Python Strings and String Methods

1. Introduction to Strings

In Python, a string is a sequence of characters enclosed within single (`' '`), double (`" "`), or triple quotes (`''' '''` or `""" """`). Strings are immutable, meaning they cannot be changed after creation.

Example:

```
string1 = 'Hello'
string2 = "World"
string3 = '''Triple quotes can span
multiple lines!'''
```

2. Accessing Characters in a String

You can access characters using indexing or slicing.

Example:

```
string = "Python"
print(string[0])  # Output: 'P'
print(string[-1])  # Output: 'n'
print(string[1:4])  # Output: 'yth'
```

3. String Methods

3.1 `len()`

Returns the length of the string.

Example:

```
string = "Python"
print(len(string))  # Output: 6
```

3.2 `lower()`

Converts all characters in a string to lowercase.

Example:

```
string = "HELLO"
print(string.lower())  # Output: 'hello'
```

3.3 `upper()`

Converts all characters in a string to uppercase.

Example:

```
string = "hello"
print(string.upper())  # Output: 'HELLO'
```

3.4 `replace()`

Replaces a substring with another substring.

Example:

```
string = "Hello, World"
new_string = string.replace("World", "Python")
print(new_string)  # Output: 'Hello, Python'
```

3.5 `split()`

Splits a string into a list of substrings based on a delimiter.

Example:

```
string = "apple, banana, cherry"
fruits = string.split(", ")
print(fruits)  # Output: ['apple', 'banana', 'cherry']
```

3.6 `join()`

Joins elements of a list into a string with a specified delimiter.

Example:

```
fruits = ['apple', 'banana', 'cherry']
joined_string = ", ".join(fruits)
print(joined_string)  # Output: 'apple, banana, cherry'
```

3.7 `strip()`

Removes leading and trailing whitespace from a string.

Example:

```
string = "   Hello, World!   "
trimmed_string = string.strip()
print(trimmed_string)  # Output: 'Hello, World!'
```

3.8 `find()`

Returns the index of the first occurrence of a substring. Returns `-1` if not found.

Example:

```
string = "Hello, World"
index = string.find("World")
print(index)  # Output: 7
```

3.9 `startswith()` and `endswith()`

Checks if the string starts or ends with a specific substring.

Example:

```
string = "Hello, World"
print(string.startswith("Hello"))  # Output: True
print(string.endswith("World"))    # Output: True
```

3.10 `count()`

Returns the number of occurrences of a substring.

Example:

```
string = "banana"
count = string.count("a")
print(count)  # Output: 3
```

3.11 `capitalize()`

Capitalizes the first character of the string.

Example:

```
string = "hello"
print(string.capitalize())  # Output: 'Hello'
```

3.12 `title()`

Converts the first character of each word to uppercase.

Example:

```
string = "hello world"
print(string.title())  # Output: 'Hello World'
```

3.13 `isnumeric()`

Checks if all characters in the string are numeric.

Example:

```
string = "12345"
print(string.isnumeric())  # Output: True
```

3.14 `swapcase()`

Converts all uppercase characters to lowercase and vice versa.

Example:

```
string = "Hello World"
print(string.swapcase())  # Output: 'hELLO wORLD'
```

3.15 `zfill()`

Pads the string on the left with zeros to fill the specified width.

Example:

```
string = "42"
print(string.zfill(5))  # Output: '00042'
```

3.16 `ljust()`

Returns a left-justified string of the given width by padding with spaces (or a specified character).

Example:

```
string = "Hello"
print(string.ljust(10, '-'))  # Output: 'Hello-----'
```

3.17 `rjust()`

Returns a right-justified string of the given width by padding with spaces (or a specified character).

Example:

```
string = "Hello"
print(string.rjust(10, '-'))  # Output: '-----Hello'
```

3.18 `center()`

Centers the string with the specified width by padding with spaces (or a specified character).

Example:

```
string = "Hello"
print(string.center(11, '-'))  # Output: '---Hello---'
```

3.19 `partition()`

Splits the string into a tuple at the first occurrence of the separator: (before separator, separator, after separator).

Example:

```
string = "Hello, World!"
result = string.partition(", ")
print(result)  # Output: ('Hello', ', ', 'World!')
```

3.20 `rpartition()`

Splits the string into a tuple at the last occurrence of the separator: (before separator, separator, after separator).

Example:

```
string = "apple, banana, cherry"
result = string.rpartition(", ")
print(result)  # Output: ('apple, banana', ', ', 'cherry')
```

3.21 `expandtabs()`

Replaces tab characters (`\t`) with spaces. You can specify the number of spaces per tab.

Example:

```
string = "Hello\tWorld"
print(string.expandtabs(4))  # Output: 'Hello   World'
```

3.22 `isspace()`

Returns `True` if the string contains only whitespace characters.

Example:

```
string = "   "
print(string.isspace())  # Output: True
```

3.23 `isalpha()`

Returns `True` if all characters in the string are alphabetic (letters).

Example:

```
string = "Python"
print(string.isalpha())  # Output: True
```

3.24 `isalnum()`

Returns `True` if all characters in the string are alphanumeric (letters and digits).

Example:

```
string = "Python3"
print(string.isalnum())  # Output: True
```

3.25 `isdigit()`

Returns `True` if all characters in the string are digits.

Example:

```
string = "12345"
print(string.isdigit())  # Output: True
```

3.26 `isupper()`

Returns `True` if all characters in the string are uppercase.

Example:

```
string = "HELLO"
print(string.isupper())  # Output: True
```

3.27 `islower()`

Returns `True` if all characters in the string are lowercase.

Example:

```
string = "hello"
print(string.islower())  # Output: True
```

3.28 `istitle()`

Returns `True` if the string follows the title case rule, where each word starts with an uppercase letter followed by lowercase letters.

Example:

```
string = "Hello World"
print(string.istitle())  # Output: True
```

3.29 `format()`

Performs string formatting using curly braces `{}` as placeholders.

Example:

```
name = "Alice"
age = 30
print("My name is {} and I am {} years old.".format(name, age))
# Output: 'My name is Alice and I am 30 years old.'
```

3.30 `encode()`

Encodes the string using a specified encoding.

Example:

```
string = "Hello, World!"
encoded_string = string.encode("utf-8")
print(encoded_string)  # Output: b'Hello, World!'
```

3.31 `decode()`

Decodes the encoded string back to its original form.

Example:

```
encoded_string = b'Hello, World!'
decoded_string = encoded_string.decode("utf-8")
print(decoded_string)  # Output: 'Hello, World!'
```

4. String Formatting

Python supports various ways of formatting strings, including `%` formatting, `str.format()`, and f-strings.

4.1 `%` Formatting

```
name = "Alice"
age = 30
print("My name is %s and I am %d years old." % (name, age))
# Output: My name is Alice and I am 30 years old.
```

4.2 `str.format()`

```
name = "Bob"
age = 25
print("My name is {} and I am {} years old.".format(name, age))
# Output: My name is Bob and I am 25 years old.
```

4.3 f-Strings with Format Specifiers

Basic f-string formatting

```
name = "Charlie"
age = 35
print(f"My name is {name} and I am {age} years old.")
# Output: My name is Charlie and I am 35 years old.
```

Floating-point precision

```
pi = 3.141592653589793
print(f"Pi rounded to 3 decimal places: {pi:.3f}")
# Output: Pi rounded to 3 decimal places: 3.142
```

Field width with floating-point numbers

```
number = 12.34
print(f"Number with field width 10: {number:10.2f}")
# Output: Number with field width 10:      12.34
```

Field width with integers

```
number = 123
print(f"Number with field width 5: {number:5}")
# Output: Number with field width 5:   123
```

Left-align with width specifier

```
number = 123
print(f"Left-aligned number: {number:<5}")
# Output: Left-aligned number: 123
```

Right-align with width specifier

```
number = 123
print(f"Right-aligned number: {number:>5}")
# Output: Right-aligned number:   123
```

Center-align with width specifier

```
number = 123
print(f"Center-aligned number: {number:^5}")
# Output: Center-aligned number:  123
```

Sign display for positive and negative numbers

```
pos_number = 25
neg_number = -25
print(f"Positive number with sign: {pos_number:+}")
print(f"Negative number with sign: {neg_number:+}")
# Output: Positive number with sign: +25
#         Negative number with sign: -25
```

Thousands separator

```
large_number = 1000000
print(f"Number with thousands separator: {large_number:,}")
# Output: Number with thousands separator: 1,000,000
```

Binary, octal, hexadecimal formatting

```
number = 255
print(f"Binary: {number:b}")
print(f"Octal: {number:o}")
print(f"Hexadecimal (lowercase): {number:x}")
print(f"Hexadecimal (uppercase): {number:X}")
# Output: Binary: 11111111
#         Octal: 377
#         Hexadecimal (lowercase): ff
#         Hexadecimal (uppercase): FF
```

Percentage formatting

```
percentage = 0.75
print(f"Percentage: {percentage:.2%}")
# Output: Percentage: 75.00%
```

Exponent notation

```
large_number = 123456789
print(f"Exponent notation: {large_number:e}")
# Output: Exponent notation: 1.234568e+08
```

Custom padding with characters

```
number = 42
print(f"Number padded with zeros: {number:05}")
# Output: Number padded with zeros: 00042
```

Using expressions inside f-strings

```
a = 5
b = 10
print(f"Sum of a and b: {a + b}")
# Output: Sum of a and b: 15
```

Including datetime in f-strings

```
from datetime import datetime
now = datetime.now()
print(f"Current date and time: {now:%Y-%m-%d %H:%M:%S}")
# Output: Current date and time: 2024-10-04 12:34:56
```

Formatting lists using f-strings

```
fruits = ["apple", "banana", "cherry"]
print(f"My favorite fruits are: {', '.join(fruits)}.")
# Output: My favorite fruits are: apple, banana, cherry.
```

Using conditional expressions in f-strings

```
is_raining = True
print(f"It's {'raining' if is_raining else 'sunny'} outside.")
# Output: It's raining outside.
```

5. Escape Characters

Escape characters allow you to include special characters in strings.

Common escape characters:

- `\\` - Backslash
- `\'` - Single quote
- `\"` - Double quote

- `\n` - Newline
- `\t` - Tab

Example:

```
string = "Hello\nWorld"
print(string)
# Output:
# Hello
# World
```

Conclusion

Python strings are powerful and versatile, with a wide array of methods to manipulate and format them. Understanding string operations is fundamental to writing effective Python code.