Strings in Python

Introduction

Strings are one of the most fundamental data types in Python. A string is a sequence of characters enclosed within either single quotes (`'`), double quotes (`"`), or triple quotes (`'''` or “””).

Creating Strings

You can create strings in Python using different types of quotes:

```python
# Single-quoted string
a = 'Hello, Python!'

# Double-quoted string
b = "Hello, World!"

# Triple-quoted string (useful for multi-line strings)
c = '''This is
a multi-line
string.'''
```

String Indexing

Each character in a string has an index:

```python
text = "Python"
print(text[0])  # Output: P
print(text[1])  # Output: y
print(text[-1]) # Output: n (last character)
```

String Slicing

You can extract parts of a string using slicing:

```python
text = "Hello, Python!"
print(text[0:5])   # Output: Hello
print(text[:5])    # Output: Hello
print(text[7:])    # Output: Python!
print(text[::2])   # Output: Hlo Pto!
```

String Methods

Python provides several built-in methods to manipulate strings:

```python
text = " hello world "
print(text.upper())      # Output: " HELLO WORLD "
print(text.lower())      # Output: " hello world "
print(text.strip())      # Output: "hello world"
print(text.replace("world", "Python")) # Output: " hello Python "
print(text.split())      # Output: ['hello', 'world']
```

String Formatting

Python offers multiple ways to format strings:

```python
name = "John"
age = 25

# Using format()
print("My name is {} and I am {} years old.".format(name, age))

# Using f-strings (Python 3.6+)
print(f"My name is {name} and I am {age} years old.")
```

Multiline Strings

Triple quotes allow you to create multi-line strings:

```python
message = '''
Hello,
This is a multi-line string example.
Goodbye!
'''
print(message)
```

Summary

- Strings are sequences of characters.
- Use single, double, or triple quotes to define strings.
- Indexing and slicing allow accessing parts of a string.
- String methods help modify and manipulate strings.
- f-strings provide an efficient way to format strings.

String Slicing and Indexing

Introduction

In Python, strings are sequences of characters, and each character has an index. You can access individual characters using indexing and extract substrings using slicing.

String Indexing

Each character in a string has a unique index, starting from 0 for the first character and -1 for the last character.

```python
text = "Python"
print(text[0])  # Output: P
print(text[1])  # Output: y
print(text[-1]) # Output: n (last character)
print(text[-2]) # Output: o
```

String Slicing

Slicing allows you to extract a portion of a string using the syntax `string[start:stop:step]`.

```python
text = "Hello, Python!"
print(text[0:5])   # Output: Hello
print(text[:5])    # Output: Hello (same as text[0:5])
print(text[7:])    # Output: Python! (from index 7 to end)
print(text[::2])   # Output: Hlo Pto!
print(text[-6:-1]) # Output: ython (negative indexing)
```

Step Parameter

The `step` parameter defines the interval of slicing.

```python
text = "Python Programming"
print(text[::2])   # Output: Pto rgamn
print(text[::-1])  # Output: gnimmargorP nohtyP (reverses string)
```

Practical Uses of Slicing

String slicing is useful in many scenarios: - Extracting substrings - Reversing strings - Removing characters - Manipulating text efficiently

```python
text = "Welcome to Python!"
print(text[:7])   # Output: Welcome
print(text[-7:])  # Output: Python!
print(text[3:-3]) # Output: come to Pyt
```

Summary

- Indexing allows accessing individual characters.
- Positive indexing starts from 0, negative indexing starts from -1.
- Slicing helps extract portions of a string.
- The step parameter defines the interval for selection.

- Using `[::-1]` reverses a string.

String Methods and Functions

Introduction

Python provides a variety of built-in string methods and functions to manipulate and process strings efficiently.

Common String Methods

Changing Case

```
text = "hello world"
print(text.upper())  # Output: "HELLO WORLD"
print(text.lower())  # Output: "hello world"
print(text.title())  # Output: "Hello World"
print(text.capitalize())  # Output: "Hello world"
```

Removing Whitespace

```
text = "  hello world   "
print(text.strip())  # Output: "hello world"
print(text.lstrip()) # Output: "hello world   "
print(text.rstrip()) # Output: "  hello world"
```

Finding and Replacing

```
text = "Python is fun"
print(text.find("is"))    # Output: 7
print(text.replace("fun", "awesome"))  # Output: "Python is aweso
```

Splitting and Joining

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

```python
new_text = " - ".join(fruits)
print(new_text)  # Output: "apple - banana - orange"
```

Checking String Properties

```python
text = "Python123"
print(text.isalpha())  # Output: False
print(text.isdigit())  # Output: False
print(text.isalnum())  # Output: True
print(text.isspace())  # Output: False
```

Useful Built-in String Functions

`len()` - Get Length of a String

```python
text = "Hello, Python!"
print(len(text))  # Output: 14
```

`ord()` and `chr()` - Character Encoding

```python
print(ord('A'))  # Output: 65
print(chr(65))   # Output: 'A'
```

`format()` and f-strings

```python
name = "Alice"
age = 30
print("My name is {} and I am {} years old.".format(name, age))
print(f"My name is {name} and I am {age} years old.")
```

Summary

- Python provides various string methods for modification and analysis.

- Case conversion, trimming, finding, replacing, splitting, and joining are commonly used.
- Functions like `len()`, `ord()`, and `chr()` are useful for working with string properties.

String Formatting and f-Strings

Introduction

String formatting is a powerful feature in Python that allows you to insert variables and expressions into strings in a structured way. Python provides multiple ways to format strings, including the older `.format()` method and the modern `f-strings`.

Using `.format()` Method

The `.format()` method allows inserting values into placeholders `{}`:

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

You can also specify positional and keyword arguments:

```
print("{1} is learning {0}".format("Python", "Alice"))  # Output:
print("{name} is {age} years old".format(name="Bob", age=25))
```

f-Strings (Formatted String Literals)

Introduced in Python 3.6, f-strings are the most concise and readable way to format strings:

```
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

Using Expressions in f-Strings

You can perform calculations directly inside f-strings:

```python
x = 10
y = 5
print(f"The sum of {x} and {y} is {x + y}")
```

Formatting Numbers

```python
pi = 3.14159265
print(f"Pi rounded to 2 decimal places: {pi:.2f}")
```

Padding and Alignment

```python
text = "Python"
print(f"{text:>10}")  # Right align
print(f"{text:<10}")  # Left align
print(f"{text:^10}")  # Center align
```

Important Notes

> - **Escape Sequences**: Use `\n`, `\t`, `\'`, `\"`, and `\\` to handle special characters in strings.
> - **Raw Strings**: Use `r"string"` to prevent escape sequence interpretation.
> - **String Encoding & Decoding**: Use `.encode()` and `.decode()` to work with different text encodings.
> - **String Immutability**: Strings in Python are immutable, meaning they cannot be changed after creation.
> - **Performance Considerations**: Using `''.join(list_of_strings)` is more efficient than concatenation in loops.

Summary

- `.format()` allows inserting values into placeholders.
- f-strings provide an intuitive and readable way to format strings.
- f-strings support expressions, calculations, and formatting options.