Python Class 11 Series

Strings - Chapter: 8

Saroj Kumar Jha

Saroj Codes

Please subscribe

INTRODUCTION

In Python, a string is a sequence of characters. It is used to represent textual data and is one of the built-in data types in the language. Strings in Python are immutable, meaning their values cannot be changed after they are created.

You can create a string in Python by enclosing a sequence of characters within either single quotes (' '), double quotes (" "), or triple quotes (''' ''' or """ """). Here are some examples:

```
# Using single quotes
single_quoted_string = 'Hello, world!'

# Using double quotes
double_quoted_string = "This is a string."

# Using triple quotes for multiline strings
multiline_string = '''This is a multiline string.
It can span across multiple lines.'''
```

Accessing Characters in a String

In Python, you can access individual characters in a string using indexing. The indexing in Python starts at 0, so the first character of a string is at index 0, the second character is at index 1, and so on. You can also use negative indexing, where -1 refers to the last character, -2 refers to the second-to-last character, and so on.

Here are some examples of accessing characters in a string:

```
text = "Python"

# Accessing individual characters
first_character = text[0]      # 'P'
second_character = text[1]     # 'y'
last_character = text[-1]      # 'n'
second_last_character = text[-2]  # 'o'
```

Accessing Characters in a String

```
# Slicing to access a substring of characters
substring = text[1:4]    # 'yth'
substring_from_start = text[:3]    # 'Pyt'
substring_to_end = text[3:]      # 'hon'

# Accessing characters with step in slicing
step_example = text[::2]      # 'Pto' (Every second character)

# Reverse the string using slicing
reversed_text = text[::-1]   # 'nohtyP’
```

Keep in mind that string objects in Python are immutable, so you can access individual characters or a substring, but you cannot modify them directly. If you want to modify or concatenate strings, you'll need to create a new string with the desired changes.

Accessing Characters in a String

Positive Indices	0	1	2	3	4	5
String	P	y	t	h	o	n
Negative Indices	-6	-5	-4	-3	-2	-1

In Python, strings are immutable, which means once a string is created, its contents cannot be changed. This immutability property is one of the key characteristics of strings in Python.

When you perform operations on strings, such as slicing, concatenation, or converting the case, it appears as if you are modifying the string. However, what actually happens is that new string objects are created with the modified content, while the original string remains unchanged.

For example:

```
text = "Hello"

# This will create a new string "Hello, World" and assign it to the variable "new_text"
new_text = text + ", World"

# This will create a new string "HELLO" and assign it to the variable "uppercase_text"
uppercase_text = text.upper()
```

Accessing Characters in a String

```
# The original string "text" remains unchanged
print(text)  # Output: "Hello“
```

Since strings are immutable, Python's memory management becomes more efficient when working with strings because the interpreter can safely reuse existing string objects knowing they won't change. It also prevents accidental modification of strings and helps ensure the consistency of data throughout the program. If you need to modify a string, you'll need to create a new string with the desired changes.

STRING OPERATIONS

In Python, strings support a wide range of operations for manipulation and processing. Here are some common string operations you can perform:

Concatenation: Joining two or more strings together.

```
string1 = "Hello"
string2 = "World"
result = string1 + " " + string2  # "Hello World"
```

Slicing: Extracting a substring from a string using index ranges.

```
text = "Python is awesome"
substring = text[0:6]  # "Python"
```

Length retrieval: Getting the number of characters in a string.

```
text = "Hello, World!"
length = len(text)  # 13
```

String formatting: Constructing formatted strings using placeholders.

```
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
# "My name is Alice and I am 30 years old."
```

Case conversion: Changing the case of the characters in a string.

```
text = "Hello, World!"
uppercase_text = text.upper()  # "HELLO, WORLD!"
lowercase_text = text.lower()  # "hello, world!"
```

STRING OPERATIONS

Stripping: Removing leading and trailing whitespace characters from a string.

```
text = "    Hello, World!    "
trimmed_text = text.strip()  # "Hello, World!“
```

Replace: Replacing occurrences of a substring with another substring.

```
text = "Hello, World!"
replaced_text = text.replace("Hello", "Hi")  # "Hi, World!“
```

Split: Splitting a string into a list of substrings based on a delimiter.

```
text = "apple,orange,banana"
fruits_list = text.split(",")  # ['apple', 'orange', 'banana’]
```

Join: Joining a list of strings into a single string using a delimiter.

```
fruits_list = ['apple', 'orange', 'banana']
text = ",".join(fruits_list)  # "apple,orange,banana“
```

Check for substrings: Checking if a substring exists in a string.

```
text = "Hello, World!"
contains_hello = "Hello" in text  # True
contains_hi = "Hi" in text  # False
```

STRING OPERATIONS

In Python, the "membership" operator allows you to check if a particular element exists within a sequence or collection. For strings, it helps you determine if a substring is present within the main string. The membership operator for strings is the "in" keyword.

Here's how you can use the "in" operator for membership checking in strings:

```
text = "Hello, World!"

# Check if a substring exists in the string
contains_hello = "Hello" in text  # True
contains_hi = "Hi" in text  # False
```

In Python, you can repeat a string multiple times by using the "repetition" operator, which is denoted by the asterisk (*) symbol. This allows you to create a new string by repeating the original string a specified number of times.

Here's how you can use the repetition operator to repeat a string:

```
text = "Hello"

# Repeat the string three times
repeated_text = text * 3
print(repeated_text)  # Output: "HelloHelloHello"
```

TRAVERSING A STRING

Traversing a string in Python refers to iterating or looping through each character of the string. You can use various methods to traverse a string, such as using a for loop, while loop, or even using the built-in enumerate() function.

Here are some examples of traversing a string:

Using a for loop:

```
text = "Hello, World!"
for char in text:
    print(char)
```

Using a while loop with index:

```
text = "Hello, World!"
index = 0
while index < len(text):
    print(text[index])
    index += 1
```

Using enumerate() to get both index and character:

```
text = "Hello, World!"

for index, char in enumerate(text):
    print(f"Character at index {index}: {char}")
```

STRING METHODS AND BUILT-IN FUNCTIONS

In Python, strings come with a variety of built-in methods and functions that allow you to perform various operations on strings. Here are some commonly used string methods and built-in functions:

String Methods:

str.upper(): Returns a new string with all characters in uppercase.
str.lower(): Returns a new string with all characters in lowercase.
str.strip(): Returns a new string with leading and trailing whitespace removed.
str.replace(old, new): Returns a new string with all occurrences of old replaced by new.
str.split(sep): Splits the string into a list of substrings based on the separator sep.
str.join(iterable): Joins the elements of an iterable (e.g., list) into a single string using the string as a separator.
str.startswith(prefix): Returns True if the string starts with the specified prefix.
str.endswith(suffix): Returns True if the string ends with the specified suffix.
str.isalpha(): Returns True if all characters in the string are alphabetic.
str.isdigit(): Returns True if all characters in the string are digits.
str.isalnum(): Returns True if all characters in the string are alphanumeric.
str.islower(): Returns True if all characters in the string are lowercase.
str.isupper(): Returns True if all characters in the string are uppercase.
str.count(substring): Returns the number of occurrences of substring in the string.
str.find(substring): Returns the index of the first occurrence of substring in the string or -1 if not found.

STRING METHODS AND BUILT-IN FUNCTIONS

Example:

```
text = "   Hello, World!   "
print(text.upper())  # "   HELLO, WORLD!   "
print(text.strip())  # "Hello, World!"
print(text.replace("Hello", "Hi"))  # "   Hi, World!   "
print(text.split(","))  # ['   Hello', ' World!   ']
print(" ".join(["Python", "is", "great!"]))  # "Python is great!"
print(text.startswith("   "))  # True
print(text.endswith("   "))  # True
print(text.isalpha())  # False
print(text.isdigit())  # False
print(text.count("l"))  # 3
print(text.find("World"))  # 10
```

Built-in Functions:

len(str): Returns the number of characters in the string.
str() or str(object): Converts the specified object to a string.
ord(char): Returns the Unicode code point for the given Unicode character.
chr(code) or unichr(code) (Python 2 only): Returns the character for the given Unicode code point.
ascii(str): Returns a string containing ASCII-only versions of non-ASCII characters.
format(): Formats a string using placeholders and positional or keyword arguments.
input(prompt): Reads a line from input and returns it as a string.

STRING METHODS AND BUILT-IN FUNCTIONS

Example:

```
text = "Hello, World!"
print(len(text))  # 13
print(ord('A'))   # 65
print(chr(65))    # 'A'
print(ascii("é"))  # '\xe9'
print(format(42, 'd'))  # '42'
user_input = input("Enter your name: ")  # Read user input as a string
```

Exercise

Q. Write a program with a user defined function to count the number of times a character (passed as argument) occurs in the given string

```
def count_char_in_string(input_string, character):
    count = 0
    for char in input_string:
        if char == character:
            count += 1
    return count

def main():
    input_string = input("Enter a string: ")
    char_to_count = input("Enter the character to count: ")

    # Make sure the input is a single character
    if len(char_to_count) != 1:
        print("Please enter a single character to count.")
        return

    count = count_char_in_string(input_string, char_to_count)
    print(f"The character '{char_to_count}' appears {count} times in the string.")

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program with a user defined function with string as a parameter which replaces all vowels in the string with '*'.

```
def replace_vowels_with_asterisk(input_string):
    vowels = "AEIOUaeiou"
    result = ""
    for char in input_string:
        if char in vowels:
            result += '*'
        else:
            result += char
    return result

def main():
    input_string = input("Enter a string: ")
    new_string = replace_vowels_with_asterisk(input_string)
    print("String with vowels replaced by asterisk:", new_string)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program to input a string from the user and print it in the reverse order without creating a new string.

```
def reverse_string_in_place(input_string):
    length = len(input_string)

    for index in range(length - 1, -1, -1):
        print(input_string[index], end='')

def main():
    input_string = input("Enter a string: ")
    print("String in reverse order: ", end='')
    reverse_string_in_place(input_string)
    print()  # To add a new line after printing the reversed string

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program using a user defined function to check if a string is a palindrome or not. (A string is called palindrome if it reads same backwards as forward. For example, Madam is a palindrome.)

```
def is_palindrome(input_string):
    # Remove any spaces and convert the input string to lowercase
    input_string = input_string.replace(" ", "").lower()

    # Check if the string is equal to its reverse
    return input_string == input_string[::-1]

def main():
    input_string = input("Enter a string: ")

    if is_palindrome(input_string):
        print("The string is a palindrome.")
    else:
        print("The string is not a palindrome.")

if __name__ == "__main__":
    main()
```

Exercise

Q. Consider the following string mySubject:
mySubject = "Computer Science"
What will be the output of the following string operations :

i. print(mySubject[0:len(mySubject)])
ii. print(mySubject[-7:-1])
iii. print(mySubject[::2])
iv. print(mySubject[len(mySubject)-1])
v. print(2*mySubject)
vi. print(mySubject[::-2])
vii. print(mySubject[:3] + mySubject[3:])
viii. print(mySubject.swapcase())
ix. print(mySubject.startswith('Comp'))
x. print(mySubject.isalpha())

```
mySubject = "Computer Science"

# i. print(mySubject[0:len(mySubject)])
print(mySubject[0:len(mySubject)])  # Output: "Computer Science"

# ii. print(mySubject[-7:-1])
print(mySubject[-7:-1])  # Output: " Scienc"
```

Exercise

```
# iii. print(mySubject[::2])
print(mySubject[::2])  # Output: "Cmue cec"

# iv. print(mySubject[len(mySubject)-1])
print(mySubject[len(mySubject)-1])  # Output: "e"

# v. print(2*mySubject)
print(2*mySubject)  # Output: "Computer ScienceComputer Science"

# vi. print(mySubject[::-2])
print(mySubject[::-2])  # Output: "eciemtuo"

# vii. print(mySubject[:3] + mySubject[3:])
print(mySubject[:3] + mySubject[3:])  # Output: "Computer Science"

# viii. print(mySubject.swapcase())
print(mySubject.swapcase())  # Output: "cOMPUTER sCIENCE"

# ix. print(mySubject.startswith('Comp'))
print(mySubject.startswith('Comp'))  # Output: True

# x. print(mySubject.isalpha())
print(mySubject.isalpha())  # Output: False
```

Exercise

Explanation:

i. Slicing from index 0 to the length of the string returns the entire string "Computer Science".
ii. Slicing from index -7 to -1 (exclusive) returns " Scienc" (note the space before "S").
iii. Slicing with a step of 2 returns every second character of the string.
iv. Indexing with len(mySubject)-1 returns the last character, which is "e".
v. Multiplying the string by 2 duplicates the string.
vi. Slicing with a negative step of 2 returns the string in reverse order, skipping every second character.
vii. Concatenating the slices mySubject[:3] ("Com") and mySubject[3:] ("puter Science") results in the original string.
viii. swapcase() reverses the case of each character in the string.
ix. startswith('Comp') checks if the string starts with "Comp" and returns True.
x. isalpha() checks if all characters in the string are alphabetic and returns False because of the space in the string.

Exercise

Q. Consider the following string myAddress:
myAddress = "WZ-1,New Ganga Nagar,New Delhi"
What will be the output of following string operations :

i. print(myAddress.lower())
ii. print(myAddress.upper())
iii. print(myAddress.count('New'))
iv. print(myAddress.find('New'))
v. print(myAddress.rfind('New'))
vi. print(myAddress.split(','))
vii. print(myAddress.split(' '))
viii. print(myAddress.replace('New','Old'))
ix. print(myAddress.partition(','))
x. print(myAddress.index('Agra'))

```
myAddress = "WZ-1,New Ganga Nagar,New Delhi"

# print(myAddress.lower())
print(myAddress.lower())  # Output: "wz-1,new ganga nagar,new delhi"

# print(myAddress.upper())
print(myAddress.upper())  # Output: "WZ-1,NEW GANGA NAGAR,NEW DELHI"
```

Exercise

```
# print(myAddress.count('New'))
print(myAddress.count('New'))  # Output: 2

# print(myAddress.find('New'))
print(myAddress.find('New'))  # Output: 6

# print(myAddress.rfind('New'))
print(myAddress.rfind('New'))  # Output: 23

# print(myAddress.split(','))
print(myAddress.split(','))  # Output: ['WZ-1', 'New Ganga Nagar', 'New Delhi']

# print(myAddress.split(' '))
print(myAddress.split(' '))  # Output: ['WZ-1,New', 'Ganga', 'Nagar,New', 'Delhi']

# print(myAddress.replace('New', 'Old'))
print(myAddress.replace('New', 'Old'))  # Output: "WZ-1,Old Ganga Nagar,Old Delhi"
# print(myAddress.partition(','))
print(myAddress.partition(','))  # Output: ('WZ-1', ',', 'New Ganga Nagar,New Delhi')

# print(myAddress.index('Agra'))
# Since 'Agra' is not present in myAddress, this will raise a ValueError.
# Output: ValueError: substring not found
```

Exercise

Explanation: myAddress = "WZ-1,New Ganga Nagar,New Delhi"

lower(): Converts the string to lowercase.
upper(): Converts the string to uppercase.
count('New'): Counts the occurrences of the substring 'New' in the string, which is 2.
find('New'): Finds the index of the first occurrence of 'New' in the string, which is 6.
rfind('New'): Finds the index of the last occurrence of 'New' in the string, which is 23.
split(','): Splits the string into a list using comma (',') as the separator.
split(' '): Splits the string into a list using space (' ') as the separator.
replace('New', 'Old'): Replaces all occurrences of 'New' with 'Old' in the string.
partition(','): Splits the string into a tuple using the first occurrence of comma (',') as the separator.
index('Agra'): Raises a ValueError since 'Agra' is not present in myAddress.

Exercise

Q. Write a program to input line(s) of text from the user until enter is pressed. Count the total number of characters in the text (including white spaces),total number of alphabets, total number of digits, total number of special symbols and total number of words in the given text. (Assume that each word is separated by one space).

```
import string

def count_characters(text):
    total_characters = len(text)
    total_alphabets = sum(1 for char in text if char.isalpha())
    total_digits = sum(1 for char in text if char.isdigit())
    total_special_symbols = sum(1 for char in text if char in string.punctuation)
    words = text.split()
    total_words = len(words)

    return total_characters, total_alphabets, total_digits, total_special_symbols, total_words
def main():
    print("Enter lines of text. Press Enter without typing anything to stop.")

    total_characters = 0
    total_alphabets = 0
    total_digits = 0
    total_special_symbols = 0
    total_words = 0
```

Exercise

```
    while True:
        line = input()
        if line == "":
            break

        characters, alphabets, digits, special_symbols, words = count_characters(line)
        total_characters += characters
        total_alphabets += alphabets
        total_digits += digits
        total_special_symbols += special_symbols
        total_words += words

    print(f"Total number of characters: {total_characters}")
    print(f"Total number of alphabets: {total_alphabets}")
    print(f"Total number of digits: {total_digits}")
    print(f"Total number of special symbols: {total_special_symbols}")
    print(f"Total number of words: {total_words}")

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a user defined function to convert a string with more than one word into title case string where string is passed as parameter. (Title case means that the first letter of each word is capitalised)

```
def convert_to_title_case(input_string):
    # Split the input string into words using whitespace as the separator
    words = input_string.split()

    # Capitalize the first letter of each word and convert the rest of the letters to lowercase
    title_case_words = [word.capitalize() for word in words]

    # Join the title-cased words back into a single string
    title_case_string = ' '.join(title_case_words)

    return title_case_string

def main():
    input_string = input("Enter a string with more than one word: ")
    title_case_string = convert_to_title_case(input_string)
    print("Title case string:", title_case_string)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a function deleteChar() which takes two parameters one is a string and other is a character. The function should create a new string after deleting all occurrences of the character from the string and return the new string.

```
def deleteChar(input_string, char_to_delete):
    # Use a list comprehension to build the new string without the specified character
    new_string = ''.join(char for char in input_string if char != char_to_delete)
    return new_string

def main():
    input_string = input("Enter a string: ")
    char_to_delete = input("Enter the character to delete: ")

    if len(char_to_delete) != 1:
        print("Please enter a single character to delete.")
        return

    new_string = deleteChar(input_string, char_to_delete)
    print("New string after deletion:", new_string)

if __name__ == "__main__":
    main()
```

Exercise

Q. Input a string having some digits. Write a function to return the sum of digits present in this string.

```
def sum_of_digits_in_string(input_string):
    total_sum = 0
    for char in input_string:
        if char.isdigit():
            total_sum += int(char)
    return total_sum

def main():
    input_string = input("Enter a string with some digits: ")
    digit_sum = sum_of_digits_in_string(input_string)
    print("Sum of digits in the string:", digit_sum)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a function that takes a sentence as an input parameter where each word in the sentence is separated by a space. The function should replace each blank with a hyphen and then return the modified sentence.

```
def replace_spaces_with_hyphens(sentence):
    modified_sentence = sentence.replace(' ', '-')
    return modified_sentence

def main():
    input_sentence = input("Enter a sentence: ")
    modified_sentence = replace_spaces_with_hyphens(input_sentence)
    print("Modified sentence:", modified_sentence)

if __name__ == "__main__":
    main()
```

Thank you

Saroj Kumar Jha

srojkrjha@gmail.com