

String Formatting

- A better way of formatting strings is using the **format** method

Pattern

`str = "{}. {} ({})"`



Format

`str.format(1, "Player", "Team")`



Result

`"1. Player (Team)"`

String Formatting

- The **format** method also allows indexing the values.

Pattern

`str = "{2}. {1} ({0})"`



Format

`str.format("Team", "Player", 1)`



Result

`"1. Player (Team)"`

String Formatting

- The **format** method also allows using keyword arguments.

Pattern

`str = "{id}. {name} ({t})"`

Format

`str.format(
t="Team", name="Player", id=1)`

Result

`"1. Player (Team)"`

- There is a third way of formatting strings: **f-strings**.

Values



```
name = "Player"; id = 1
```



F-string



```
str = f"{id}. {name}"
```



Result



```
"1. Player"
```

- F-strings are more flexible than the alternatives. They can be used to execute functions.

Values

`name = "PLAYER"; id = 1`

F-string

`str = f"{id}. {name.lower()}"`

Result

`"1. player"`

At the core of the lesson

Python String methods:

- Using the variety of Python methods and concatenation/slicing operators we can manipulate Strings.
 - With the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.
- We used a variety of String methods for string manipulation.

Self Study



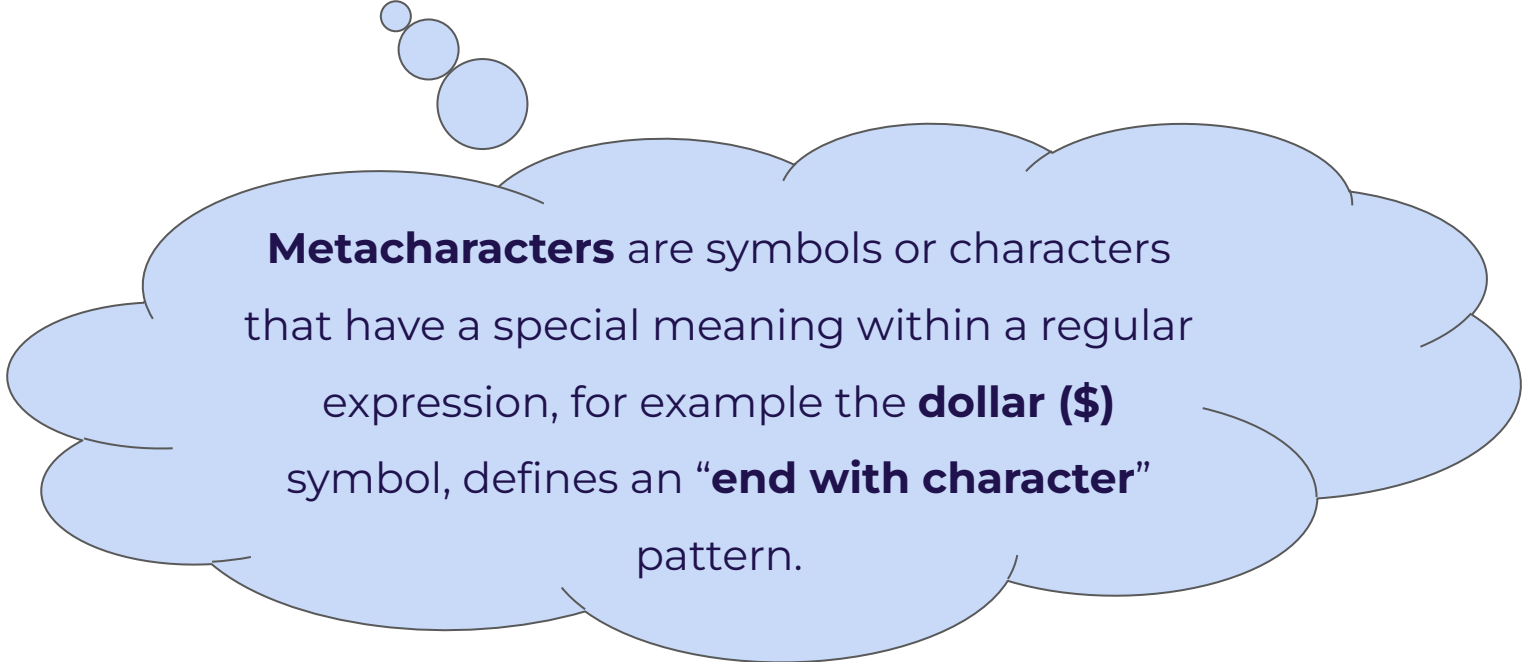
String Methods:

- multiline
- escaping
- Template

Regular Expressions 1/2

- A regular expression is a **sequence of characters** that forms a search **pattern**. When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single **character**, or a more complicated **pattern**.
- Regular expressions can be used to perform all types of text search and text **replace** operations.
- Python does not have a built-in Regular Expression **library**, but we can import the **re** package to work with regular expressions.

In Python the **re** library allows you to define **regular expression functions** and **regular expression metacharacters**



Metacharacters are symbols or characters that have a special meaning within a regular expression, for example the **dollar (\$)** symbol, defines an “**end with character**” pattern.

Character Classes	Description
<code>[]</code>	A set of characters, e.g. <code>[a,b]</code> or <code>[a-e]</code>
<code>.</code>	Any character, except the newline e.g. <code>"He."</code> (it could be <i>Hello</i> or <i>Help</i>)
<code>^</code>	Starts with e.g. <code>"^Hi"</code> .
<code>\$</code>	Ends with e.g. <code>"world\$"</code> .
<code>*</code>	Zero or more occurrences e.g. <code>"Hello*"</code>
<code> </code>	Either one or the other e.g. <code>"Hello Hi"</code>

Special Sequences I

Character Classes	Description
<code>\A</code>	Returns a match if the specified characters are at the beginning of the string
<code>\b</code>	Returns a match where the specified characters are at the beginning or at the end of a word
<code>\d</code>	Returns a match where the string contains digits (numbers from 0-9)
<code>\D</code>	Returns a match where the string does not contain digits

Special Sequences II

Character Classes	Description
<code>\s</code>	Returns a match where the string contains a white space character
<code>\S</code>	Returns a match where the string does not contain a white space character
<code>\w</code>	Returns a match where the string contains any word characters
<code>\W</code>	Returns a match where the string does not contain any word characters

Set	Description
<code>[xyz]</code> <code>[0123]</code>	Returns a match where one of the specified characters (x, y, z) are present . Same application for numbers
<code>[a-e]</code> <code>[0-5]</code>	Returns a match for any lower case character, alphabetically between a and e . Same application for numbers
<code>[^xyz]</code>	Returns a match for any character except x, y, z

Set	Description
<code>[a-eA-E]</code>	Returns a match for any character alphabetically between a and e, lower case OR upper case
<code>[0-5][0-9]</code>	Returns a match for any two-digit numbers from 00 and 59
<code>[+]</code>	In sets, symbols such as: +, *, ., , (), \$, {} has no special meaning, so <code>[+]</code> means: return a match for any + character in the string

At the core of the lesson

Lessons Learned:

- We know what is the regular expression package **re** provided by Python.
- We have described metacharacter classes, special sequences and use of sets for regular expression matching in Python.

Regular Expressions 2/2

- The **re** package provides a variety of methods to use in order to extract data based on a preconfigured pattern.
- The **re** package provides the following classes for regular expressions.

Python Regular Expression Library

The `import` statement will provide access to the library

```
import re
```

Regex (`re`) library provides access to a variety of tools

Pattern: It is the compiled version of a regular expression. It is used to define a **pattern** for the **regex library**.

Method	Description
<code>re.findall(<i>tofind</i>, <i>string</i>)</code>	The method will return a list of data for all <i>tofind</i> matches in the <i>string</i> variable. Matches are stored in the list in the order that that have been found.
<code>re.search(<i>pattern</i>, <i>string</i>)</code>	The method searches for a match and return a match object. The <i>pattern</i> it could be the metacharacter (\s space), and the <i>string</i> is the text to look for.

Method	Description
<code>re.split(pattern, string)</code>	The method will return a list of data by splitting <i>pattern</i> , e.g. we can use <code>\s</code> for splitting by space or create custom patterns (as we will explore in the next slides).
<code>re.sub(pattern, replacewith, string)</code>	The method replaces every <i>pattern</i> character (<code>\s</code> for space) with a <i>replacewith</i> character(s), for example <code>%20</code> that is used for URL encoding.

Method	Description
<code>re.start()</code>	The method returns the index of the start of the matched substring
<code>re.end()</code>	The method returns the index of the end of the matched substring

Creating Custom Regex Patterns I

Pattern I: We can create custom patterns to extract data using the regex library, in this example, we will explore numbering patterns.

Example I	Pattern I
<code>string = '39801 356, 2102 1111'</code>	<code>pattern = '(\d{3}) (\d{2})'</code> <ul style="list-style-type: none">The <code>pattern</code> is a three digit number followed by space followed by two digit number

Hint: Custom patterns can be used in the regex methods such as in `search()`

Creating Custom Regex Patterns II

Pattern II: We can create custom patterns to extract data using the regex library, in this example, we will explore text patterns.

Example II	Pattern II
<pre><i>string = 'Berlin is a beautiful city'</i></pre>	<pre><i>pattern = '^Berlin.*city\$'</i></pre> <ul style="list-style-type: none">• The pattern defines that the string starts with Berlin and ends with city

Hint: Custom patterns can be used in the regex methods such as in **search()**

At the core of the lesson

Lessons Learned:

- At this lesson we have explained the use of text (Strings) in Python.
- We learned how to create and manipulate Strings using a variety of methods.
- We have also explained how to work with Strings in terms of extracting subcontinent and to concatenate them.
- We have Also learned what are the methods of the **re** (regular expression library) in Python using a variety of concepts including use of metacharacters, special characters, sets and methods for data extraction.

Documentation

1. [Python.org documentation](#)
2. [W3Schools](#)
3. [Digital Ocean](#)
4. [Tutorialspoint](#)

THANK YOU

A large group of diverse young adults, likely students or employees, are posing for a group photo in a room with a projector screen in the background. The group is arranged in several rows, with some people sitting on the floor in the front. Many are making playful gestures like peace signs or giving thumbs up. The text "THANK YOU" is overlaid in large white letters across the center of the image.