_word = "-".join(["my", "variable", "name"]) # Join with hyphen
print(f"Joined with hyphen: '{hyphenated_word}'")
```

```
Joined with space: 'This is a list of words.'
Joined with hyphen: 'my-variable-name'
```

In [35]:
```
# find(substring): Returns the lowest index in the string where the substring is fo
# index(substring): Similar to find() but raises a `ValueError` if the substring is

# Searching for substrings
search_string = "Python programming is fun."
print(f"String to search in: '{search_string}'")
print(f"Index of 'Python': {search_string.find('Python')}")
print(f"Index of 'java': {search_string.find('java')}") # Returns -1 if not found

try:
    print(f"Index of 'Java': {search_string.index('Java')}") # Will raise ValueErro
except ValueError as e:
    print(f"Error using index() for 'Java': {e}")
```

```
String to search in: 'Python programming is fun.'
Index of 'Python': 0
Index of 'java': -1
Error using index() for 'Java': substring not found
```

In [21]:
```
# startswith(prefix): Returns `True` if the string starts with the specified prefix
# endswith(suffix): Returns `True` if the string ends with the specified suffix.
print(f"Does it start with 'Python'?: {search_string.startswith('Python')}")
print(f"Does it end with 'fun.'?: {search_string.endswith('fun.')}")
print(f"Does it end with 'boring'?: {search_string.endswith('boring')}")
```

```
Does it start with 'Python'?: True
Does it end with 'fun.'?: True
Does it end with 'boring'?: False
```

In [22]:
```
# isalpha(): Returns `True` if all characters in the string are alphabetic.
# isdigit(): Returns `True` if all characters in the string are digits.
# isalnum(): Returns `True` if all characters are alphanumeric (letters or numbers)
# isspace(): Returns `True` if the string contains only whitespace characters.

# Checking string content
check_string1 = "HelloWorld"
check_string2 = ""
check_string3 = "12345@"
check_string4 = "Python"
```

```
print(f"'{check_string1}' is alphanumeric?: {check_string1.isalnum()}")
print(f"'{check_string2}' is space?: {check_string2.isspace()}")
print(f"'{check_string3}' is digit?: {check_string3.isdigit()}")
print(f"'{check_string4}' is alphabetic?: {check_string4.isalpha()}")
```

```
'HelloWorld' is alphanumeric?: True
'' is space?: False
'12345@' is digit?: False
'Python' is alphabetic?: True
```

In [23]:
```
# count(substring): Returns the number of occurrences of a substring in the string.
# Counting substrings
count_string = "banana"
print(f"Number of 'a' in '{count_string}': {count_string.count('b')}")
```

```
Number of 'a' in 'banana': 1
```

In [24]:
```
# len(): (Note: len() is a built-in function, not a string method, but commonly use
# Getting string length using len() function
my_string = 'pyshaala'
print(f"Length of '{my_string}': {len(my_string)}")
```

```
Length of 'pyshaala': 8
```

🧠 Advanced Concepts (Escape sequences, Raw strings, Unicode)

- **Escape sequences**: Special chars like `\n` (newline), `\t` (tab), `\\` (backslash).
- **Raw strings**: Prefix `r` to ignore escapes, useful for regex or paths.
- **Unicode**: Strings support Unicode (e.g., emojis, non-ASCII). Use `\u` for codes.

Handle carefully in file paths or international text.

In [25]:
```
# Escape sequences
# print("Line1\nLine2\tTabbed")  # Line1 (newline) Line2 (tab) Tabbed
# print("Backslash: \\")         # Backslash: \

# Raw strings
print(r"C:\new\folder")        # C:\new\folder (no escape interpretation)

# Unicode
print("Emoji: \U0001F600")     # Emoji: 😀
print("\u2764 PyShaala")       # ❤ PyShaala
```

```
C:\new\folder
Emoji: 😀
❤ PyShaala
```

✅ Do's and ❌ Don'ts

Here are some important things to remember when working with Python strings:

✅ Do's

- **Do** use clear and consistent quoting styles (single or double quotes) within your projects.
- **Do** use triple quotes for multiline strings or when your string contains both single and double quotes.
- **Do** use f-strings (formatted string literals) for easy and readable string formatting.
- **Do** use string methods (like `.lower()`, `.upper()`, `.strip()`, `.split()`, `.replace()`) for common string manipulations, as they are generally efficient.
- **Do** check if a substring exists in a string using the `in` keyword (e.g., `'sub' in my_string`).
- **Do** be mindful that string operations often create new string objects due to their immutability.

❌ Don'ts

- **Don't** try to modify a string in place; remember they are immutable. Operations like `my_string[0] = 'H'` will raise a `TypeError`.
- **Don't** build strings using repeated concatenation with the `+` operator in loops, especially for many small strings. This can be inefficient. Use `.join()` or f-strings instead.
- **Don't** use outdated string formatting methods (like `%` formatting or `.format()`) if f-strings are sufficient for your needs, as f-strings are generally more readable and faster.
- **Don't** assume string comparisons are case-insensitive unless you explicitly convert both strings to the same case (e.g., using `.lower()`).

In [36]:
```
s1 = 'python'
s2 = 'PYTHON'
s1 == s2
s1 == s2.lower()
```

Out[36]:
```
True
```

🏋️ Practice Exercises

Test your understanding of Python strings and their methods with the following exercises. Write your code in the cells below each problem description.

Exercise 1: Basic String Manipulation

1. Create a string variable `greeting` with the value " Hello, World! ".
2. Use the `.strip()` method to remove leading and trailing whitespace from `greeting` and print the result.
3. Use the `.upper()` method to convert the stripped string to uppercase and print it.

4. Use the `.replace()` method to replace "World" with "Python" in the stripped string and print the result.

Exercise 2: String Splitting and Joining

1. Create a string variable `sentence` with the value "Python programming is fun and useful".
2. Use the `.split()` method to split the `sentence` into a list of words and print the list.
3. Join the list of words back into a single string, but this time use a hyphen `-` as the separator. Print the resulting string.

Exercise 3: String Searching and Checking

1. Create a string variable `data_string` with the value "Data Analysis with Python 3.9".
2. Use the `.find()` method to find the index of the substring "Python" in `data_string` and print the index.
3. Check if `data_string` starts with "Data" using `.startswith()` and print the boolean result.
4. Check if `data_string` ends with "3.9" using `.endswith()` and print the boolean result.
5. Check if the string "123" contains only digits using `.isdigit()` and print the result.
6. Check if the string "HelloWorld" contains only alphabetic characters using `.isalpha()` and print the result.

Exercise 4: String Formatting

1. Create variables `item = "Book"`, `price = 19.99`, and `discount = 0.15`.
2. Calculate the discounted price: `discounted_price = price * (1 - discount)`.
3. Use an f-string to print a formatted message like: "The price of [item] after a 15% discount is $[discounted_price rounded to 2 decimal places]."

Exercise 5: Counting and Length

1. Create a string variable `sample_text = "This is a sample text. This text is for sampling."`.
2. Use the `.count()` method to find how many times the word "text" appears in `sample_text` and print the count.
3. Use the `len()` function to find the total number of characters in `sample_text` and print the length.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

✅ Solutions

Here are the solutions to the practice exercises. Try to solve them yourself before looking at the solutions!

Exercise 1 Solution: Basic String Manipulation

In [27]:
```
# Exercise 1 Solution: Basic String Manipulation

greeting = "  Hello, World!  "

# 2. Use the .strip() method
stripped_greeting = greeting.strip()
print(f"Stripped string: '{stripped_greeting}'")

# 3. Use the .upper() method
uppercase_greeting = stripped_greeting.upper()
print(f"Uppercase string: '{uppercase_greeting}'")

# 4. Use the .replace() method
replaced_greeting = stripped_greeting.replace("World", "Python")
print(f"Replaced string: '{replaced_greeting}'")
```

```
Stripped string: 'Hello, World!'
Uppercase string: 'HELLO, WORLD!'
Replaced string: 'Hello, Python!'
```

Exercise 2 Solution: String Splitting and Joining

In [28]:
```
# Exercise 2 Solution: String Splitting and Joining

sentence = "Python programming is fun and useful"

# 2. Use the .split() method
words = sentence.split()
print(f"List of words: {words}")

# 3. Join the list of words back into a single string with hyphen
```

```
hyphenated_sentence = "-".join(words)
print(f"Joined with hyphen: '{hyphenated_sentence}'")
```

```
List of words: ['Python', 'programming', 'is', 'fun', 'and', 'useful']
Joined with hyphen: 'Python-programming-is-fun-and-useful'
```

Exercise 3 Solution: String Searching and Checking

In [29]:
```
# Exercise 3 Solution: String Searching and Checking

data_string = "Data Analysis with Python 3.9"

# 2. Use the .find() method
python_index = data_string.find("Python")
print(f"Index of 'Python': {python_index}")

# 3. Check if it starts with "Data"
starts_with_data = data_string.startswith("Data")
print(f"Starts with 'Data'?: {starts_with_data}")

# 4. Check if it ends with "3.9"
ends_with_version = data_string.endswith("3.9")
print(f"Ends with '3.9'?: {ends_with_version}")

# 5. Check if "123" contains only digits
string_of_digits = "123"
is_digit_string = string_of_digits.isdigit()
print(f"'{string_of_digits}' is digit only?: {is_digit_string}")

# 6. Check if "HelloWorld" contains only alphabetic characters
alpha_string = "HelloWorld"
is_alpha_string = alpha_string.isalpha()
print(f"'{alpha_string}' is alphabetic only?: {is_alpha_string}")
```

```
Index of 'Python': 19
Starts with 'Data'?: True
Ends with '3.9'?: True
'123' is digit only?: True
'HelloWorld' is alphabetic only?: True
```

Exercise 4 Solution: String Formatting

In [30]:
```
# Exercise 4 Solution: String Formatting

item = "Book"
price = 19.99
discount = 0.15

# 2. Calculate the discounted price
discounted_price = price * (1 - discount)

# 3. Use an f-string to print a formatted message
print(f"The price of {item} after a {discount*100:.0f}% discount is ${discounted_pr
```

```
The price of Book after a 15% discount is $16.99.
```

Exercise 5 Solution: Counting and Length

In [31]:
```
# Exercise 5 Solution: Counting and Length

sample_text = "This is a sample text. This text is for sampling."

# 2. Use the .count() method
text_count = sample_text.count("text")
print(f"Number of times 'text' appears: {text_count}")

# 3. Use the len() function
text_length = len(sample_text)
print(f"Total number of characters: {text_length}")
```

```
Number of times 'text' appears: 2
Total number of characters: 49

Total number of characters: 49
```

✨ Summary

- What is String?
- Syntax
- Accesing String Elements
- String Slicing
- Real-life Examples
- String Formatting
- String Operations
- String Methods (Functions)
- Advanced Concepts
- Do's and Don'ts
- Practice Exercises
- Solutions
- Interview Questions

🏋️ Interview Question

`For Answer please visit our youtube channel short video section or Instagram page`

YouTube Channel: PyShaala

🟢 Basic Level

1. What is a string in Python and how is it defined?
2. How to create multi-line strings?

3. Difference between single quotes, double quotes, and triple quotes in strings.
4. How to find the length of a string?
5. How to access individual characters in a string?
6. How to perform string slicing in Python?
7. Are Python strings mutable or immutable? Explain.
8. How to concatenate two or more strings?
9. How to repeat a string multiple times?
10. How to check if a substring exists within a string?

🟡 Intermediate Level

1. Explain the difference between find() and index() methods.
2. How to remove spaces or specific characters from a string? (strip(), lstrip(), rstrip())
3. How to change case of strings (upper(), lower(), title(), capitalize())?
4. How to replace parts of a string using replace()?
5. How to split and join strings in Python?
6. How to check if a string starts or ends with a certain substring?
7. How to count occurrences of a substring in a string?
8. How to check if a string contains only digits, alphabets, or alphanumeric characters? (isdigit(), isalpha(), isalnum())
9. How to pad strings with a specific character? (zfill(), ljust(), rjust())
10. How to center-align a string?

🔴 Advanced / Tricky Level

1. What are raw strings in Python?
2. How to reverse a string?
3. How to remove punctuation from a string?
4. How to check if a string is a palindrome?
5. How to format strings using f-strings, format(), and % formatting?
6. How to use string translation (maketrans(), translate()) in Python?
7. How to encode and decode strings in Python?
8. How to search for patterns in strings using regex? (re module)
9. How to convert a list of characters into a string?
10. How to efficiently concatenate large numbers of strings? (join() vs +)