Strings: String Manipulation, Basic operations, Slicing, Built in String Functions

Strings in Python are sequences of characters, represented within either single quotes (`'`) or double quotes (`"`). Python treats single quotes and double quotes interchangeably for defining strings, so `'hello'`, `"hello"`, and `"'hello'"` are all equivalent.

Here are some key points about strings in Python:

1. **Immutable**: Strings in Python are immutable, meaning they cannot be changed after they are created. You cannot modify a string's contents in-place; instead, you create a new string.

2. **Indexing and Slicing**: You can access individual characters of a string using indexing, where the first character has an index of 0. Slicing allows you to extract substrings by specifying a range of indices.

```python
my_string = "Python"
print(my_string[0])     # Output: 'P'
print(my_string[2:5])   # Output: 'tho'
```

3. **String Concatenation**: You can concatenate strings using the `+` operator or by using string formatting.

```python
str1 = "Hello"
str2 = "world"
result = str1 + " " + str2
print(result)  # Output: Hello world

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

```

4. **Escape Characters**: Python supports various escape characters like `\n` for newline, `\t` for tab, `\b` for backspace, etc.

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

5. **String Methods**: Python provides a rich set of methods for string manipulation, such as `split()`, `join()`, `replace()`, `find()`, `upper()`, `lower()`, `strip()`, `startswith()`, `endswith()`, etc.

```python
my_string = "Hello, World!"
print(my_string.upper())              # Output: HELLO, WORLD!
print(my_string.replace("Hello", ""))  # Output: , World!
```

6. **Raw Strings**: Prefixing a string literal with `r` or `R` makes it a raw string, which treats backslashes as literal characters.

```python
path = r'C:\new\text.txt'
print(path)  # Output: C:\new\text.txt
```

These are just some of the basics of working with strings in Python. Strings are fundamental data types in Python and are extensively used in various applications.

String manipulation in Python involves various operations such as concatenation, slicing, formatting, searching, replacing, and more. Here's a brief overview of some common string manipulation techniques in Python:
```

1. **Concatenation**: Combining two or more strings together using the `+` operator.

   ```python
   str1 = "Hello"
   str2 = "world"
   result = str1 + " " + str2
   print(result)  # Output: Hello world
   ```

2. **Slicing**: Extracting substrings from a string using indices.

   ```python
   string = "Python is awesome"
   substring = string[7:10]
   print(substring)  # Output: is
   ```

3. **Formatting**: Constructing strings with formatted values.

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

4. **Searching**: Finding substrings within a string.

   ```python
   string = "Python is easy to learn"
   if "easy" in string:
       print("Substring found")
   ```

5. **Replacing**: Replacing occurrences of a substring within a string.

   ```python
   ```

```
string = "Python is easy to learn"
new_string = string.replace("easy", "simple")
print(new_string)  # Output: Python is simple to learn
```

6. **Splitting**: Breaking a string into a list of substrings.
```python
string = "Python,Java,C++,JavaScript"
languages = string.split(",")
print(languages)  # Output: ['Python', 'Java', 'C++', 'JavaScript']
```

7. **Stripping**: Removing leading and trailing whitespace characters.
```python
string = "   Hello, world!   "
stripped_string = string.strip()
print(stripped_string)  # Output: Hello, world!
```

8. Upper/Lower Case Conversion**: Converting a string to upper or lower case.
```python
string = "Hello, World!"
upper_case = string.upper()
lower_case = string.lower()
print(upper_case)  # Output: HELLO, WORLD!
print(lower_case)  # Output: hello, world!
```

3333333

Basic string operations in Python include various tasks such as concatenation, repetition, indexing, slicing, length determination, and more. Here are some common string operations in Python:

1. **Concatenation**: Combining two or more strings together using the `+` operator.

   ```python
   str1 = "Hello"
   str2 = "World"
   result = str1 + ", " + str2
   print(result)  # Output: Hello, World
   ```

2. **Repetition**: Repeating a string multiple times using the `*` operator.

   ```python
   str1 = "Hello"
   repeated_str = str1 * 3
   print(repeated_str)  # Output: HelloHelloHello
   ```

3. **Indexing**: Accessing individual characters of a string using indices.

   ```python
   my_string = "Python"
   print(my_string[0])  # Output: P
   ```

4. **Slicing**: Extracting substrings by specifying a range of indices.

   ```python
   my_string = "Python"
   print(my_string[2:5])  # Output: tho
   ```

5. **Length**: Determining the length of a string using the `len()` function.

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

6. **Membership Test**: Checking if a substring exists within a string using the `in` keyword.

```python
my_string = "Python"
if "th" in my_string:
    print("Substring found")
```

7. **String Formatting**: Constructing strings with formatted values using `%` formatting or the `format()` method.

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

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

8. **Upper/Lower Case Conversion**: Converting a string to upper or lower case using the `upper()` and `lower()` methods.

```python
```

```
my_string = "Hello, World!"
print(my_string.upper())  # Output: HELLO, WORLD!
print(my_string.lower())  # Output: hello, world!
```

These are some of the basic string operations in Python that you'll commonly encounter when working with strings.

44444

Slicing in Python allows you to extract a substring from a string by specifying a range of indices. The syntax for slicing is `string[start:stop:step]`, where:

- `start` is the starting index of the slice (inclusive).
- `stop` is the ending index of the slice (exclusive).
- `step` is the step size, which determines how many characters to skip (optional, defaults to 1).

Here are some examples of slicing in Python:

1. **Basic slicing**:

```python
my_string = "Hello, World!"
sliced_string = my_string[1:5]  # Get characters from index 1 to 4
print(sliced_string)  # Output: "ello"
```

2. **Negative indices**:

Negative indices count from the end of the string. `-1` refers to the last character, `-2` refers to the second last character, and so on.

```python
my_string = "Hello, World!"
sliced_string = my_string[-6:-1]  # Get characters from the 6th last to the 2nd last
print(sliced_string)  # Output: "World"
```

3. **Omitting indices**:

   If you omit `start`, `stop`, or both, Python will assume default values. If `start` is omitted, it defaults to the beginning of the string. If `stop` is omitted, it defaults to the end of the string.

   ```python
   my_string = "Hello, World!"
   print(my_string[:5])   # Output: "Hello"
   print(my_string[7:])   # Output: "World!"
   print(my_string[:])    # Output: "Hello, World!"
   ```

4. **Using step**:

   You can specify a step value to skip characters while slicing.

   ```python
   my_string = "Hello, World!"
   sliced_string = my_string[::2]  # Get every second character
   print(sliced_string)  # Output: "Hlo ol!"
   ```

5. **Reversing a string**:

   Slicing with a step of `-1` can be used to reverse a string.

   ```python
   my_string = "Hello, World!"
   reversed_string = my_string[::-1]
   print(reversed_string)  # Output: "!dlroW ,olleH"
   ```

Slicing is a powerful feature in Python that allows you to efficiently work with substrings. It's commonly used for tasks like extracting specific parts of a string, manipulating text data, and more.

5555

Python offers a wide range of built-in string functions that allow you to manipulate and work with strings efficiently. Here's a list of some commonly used built-in string functions in Python:

1. `len(str)`: Returns the length of the string.

```python
my_string = "Hello, World!"
print(len(my_string))  # Output: 13
```

2. `str.upper()`, `str.lower()`: Converts the string to upper or lower case.

```python
my_string = "Hello, World!"
print(my_string.upper())  # Output: HELLO, WORLD!
print(my_string.lower())  # Output: hello, world!
```

3. `str.strip()`: Removes leading and trailing whitespaces from the string.

```python
my_string = "   Hello, World!   "
print(my_string.strip())  # Output: Hello, World!
```

4. `str.startswith(prefix)`, `str.endswith(suffix)`: Checks if the string starts or ends with the specified prefix or suffix.

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

5. `str.find(substring)`, `str.index(substring)`: Returns the lowest index where the substring is found. `find()` returns -1 if the substring is not found, while `index()` raises a ValueError.

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

6. `str.replace(old, new)`: Replaces occurrences of a substring with another substring.

```python
my_string = "Hello, World!"
print(my_string.replace("World", "Python"))  # Output: Hello, Python!
```

7. `str.split(separator)`: Splits the string into a list of substrings based on the specified separator.

```python
my_string = "Hello, World!"
print(my_string.split(","))  # Output: ['Hello', ' World!']
```

8. `str.join(iterable)`: Joins elements of an iterable (e.g., a list) into a single string, with the string as a separator.

```python
my_list = ['Hello', 'World']
print("-".join(my_list))  # Output: Hello-World
```

These are just a few examples of built-in string functions in Python. Python's standard library offers many more functions for string manipulation and formatting.