PART 1: Introduction to Strings (15 minutes)

What is a String?

A string is a sequence of Unicode characters used to represent text data in Python. Strings are one of the most commonly used data types in programming.

Key Characteristics:

- Strings are **immutable** (cannot be changed after creation)
- Strings are **sequences** (ordered collection of characters)
- Strings can contain letters, numbers, symbols, and spaces
- Strings are objects of the str class

Creating Strings

Python allows three ways to create strings:

```
# Single quotes
name = 'Alice'

# Double quotes
greeting = "Hello, World!"

# Triple quotes (for multi-line strings)
message = '''This is a
multi-line
string'''

# Triple double quotes also work
paragraph = """Python is
an amazing
programming language"""
```

Important Note: Single and double quotes are equivalent - choose one style and be consistent.

When to Use Triple Quotes

- Multi-line strings
- Docstrings (documentation strings)
- Strings containing both single and double quotes

```
quote = '''She said, "Python is easy to learn!"'''
```

PART 2: String Indexing (15 minutes)

Understanding Indexing

Every character in a string has a position (index). Python uses **zero-based indexing**.

```
text = "PYTHON"

# Positive indexing (left to right)
# P  Y  T  H  O  N
# 0  1  2  3  4  5

print(text[0])  # Output: P
print(text[1])  # Output: Y
print(text[5])  # Output: N
```

Negative Indexing

Python also supports negative indexing (right to left):

```
text = "PYTHON"

# Negative indexing (right to left)
# P  Y  T  H  O  N
# -6 -5 -4 -3 -2 -1

print(text[-1])  # Output: N
print(text[-2])  # Output: O
print(text[-6])  # Output: P
```

Common Indexing Errors

```
text = "HELLO"

# This will raise an IndexError
# print(text[10])  # Index out of range!
```

Classroom Activity (5 minutes): Have students practice indexing with their own names.

PART 3: String Slicing (20 minutes)

What is Slicing?

Slicing allows you to extract a portion (substring) of a string.

Syntax: string[start:stop:step]

- **start**: Starting index (inclusive)
- **stop**: Ending index (exclusive)
- **step**: Increment value (default is 1)

Basic Slicing Examples

```
text = "Hello World"

# Get characters from index 0 to 4 (not including 5)
print(text[0:5])    # Output: Hello

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

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

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

Slicing with Step

```
text = "PYTHON"

# Every second character
print(text[::2])    # Output: PTO

# Reverse a string
print(text[::-1])   # Output: NOHTYP

# Every second character in reverse
print(text[::-2])   # Output: NHY
```

Practical Slicing Examples

```
email = "student@example.com"

# Extract username (before @)
username = email[:email.index('@')]
print(username)  # Output: student

# Extract domain
domain = email[email.index('@')+1:]
print(domain)  # Output: example.com
```

Classroom Activity (10 minutes): Students extract first name, last name from "John_Doe" format strings.

PART 4: String Operators (15 minutes)

Concatenation (+)

Joining two or more strings together:

```
first_name = "John"
last_name = "Doe"
```

```
# Concatenate strings
full_name = first_name + " " + last_name
print(full_name)  # Output: John Doe

# Multiple concatenations
greeting = "Hello" + ", " + "welcome" + " " + "to" + " " + "Python!"
print(greeting)  # Output: Hello, welcome to Python!
```

Repetition (*)

Repeating a string multiple times:

```
laugh = "Ha"
print(laugh * 3)  # Output: HaHaHa

separator = "-" * 20
print(separator)  # Output: --------------------

# Creating patterns
print("*" * 10)  # Output: **********
```

Membership Operators (in, not in)

Check if a substring exists in a string:

```
sentence = "Python is fun"

print("Python" in sentence)      # Output: True
print("Java" in sentence)        # Output: False
print("difficult" not in sentence)  # Output: True
```

Comparison Operators

Strings can be compared lexicographically:

```
print("apple" < "banana")   # Output: True
print("Python" == "python") # Output: False (case-sensitive)
print("abc" > "ABC")        # Output: True (lowercase > uppercase)
```

PART 5: String Methods (20 minutes)

Case Conversion Methods

```
text = "Python Programming"

print(text.upper())      # Output: PYTHON PROGRAMMING
print(text.lower())      # Output: python programming
print(text.title())      # Output: Python Programming
print(text.capitalize())  # Output: Python programming
print(text.swapcase())    # Output: pYTHON pROGRAMMING
```

Searching Methods

```
sentence = "Learning Python is fun and Python is powerful"

# Find position of substring
print(sentence.find("Python"))     # Output: 9
print(sentence.find("Java"))       # Output: -1 (not found)

# Count occurrences
print(sentence.count("Python"))    # Output: 2
print(sentence.count("is"))        # Output: 2

# Check if string starts/ends with substring
print(sentence.startswith("Learning"))  # Output: True
print(sentence.endswith("powerful"))    # Output: True
```

Cleaning Methods

```
messy_text = "   Hello World   "

print(messy_text.strip())   # Output: "Hello World"
print(messy_text.lstrip())  # Output: "Hello World   "
print(messy_text.rstrip())  # Output: "   Hello World"

# Remove specific characters
text = "***Python***"
print(text.strip("*"))  # Output: Python
```

Replacement Methods

```
sentence = "I like Java"

# Replace substring
new_sentence = sentence.replace("Java", "Python")
print(new_sentence)  # Output: I like Python

# Replace multiple occurrences
text = "one one one"
print(text.replace("one", "two"))  # Output: two two two

# Limit replacements
print(text.replace("one", "two", 2))  # Output: two two one
```

Splitting and Joining

```
# Split string into list
sentence = "Python is awesome"
words = sentence.split()
print(words)  # Output: ['Python', 'is', 'awesome']

# Split by specific delimiter
csv_data = "John,25,Engineer"
data = csv_data.split(",")
print(data)  # Output: ['John', '25', 'Engineer']

# Join list into string
words = ["Python", "is", "fun"]
sentence = " ".join(words)
print(sentence)  # Output: Python is fun

# Join with different delimiter
csv_line = ",".join(["Alice", "30", "Doctor"])
print(csv_line)  # Output: Alice,30,Doctor
```

Validation Methods

```
# Check if string contains only letters
print("Python".isalpha())      # Output: True
print("Python3".isalpha())     # Output: False

# Check if string contains only digits
print("12345".isdigit())       # Output: True
print("123abc".isdigit())      # Output: False

# Check if string contains only alphanumeric characters
print("Python3".isalnum())     # Output: True
print("Python 3".isalnum())    # Output: False

# Check if string is in lowercase/uppercase
print("python".islower())      # Output: True
print("PYTHON".isupper())      # Output: True

# Check if string contains only whitespace
print("   ".isspace())         # Output: True
```

PART 6: String Immutability (10 minutes)

Understanding Immutability

Strings in Python cannot be changed after creation. Any operation that appears to modify a string actually creates a new string.

```
text = "Hello"

# This will cause an error!
# text[0] = "h"  # TypeError: 'str' object does not support item assignment

# The correct way is to create a new string
text = "h" + text[1:]
print(text)  # Output: hello

# Or use replace
text = "Hello"
text = text.replace("H", "h")
print(text)  # Output: hello
```

Why Immutability Matters

1. **Memory efficiency**: Identical strings can share memory
2. **Thread safety**: Immutable objects are inherently thread-safe
3. **Dictionary keys**: Strings can be used as dictionary keys because they're immutable

---

PART 7: String Formatting (15 minutes)

Method 1: Using + Operator

```
name = "Alice"
age = 25
message = "My name is " + name + " and I am " + str(age) + " years old"
print(message)
```

Method 2: Using format() Method

```
name = "Alice"
age = 25

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

# Named arguments
message = "My name is {n} and I am {a} years old".format(n=name, a=age)
print(message)

# Index-based
message = "I am {1} years old and my name is {0}".format(name, age)
print(message)
```

Method 3: Using f-strings (Python 3.6+) - RECOMMENDED

```
name = "Alice"
age = 25
height = 5.6

# Basic f-string
message = f"My name is {name} and I am {age} years old"
print(message)

# Expressions inside f-strings
print(f"In 5 years, I'll be {age + 5} years old")

# Formatting numbers
price = 49.99
print(f"The price is ${price:.2f}")  # Output: The price is $49.99

# Formatting with width and alignment
print(f"{'Name':<10} {'Age':>5}")  # Left and right align
print(f"{name:<10} {age:>5}")
```

PART 8: Practical Examples (10 minutes)

Example 1: Password Validator

```
password = input("Enter password: ")

# Check password criteria
has_length = len(password) >= 8
has_digit = any(char.isdigit() for char in password)
has_upper = any(char.isupper() for char in password)
has_lower = any(char.islower() for char in password)

if has_length and has_digit and has_upper and has_lower:
    print("Strong password!")
else:
    print("Weak password. Must be 8+ characters with digits, uppercase and lowercase.")
```

Example 2: Text Analyzer

```
text = "Python is a high-level programming language"

print(f"Length: {len(text)}")
print(f"Words: {len(text.split())}")
print(f"Uppercase letters: {sum(1 for c in text if c.isupper())}")
print(f"Lowercase letters: {sum(1 for c in text if c.islower())}")
print(f"Spaces: {text.count(' ')}")
```

Example 3: Name Formatter

```
def format_name(full_name):
    # Convert to title case and remove extra spaces
    full_name = full_name.strip().title()

    # Split into parts
    parts = full_name.split()

    if len(parts) >= 2:
        first_name = parts[0]
        last_name = parts[-1]
        return f"{last_name}, {first_name}"
    else:
        return full_name

print(format_name("john doe"))       # Output: Doe, John
print(format_name("  ALICE SMITH  ")) # Output: Smith, Alice
```
