Python Strings

Strings are sequences of characters enclosed in single, double, or triple quotes. In Python, strings are immutable, which means they cannot be changed after they are created. This means that any operation that modifies a string actually creates a new string.

Creating Strings

Strings can be created using single quotes (`'`), double quotes (`"`). Triple quotes (`'''` or `"""`) are used to create multi-line strings.

Example:

```
# Single quotes
string1 = 'Hello, World!'
print(string1)

# Double quotes
string2 = "Hello, World!"
print(string2)
```

Multi-line Strings

Multi-line strings can be created using triple quotes (`'''` or `"""`).

Example:

```
multiline_string = '''This is a
multi-line
string.'''
print(multiline_string)
```

Accessing Characters in a String

Individual characters in a string can be accessed using indexing. Python uses zero-based indexing, which means the first character has an index of 0, the second character has an index of 1, and so on.

Example:

```
string = "Hello, World!"

print(string[0])  # Output: H

print(string[7])  # Output: W
```

Slicing Strings

You can extract a substring from a string using slicing. The syntax for slicing is `string[start:end:step]`, where `start` is the starting index, `end` is the ending index (exclusive), and `step` is the step size. more read here

Example:

```
string = "Hello, World!"

print(string[7:12])  # Output: World
```

String Concatenation

Strings can be concatenated using the `+` operator.

Example:

```
string1 = "Hello"
string2 = "World"

result = string1 + ", " + string2 + "!"

print(result)  # Output: Hello, World!
```

String Methods

Python provides several built-in methods for working with strings. Some common string methods include:

- `upper()`: Converts all characters in a string to uppercase.
- `lower()`: Converts all characters in a string to lowercase.
- `strip()`: Removes leading and trailing whitespace from a string.
- `replace()`: Replaces a substring with another substring.
- `split()`: Splits a string into a list of substrings based on a delimiter.

Example:

```
string = "Hello, World!"

print(string.upper())  # Output: HELLO, WORLD!
print(string.lower())  # Output: hello, world!
print(string.strip())  # Output: Hello, World!
print(string.replace("Hello", "Hi"))  # Output: Hi, World!
print(string.split(", "))  # Output: ['Hello', 'World!']
```

String Formatting

String formatting allows you to create dynamic strings by inserting variables or expressions into a string. There are several ways to format strings in Python, including:

- Using the `%` operator
- Using the `format()` method
- Using f-strings (formatted string literals)

Example:

```
name = "Alice"
age = 30

# Using the % operator
formatted_string = "Hello, %s! You are %d years old." % (name, age)
print(formatted_string)

When using the `%` operator for string formatting, the format
specifier `%s` is used for strings, `%d` is used for integers, and
`%f` is used for floating-point numbers.

# Using the format() method

formatted_string = "Hello, {}! You are {} years old.".format(name,
age)

print(formatted_string)

# Using f-strings

formatted_string = f"Hello, {name}! You are {age} years old."

print(formatted_string)
```