Strings in Python

Anab Batool Kazmi

Strings in Python

- In Python, strings are sequences of characters enclosed in either single quotes (‘) or double quotes (") or three double quotes (""") or three single quotes (‘‘‘).
- Strings are **immutable,** which means you cannot change their content after they are created. To create a modified string, you generate a new one with the desired changes.
- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.

Strings in Python

```
string1 = 'This is a string.'
string2 = "This is another string."
string3 = """
This is a
multi-line
string.
"""
string4 = '''
This is a
multi-line
string.
'''
string5 = '''This is a string.'''
print(string1)
print(string2)
print(string3)
print(string4)
print(string5)
```

```
This is a string.
This is another string.

This is a
multi-line
string.


This is a
multi-line
string.

This is a string.
```

- In multiline string the line breaks are inserted at the same position as in the code.

Create Empty String

Using str() constructor

```
#CREATE EMPTY String Using str() Constructor:
import sys

my_string = str()
print("Type of my_string data structure is :",type(my_string))
print("Length of n my_string are :",len(my_string))
print(f"Memory reserved by my_string is {sys.getsizeof(my_string)} bytes")
print(f"Memory address of my_string is {id(my_string)}")
```

```
Type of my_string data structure is : <class 'str'>
Length of n my_string are : 0
Memory reserved by my_string is 49 bytes
Memory address of my_string is 1569396062896
```

Using quotes

```
#CREATE EMPTY String Using double quote:
import sys

my_string1 = ""
print("Type of my_string data structure is :",type(my_string1))
print("Length of n my_string are :",len(my_string1))
print(f"Memory reserved by my_string is {sys.getsizeof(my_string1)} bytes")
print(f"Memory address of my_string is {id(my_string1)}")
```

```
Type of my_string data structure is : <class 'str'>
Length of n my_string are : 0
Memory reserved by my_string is 49 bytes
Memory address of my_string is 1569396062896
```

Access string characters

Indexing

- To access a specific character in a string, you can use square brackets [] with the index of the character you want to access.
- Both positive and negative characters can be used to access string characters

Slicing

- You can extract a substring from a string using slicing. Slicing allows you to specify a range of characters to extract, given by the start and end positions (non-inclusive).
- Slicing can also include a step value to skip characters.

Access string characters-Indexing

```python
# access string elements using indexing
text = "Programming for AI using Python"
first_character = text[0]  # Access the first character (P)
print(first_character)
third_character = text[2]  # Access the third character (0)
print(third_character)
character_space = text[11]  # Access the character space
print(character_space)
last_character = text[-1]  # Access the last character (n)
print(last_character)
second_to_last = text[-2]  # Access the second-to-last character (o)
print(second_to_last)
```

```
P
o

n
o
```

Access string characters-Slicing

```python
# access string elements using slicing
text = "Python"
substring = text[1:4]  # Extracts characters at positions 1, 2, and 3 (yth)
print(substring)
substring1 = text[-3:-1]  # Extracts characters at positions 1, 2, and 3 (ho)
print(substring1)
every_second_character = text[::2]  # Extracts every second character (Pto)
print(every_second_character)
```

```
yth
ho
Pto
```

Loop through string characters

- To loop through the characters in a string in Python, you can use a for loop or a while loop.
- In Python, **strings are iterable**, which means you can iterate through their characters one by one.

```
#Loop through string
text = "Python"
for char in text:
    print(char)
```

```
P
y
t
h
o
n
```

```
#Loop through string
text = "Python"
for index in range(len(text)):
    print(text[index])
    index += 1
```

```
P
y
t
h
o
n
```

```
text = "Python"
index = 0

while index < len(text):
    print(text[index])
    index += 1
```

```
P
y
t
h
o
n
```

Check String – in keyword or not in keyword

- To check if a certain phrase or **character is present in a string**, we can use the **keyword in**
- To check if a certain **phrase or character is NOT present in a string**, we can use the **keyword not in**.

```
text = "Programming for AI using Python"
if "AI us" in text:
  print("Yes, 'AI us' is present.")
```

```
Yes, 'AI us' is present.
```

```
text = "Programming for AI using Python"
print("java" not in txt)
print("java" in txt)
```

```
True
False
```

String Concatenation

- String concatenation in Python involves combining two or more strings to create a single, longer string. You can concatenate strings using various methods, and it's a common operation when working with text data.
- Using the + Operator
- Using String Formatting
- Using Join with Lists

String Concatenation- Using the + Operator

- You can concatenate strings using the + operator.
- The + operator allows you to combine multiple strings into one.

```
first_name = "Anab"
last_name = "Kazmi"
full_name = first_name + " " + last_name
print(full_name)
print(first_name + last_name)
```

```
Anab Kazmi
AnabKazmi
```

String Concatenation- Using String Formatting

- You can use string formatting to concatenate strings, which provides more control over the final output.

```
first_name = "Anab"
last_name = "Kazmi"
full_name = "{} {}".format(first_name, last_name)
print(full_name)
full_name1 = f"{first_name} {last_name}"
print(full_name1)
```

```
Anab Kazmi
Anab Kazmi
```

String Concatenation- Using Join with Lists

- You can use the **join() method to concatenate a list of strings**. This is particularly useful when you have many strings to concatenate.
- The join() method is used to concatenate a sequence of strings (e.g., a list or tuple of strings) with a specified delimiter between them, resulting in a single string.

sub_string=delimiter.join(sequence)

- **delimiter**: This is the string that you want to insert between the elements of the sequence. It specifies how the elements should be separated in the resulting string.
- **sequence**: This is the sequence of strings that you want to concatenate with the specified delimiter. It can be a **list, tuple, or any other iterable containing strings.**

String Concatenation- Using Join with Lists

```
words = ["Hello", "world", "!"]
sentence = " ".join(words)
print(sentence)
print(type(sentence))
```

```
Hello world !
<class 'str'>
```

```
words = ["Hello", "world", "!"]
delimiter = "[]"
sentence = delimiter.join(words)
print(sentence)
```

```
Hello[]world[]!
```

String Concatenation- Using String Repetition

- You can concatenate multiple copies of a string using the * operator.

```
repeated_text = "Hello" * 3
print(repeated_text)
```

```
HelloHelloHello
```

String Split-using the split() method

- The split() method splits a string into a list.

Syntax

List1=string.split(separator, maxsplit)

- **separator**
  - Optional. Specifies the separator to use when splitting the string.
  - By default any whitespace is a separator
- **maxsplit**
  - Optional. Specifies how many splits to do.
  - Default value is -1, which is "all occurrences"

String Split-using the split() method

```
text = "This is a sample sentence."
word_list = text.split()  # Split by whitespace (default)
print(word_list)
parts = text.split(" ", 2)  # Split into 3 parts at most
print(parts)
```

```
['This', 'is', 'a', 'sample', 'sentence.']
['This', 'is', 'a sample sentence.']
```

```
data = "apple,banana,grape,kiwi"
fruits = data.split("a")  # Split by a
print(fruits)
```

```
['', 'pple,b', 'n', 'n', ',gr', 'pe,kiwi']
```

String Split into characters

Using List Comprehension

```
text = "Python"
char_list = [char for char in text]
print(char_list)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

Using Type Casting

```
text = "Python"
char_list = list(text)
print(char_list)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

Escape Characters

- String escape characters, also known as escape sequences, are **special sequences of characters used within strings to represent certain non-printable or special characters.**
- They begin with a backslash (\) followed by a character or combination of characters.
- Escape characters are commonly used to represent characters like **newline, tab, backslash**, and others.

Escape Characters

- \n – Newline

Represents a newline character, causing text to start on a new line.

```
text = "Line 1\nLine 2"
print(text)
```

```
Line 1
Line 2
```

- \t – Tab

Represents a tab character, which creates horizontal spacing.

```
text = "Tabbed\ttext"
print(text)
```

```
Tabbed	text
```

Escape Characters

- \\ - Backslash:

Represents a single backslash character. This is used to escape a backslash itself.

```
text = "This is a backslash: \\"
print(text)
```

```
This is a backslash: \
```

- \' and \" - Single and Double Quotes:

Represent single and double quote characters within a string.

```
single_quote = 'This is a single quote: \''
double_quote = "This is a double quote: \""
print(single_quote)
print(double_quote)
```

```
This is a single quote: '
This is a double quote: "
```

Escape Characters

- \xhh - Hexadecimal Escape

Represents a character using its hexadecimal Unicode code point. Replace hh with the hexadecimal code.

- \ooo- Octal Escape

A backslash followed by three integers will result in a octal value

```
txt = "Octal escape: \110\145\154\154\157"
print(txt)
```

```
Octal escape: Hello
```

```
text = "Hex escape: \x48\x65\x6C\x6C\x6F"
print(text)
```

```
Hex escape: Hello
```

Raw Strings

- In Python, a raw string is a string literal that is prefixed with the letter 'r' (or 'R'), which is used to treat backslashes (\) as literal characters, rather than as escape characters.
- This means that in a raw string, backslashes are not used to escape special characters or sequences.

```
raw_string = r"This is a raw string\n"  # Will include '\n' as-is
print(raw_string)
```

```
This is a raw string\n
```

Raw Strings

- Raw strings are often used when dealing with **regular expressions, file paths,** or any situation where you want to treat backslashes as regular characters without escaping them.

```
#you need to escape each backslash with an additional backslash
#, which can make the path harder to read.
file_path = "C:\\Users\\username\\Documents\\file.txt"
print(file_path)
```

```
C:\Users\username\Documents\file.txt
```

```
#Using a raw string, you don't need to escape backslashes,
#making the file path more readable and less error-prone.
file_path = r"C:\Users\username\Documents\file.txt"
print(file_path)
```

```
C:\Users\username\Documents\file.txt
```

```
file_path = "C:\Users\username\Documents\file.txt"
print(file_path)
```

```
  File "<ipython-input-53-3dc517ba1909>", line 1
    file_path = "C:\Users\username\Documents\file.txt"
                ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape
```

String Methods

capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters

String Methods

isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations

String Methods

partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case

String Methods

translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Reference

https://www.w3schools.com/python/python_strings_methods.asp