String Operations

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Work with Strings
- Perform operations on String
- Manipulate Strings using indexing and escape sequences

The following example shows a string contained within 2 quotation marks:

```
# Use quotation marks for defining string

"Michael Jackson"
```

We can also use single quotation marks:

```
# Use single quotation marks for defining string

'Michael Jackson'
```

A string can be a combination of spaces and digits:

```
# Digitals and spaces in string

'1 2 3 4 5 6 '
```

A string can also be a combination of special characters :

```
# Special characters in string

'@#2_#]&*^%$'
```

We can print our string using the print statement:

```
# Print the string

print("hello!")
```

We can bind or assign a string to another variable:

```
# Assign string to variable
```

```python
name = "Michael Jackson"
name
```

It is helpful to think of a string as an ordered sequence. Each element in the sequence can be accessed using an index represented by the array of numbers:

The first index can be accessed as follows:

```python
# Print the first element in the string

print(name[0])
```

We can access index 6:

```python
# Print the element on index 6 in the string

print(name[6])
```

Moreover, we can access the 13th index:

```python
# Print the element on the 13th index in the string

print(name[13])
```

We can also use negative indexing with strings:

Negative index can help us to count the element from the end of the string.

The last element is given by the index -1:

```python
# Print the last element in the string

print(name[-1])
```

The first element can be obtained by index -15:

```python
# Print the first element in the string

print(name[-15])
```

We can find the number of characters in a string by using len, short for length:

```python
# Find the length of string

len("Michael Jackson")
```

We can obtain multiple characters from a string using slicing, we can obtain the 0 to 4th and 8th to the 12th element:

```
# Take the slice on variable name with only index 0 to index 3

name[0:4]

# Take the slice on variable name with only index 8 to index 11

name[8:12]
```

We can also input a stride value as follows, with the '2' indicating that we are selecting every second variable:

```
# Get every second element. The elments on index 1, 3, 5 ...

name[::2]
```

We can also incorporate slicing with the stride. In this case, we select the first five elements and then use the stride:

```
# Get every second element in the range from index 0 to index 4

name[0:5:2]
```

We can concatenate or combine strings by using the addition symbols, and the result is a new string that is a combination of both:

```
# Concatenate two strings

statement = name + "is the best"
statement
```

To replicate values of a string we simply multiply the string by the number of times we would like to replicate it. In this case, the number is three. The result is a new string, and this new string consists of three copies of the original string:

```
# Print the string for 3 times

3 * "Michael Jackson"
```

You can create a new string by setting it to the original variable. Concatenated with a new string, the result is a new string that changes from Michael Jackson to “Michael Jackson is the best".

```python
# Concatenate strings

name = "Michael Jackson"
name = name + " is the best"
name
```

Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, back slash "n" represents a new line. The output is given by a new line after the back slash "n" is encountered:

```python
# New line escape sequence

print(" Michael Jackson \n is the best" )
```

Similarly, back slash "t" represents a tab:

```python
# Tab escape sequence

print(" Michael Jackson \t is the best" )
```

If you want to place a back slash in your string, use a double back slash:

```python
# Include back slash in string

print(" Michael Jackson \\ is the best" )
```

We can also place an "r" before the string to display the backslash:

```python
# r will tell python that string will be display as raw string

print(r" Michael Jackson \ is the best" )
```

There are many string operation methods in Python that can be used to manipulate the data. We are going to use some basic string operations on the data.

Let's try with the method upper; this method converts lower case characters to upper case characters:

```python
# Convert all the characters in string to upper case

a = "Thriller is the sixth studio album"
print("before upper:", a)
b = a.upper()
print("After upper:", b)
```

Let's try with the method lower; this method converts upper case characters to lower case characters:

```python
# Convert all the characters in string to lower case
a = "MICHAEL JACKSON IS THE BEST"
print("Before lower:", a)
b = a.lower()
print("After lower:", b)
```

The method replace replaces a segment of the string, i.e. a substring with a new string. We input the part of the string we would like to change. The second argument is what we would like to exchange the segment with, and the result is a new string with the segment changed:

```python
a = "Michael Jackson is the best"
b = a.replace('Michael', 'Janet')
b

# Replace the old substring with the new target substring by removing
some punctuations.

a = "Hello! Michael Jackson has: 12 characters."
print(a)
b = a.replace('!','').replace(':','').replace('.','')
print(b)
```

The method find finds a sub-string. The argument is the substring you would like to find, and the output is the first index of the sequence. We can find the sub-string jack or el.

```python
# Find the substring in the string. Only the index of the first elment
of substring in string will be the output

name = "Michael Jackson"
name.find('el')

# Find the substring in the string.

name.find('Jack')
```

If the sub-string is not in the string then the output is a negative one. For example, the string 'Jasdfasdasdf' is not a substring:

```python
# If cannot find the substring in the string

name.find('Jasdfasdasdf')
```

The method Split splits the string at the specified separator, and returns a list:

```python
#Split the substring into list
name = "Michael Jackson"
```

```
split_string = (name.split())
split_string
```

RegEx

In Python, RegEx (short for Regular Expression) is a tool for matching and handling strings.

This RegEx module provides several functions for working with regular expressions, including search, split, findall, and sub.

Python provides a built-in module called re, which allows you to work with regular expressions. First, import the re module

```
import re
```

The search() function searches for specified patterns within a string. Here is an example that explains how to use the search() function to search for the word "Jackson" in the string "Michael Jackson is the best".

```
s1 = "Michael Jackson is the best"

# Define the pattern to search for
pattern = r"Jackson"

# Use the search() function to search for the pattern in the string
result = re.search(pattern, s1)

# Check if a match was found
if result:
    print("Match found!")
else:
    print("Match not found.")
```

Regular expressions (RegEx) are patterns used to match and manipulate strings of text. There are several special sequences in RegEx that can be used to match specific characters or patterns.

Special Sequence	Meaning	Example
\d	Matches any digit character (0-9)	"123" matches "\d\d\d"
\D	Matches any non-digit character	"hello" matches "\D\D\D\D\D"
\w	Matches any word character (a-z, A-Z, 0-9, and _)	"hello_world" matches "\w\w\w\w\w\w\w\w\w\w\w"
\W	Matches any non-word character	"@#$%" matches "\W\W\W\W"
\s	Matches any whitespace character (space, tab, newline, etc.)	"hello world" matches "\w\w\w\w\w\s\w\w\w\w\w"
\S	Matches any non-whitespace character	"hello_world" matches "\S\S\S\S\S\S\S\S\S\S"

Special Sequence	Meaning	Example
\b	Matches the boundary between a word character and a non-word character	"cat" matches "\bcat\b" in "The cat sat on the mat"
\B	Matches any position that is not a word boundary	"cat" matches "\Bcat\B" in "category" but not in "The cat sat on the mat"

Special Sequence Examples:

A simple example of using the \d special sequence in a regular expression pattern with Python code:

```
pattern = r"\d\d\d\d\d\d\d\d\d\d"  # Matches any ten consecutive digits
text = "My Phone number is 1234567890"
match = re.search(pattern, text)

if match:
    print("Phone number found:", match.group())
else:
    print("No match")
```

The regular expression pattern is defined as r"\d\d\d\d\d\d\d\d\d\d", which uses the \d special sequence to match any digit character (0-9), and the \d sequence is repeated ten times to match ten consecutive digits

A simple example of using the \W special sequence in a regular expression pattern with Python code:

```
pattern = r"\W"  # Matches any non-word character
text = "Hello, world!"
matches = re.findall(pattern, text)

print("Matches:", matches)
```

The regular expression pattern is defined as r"\W", which uses the \W special sequence to match any character that is not a word character (a-z, A-Z, 0-9, or _). The string we're searching for matches in is "Hello, world!".

The findall() function finds all occurrences of a specified pattern within a string.

```
s2 = "Michael Jackson was a singer and known as the 'King of Pop'"


# Use the findall() function to find all occurrences of the "as" in the string
result = re.findall("as", s2)
```

```
# Print out the list of matched words
print(result)
```

A regular expression's split() function splits a string into an array of substrings based on a specified pattern.

```
# Use the split function to split the string by the "\s"
split_array = re.split("\s", s2)

# The split_array contains all the substrings, split by whitespace
characters
print(split_array)
```

The sub function of a regular expression in Python is used to replace all occurrences of a pattern in a string with a specified replacement.

```
# Define the regular expression pattern to search for
pattern = r"King of Pop"

# Define the replacement string
replacement = "legend"

# Use the sub function to replace the pattern with the replacement
string
new_string = re.sub(pattern, replacement, s2, flags=re.IGNORECASE)

# The new_string contains the original string with the pattern
replaced by the replacement string
print(new_string)
```

What is the value of the variable a after the following code is executed?

```
# Write your code below and press Shift+Enter to execute

a = "1"
```

What is the value of the variable b after the following code is executed?

```
# Write your code below and press Shift+Enter to execute

b = "2"
```

What is the value of the variable c after the following code is executed?

```
# Write your code below and press Shift+Enter to execute
```

```
c = a + b
#Write 12
```

Consider the variable d use slicing to print out the first three elements:

```
# Write your code below and press Shift+Enter to execute

d = "ABCDEFG"

print(d[:3]) # or print(d[0:3])
```

Use a stride value of 2 to print out every second character of the string e:

```
# Write your code below and press Shift+Enter to execute

e = 'clocrkr1e1c1t'
print(e[::2])
```

Print out a backslash:

```
# Write your code below and press Shift+Enter to execute
print("\\\\\\\\") #or print(r"\ ")
```

Convert the variable f to uppercase:

```
# Write your code below and press Shift+Enter to execute

f = "You are wrong"
f.upper()
```

Convert the variable f2 to lowercase:

```
# Write your code below and press Shift+Enter to execute
f2="YOU ARE RIGHT"
f2.lower()
```

Consider the variable g, and find the first index of the sub-string snow:

```
# Write your code below and press Shift+Enter to execute

g = "Mary had a little lamb Little lamb, little lamb Mary had a little
lamb \
Its fleece was white as snow And everywhere that Mary went Mary went,
Mary went \
Everywhere that Mary went The lamb was sure to go"

g.find("snow")
```

In the variable g, replace the sub-string Mary with Bob:

```
# Write your code below and press Shift+Enter to execute
g.replace("Mary", "Bob")
```

In the variable g, replace the sub-string , with .:

```
# Write your code below and press Shift+Enter to execute
g.replace(',','.')
```

In the variable g, split the substring to list:

```
# Write your code below and press Shift+Enter to execute
g.split()
```

In the string s3, find the four consicutive digit character using \d and search() function:

```
# Write your code below and press Shift+Enter to execute
s3 = "House number- 1105"

# Use the search() function to search for the "\d" in the string
result = re.search("\d", s3)

# Check if a match was found
if result:
    print("Digit found")
else:
    print("Digit not found.")
```

In the string str1, replace the sub-string fox with bear using sub() function:

```
# Write your code below and press Shift+Enter to execute

str1= "The quick brown fox jumps over the lazy dog."

```

In the string str2 find all the occurrences of woo using findall() function:

```
str2= "How much wood would a woodchuck chuck, if a woodchuck could
chuck wood?"

# Write your code below and press Shift+Enter to execute
```

Congratulations, you have completed your first lesson and hands-on lab in Python.

Author

Joseph Santarcangelo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-12-28	2.2	Pratiksha	Updated exercises
2022-01-10	2.1	Malika	Removed the readme for GitShare
2020-11-11	2.1	Aije	Updated variable names to lowercase
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab