Python Course - Texts in Python

Digital Career Institute

Digital Career Institute

Goal of the Submodule

The goal of this submodule is to help the learners work with texts in Python. By the end of this submodule, the learners should be able to understand:

- Different types of texts used in Python.
- The easy to handle and work with text including text concatenation and modification.
- How to use the String method.
- Learning the concepts of regular expressions and patterns in Python.

Digital Career Institute

Topics

- **Introduction to Texts in Python**
- **Difference between characters and strings**
- **Working with Strings:**
  - Length and concat methods
  - Concat
- **Modifying Strings:**
  - Using a variety of methods including the strip, upper, lower, split and other methods
- **Using the String methods:**
  - Using a variety of methods including find, replace, split and other methods
- **Using regular Expressions with Python**
  - Introduction to basic Python Regular Expressions to search and replace text

Digital Career Institute

Glossary

Term	Definition
Char	A char is a single character, that is a letter, a digit, a punctuation mark, a tab, a space or something similar.
String	String is a sequence of characters, for e.g. “Hello” is a string of 5 characters.
Built-in method	A ready made functions to be used in Python

Introduction to Python Strings

Digital Career Institute

Introduction to Strings

A **String** is a Python data type to store textual data. **String** characteristics include:

- It is a collection of characters stored in an array format
- We use double quotes to create a **String**
  - If we want to span a String in multiple lines we can use the triple quote, this is ideal for long text.
  - Special characters like Tabs or Newlines can also be used within the triple quotes.
- We can use the backslash character (\) to escape quotes in **Strings**
- **Strings** are arrays of characters, thus we can slice them using the brackets and the character index locations

Difference between characters and Strings

Digital Career Institute

Difference Between Characters and Strings

Character	String
A **character** is a single letter, number, punctuation mark or symbol.	A **string** is a one-dimensional array of characters terminated by a null character.
Character is an element.	A string is a set of characters.
Single or double quotes are used to represent a character.	Single or double quotes are used to represent a string.
Character refers to a single letter, number, space.	String refers to a set of characters.

How to create a String?

String characteristics

A string is a series of characters.
To create a String we will need to use single or double quotes

Strings are immutable data, this means that once created we cannot change it,
but we can reinitialize it!

When a string reference is reinitialized with a new value, it is creating a new object rather than overwriting the previous value.

Python String Methods

Digital Career Institute

Method	Description
`string.capitalize()`	This method is used to `capitalize` a text (convert the first character to uppercase)
`string.upper()`	This method is used to transform a text into an `upper` (uppercase) text
`string.lower()`	This method is used to transform a text into a `lower` (lowercase) text

⇰ The methods do not accept any arguments, but they can be used using the dot operator

Python String Methods

Method	Description
`string.isalpha()`	This method is used to check if all the characters in the text are letters
`string.isdecimal()`	This method is used to check if all the characters in the text are decimals
`string.isnumeric()`	This method is used to check if all the characters in the text are numbers

⇰ The methods do not accept any arguments, but they can be used using the dot operator

Python String Methods

Digital Career Institute

Method	Description
`string.find(value,start,end)`	This method is used to find a text inside the String. This is ideal when we have a phrase. The ***value*** is required, while the ***start***/***end*** are optional and denote the index of the String positions.
`string.index(value,start,end)`	This method is used to finds the first occurrence of the specified value and return its index. This works similar to find, but it returns an exception if the value is not found, so we need to handle it with a catch statement.

Python String Methods

Method	Description
`string.split(separator, maxsplit)`	This method is used to split a string in a list of words. The `separator` defines the base String to split with (e.g. dash), the whitespace is default. The `maxsplit` refers to how many splits to do and it is optional.
`string.replace(oldvalue, newvalue, count)`	This method is used to replace a given String with a another String. The `oldvalue` and `newvalue` are required. The `count` is optional and specifies how many occurrences of the given value we want to replace.

Python String Creation

- Python Strings can be created (a process that is also known as casting) from a different data type, including integers, floats and booleans. For this reason, we can use the in-built `str()` method.

Method	Description
`str(string, encoding='utf-8', errors='strict')`	This method contracts a string version of the given object. ● The `string` is the object to be returned as String, typically a number e.g. for concatenation. ● The `encoding` refers to the encoding of the object. Defaults of UTF-8 when not provided. ● The `errors` is the response when decoding fails. Defaults is 'strict'.

At the core of the lesson

Lessons Learned:

- We know what is String and why it is immutable
- We know how to create a String and what is the difference between Strings and Characters in Python
- We know how to use different String manipulation methods including:
  - capitalize()
  - upper()
  - lower()
  - isalpha()
  - isdecimal()
  - isnumeric()
  - find()
  - index()
  - split()
  - replace()

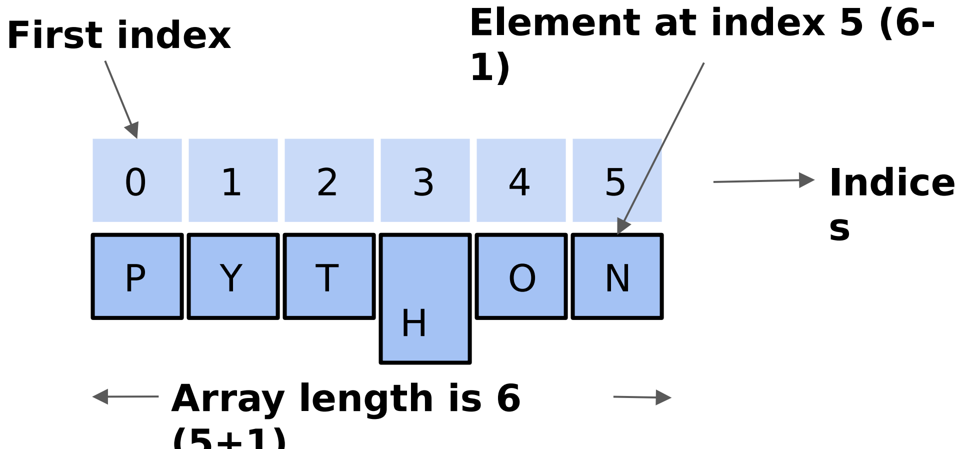
Digital Career Institute

Working with Strings

Digital Career Institute

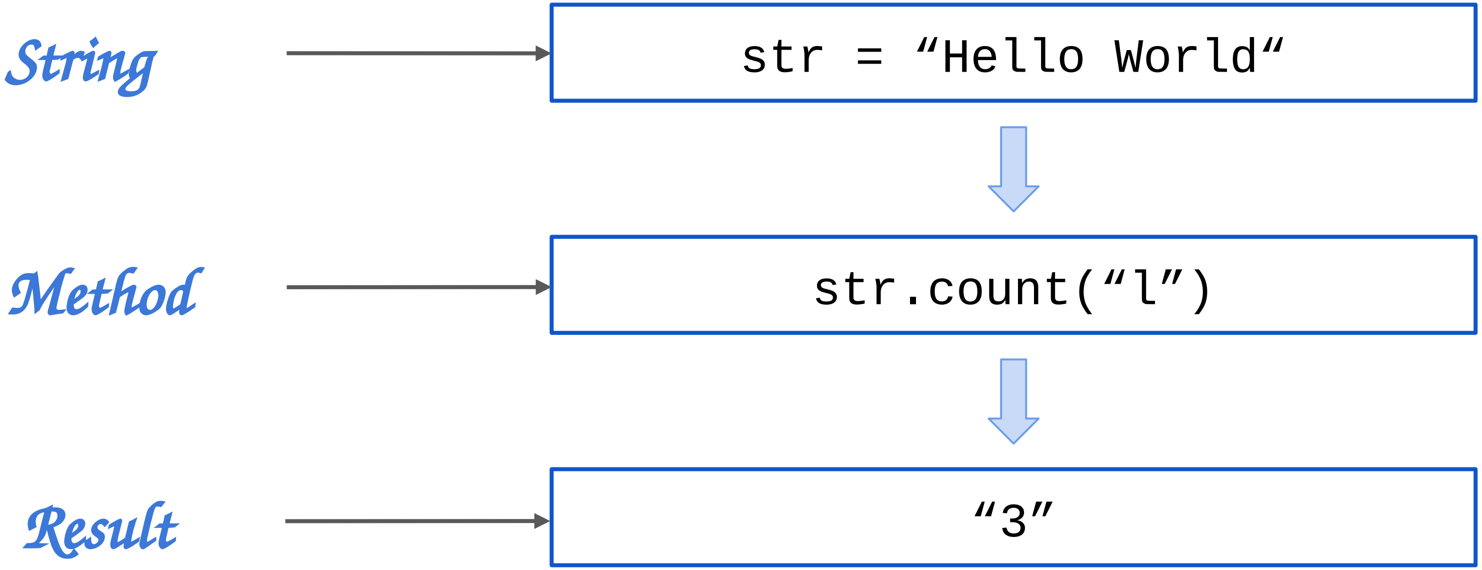
Strings are Arrays of Characters

- **len()**: is an in-built function of Python that returns the **length** of the String, or the size of the array of characters
  - An array in Python starts always from index 0
  - The length of the array equals to the last index value plus one
  - We can also say that the last index of the array equals to the length of the array minus one



Counting characters

- **count()**: is an in-built function of Python that returns the **number of time** a value appears in the String



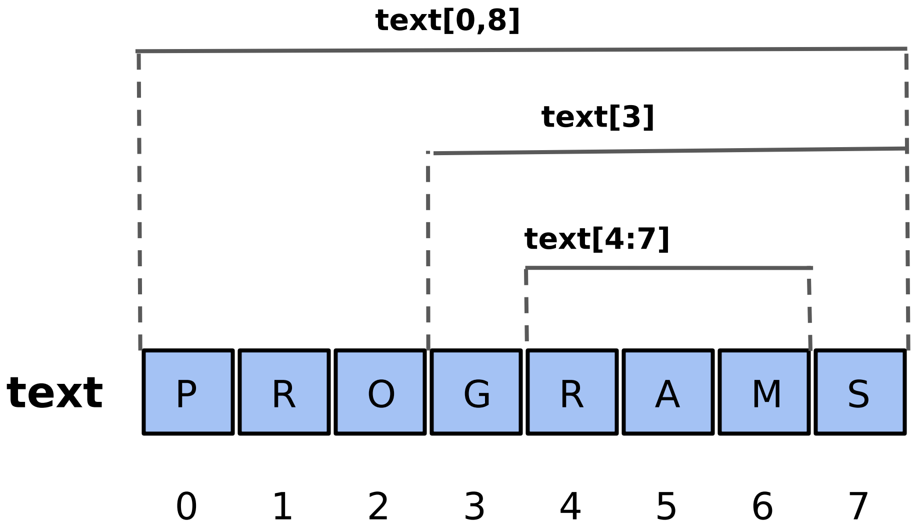
```
str = “Hello World“
```

```
str.count(“l”)
```

```
“3”
```

Extracting Substrings: Slicing Strings

- **Substring Method:** A part of string is called substring. In other words, substring is a subset of another string. In case of substring **startIndex** is inclusive and **endIndex** is exclusive.



Strings Slicing options

- **You can slice Strings with different options**

1. `string[x:y]`

   Extract a slice of characters starting from the `x` index value until the `y` index value.

2. `string[x:]`

   Extract the last slice of characters starting from the `x` index value.

3. `string[:y]`

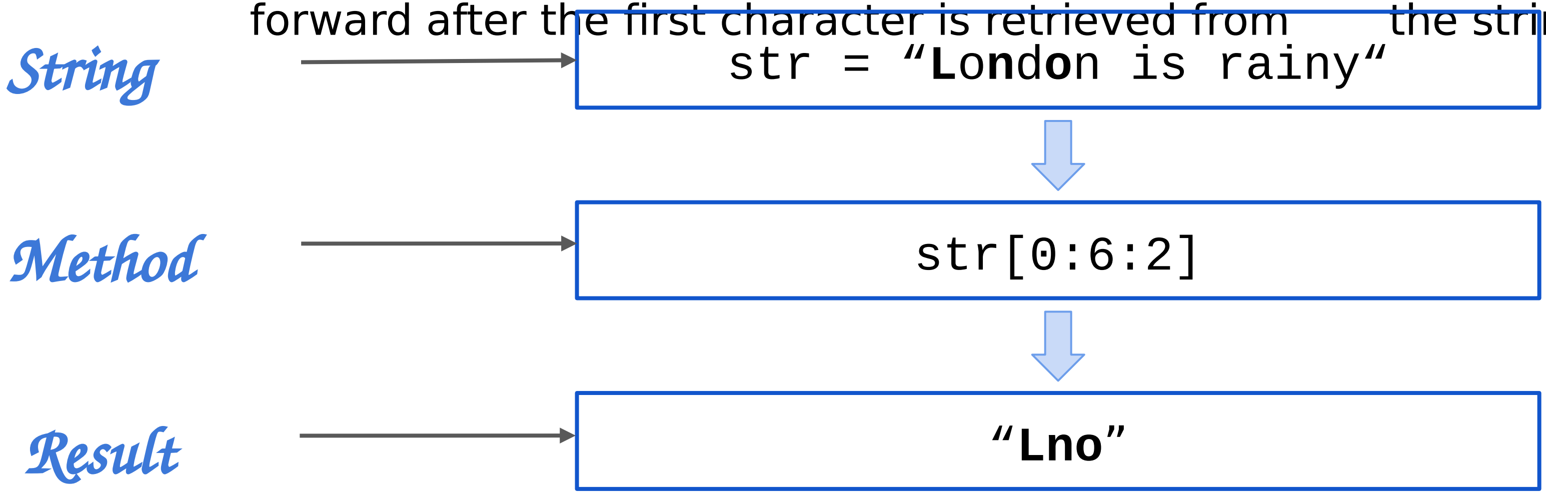
   Extract the first slice of characters until the `y` index value.

Specifying Stride while Slicing Strings

- **You can specify strides with different options**

1. **`String[x:y:stride]`**

The new parameter **`stride`** refers to how many characters to move forward after the first character is retrieved from the string.



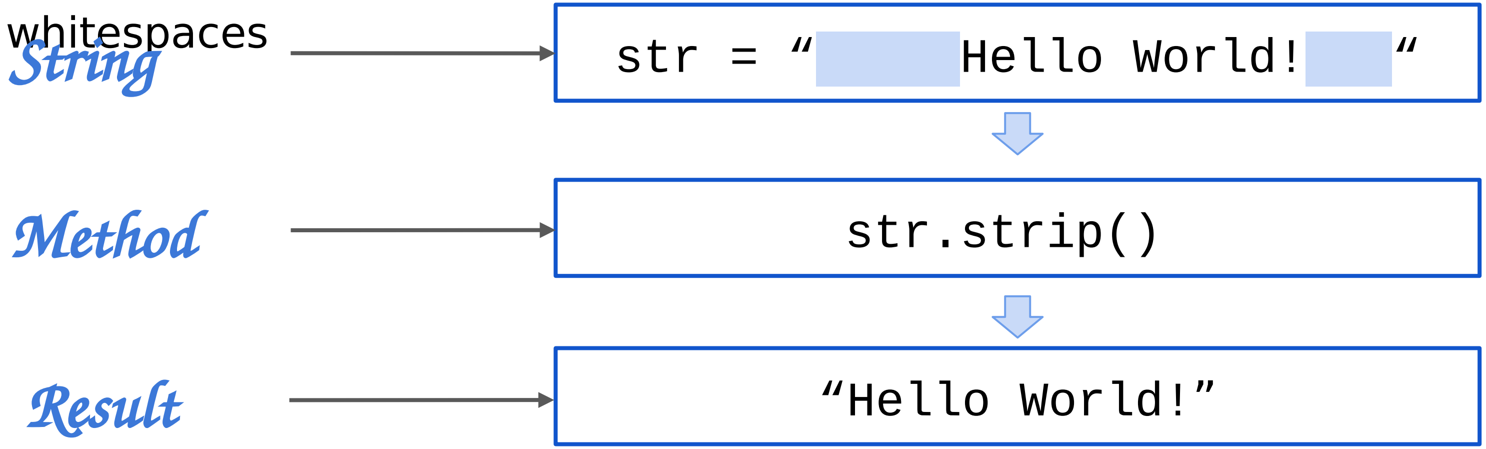
```
str = “London is rainy“
```

```
str[0:6:2]
```

```
“Lno”
```

Python Trim (strip) Methods

- **Strip Methods:** Built-in function to remove leading and trailing whitespaces



```
str = “      Hello World!     “
```

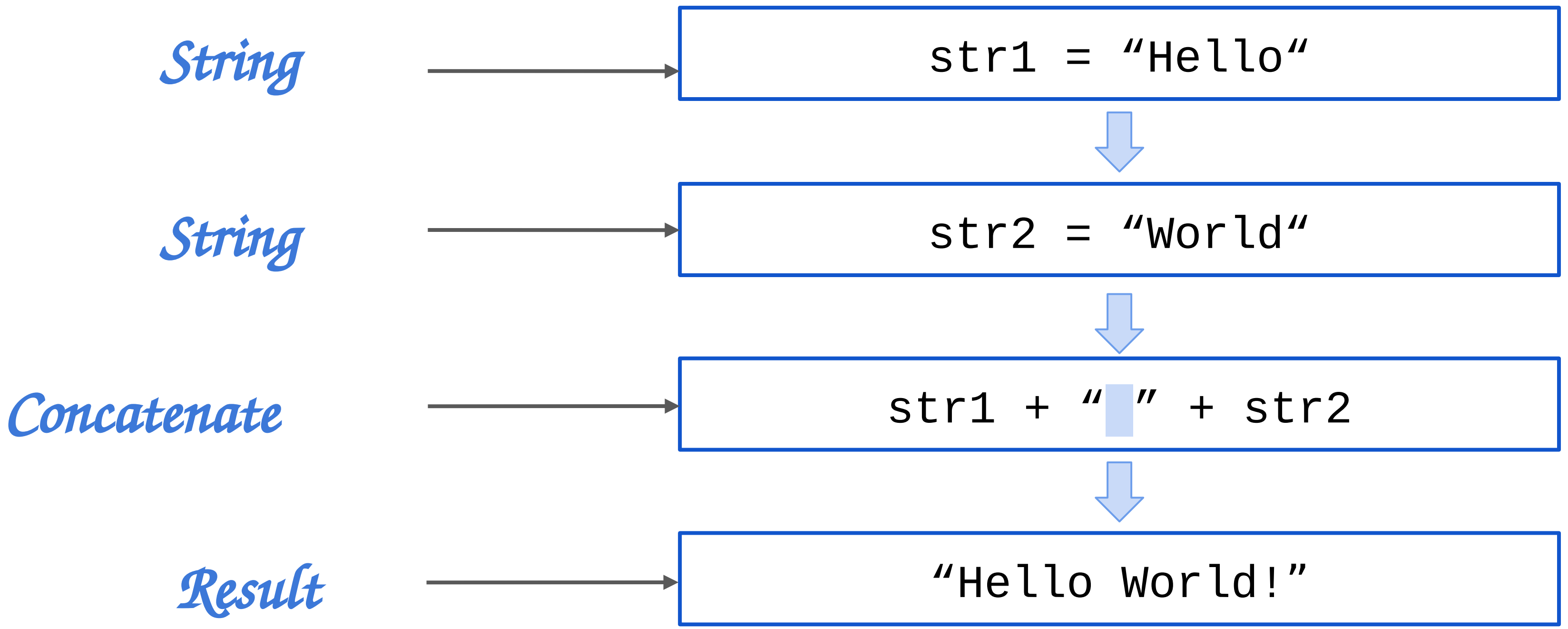
```
str.strip()
```

```
“Hello World!”
```

- `rstrip` removes leading and trailing whitespaces from “**r**ight” side of string
- `lstrip` removes leading and trailing whitespaces from “**l**eft” side of string

Strings Concatenation

- **Using the plus operator (+):** Add a variable to another variable



```
str1 = “Hello“
```

```
str2 = “World“
```

```
str1 + “ ” + str2
```

```
“Hello World!”
```

Strings Concatenation

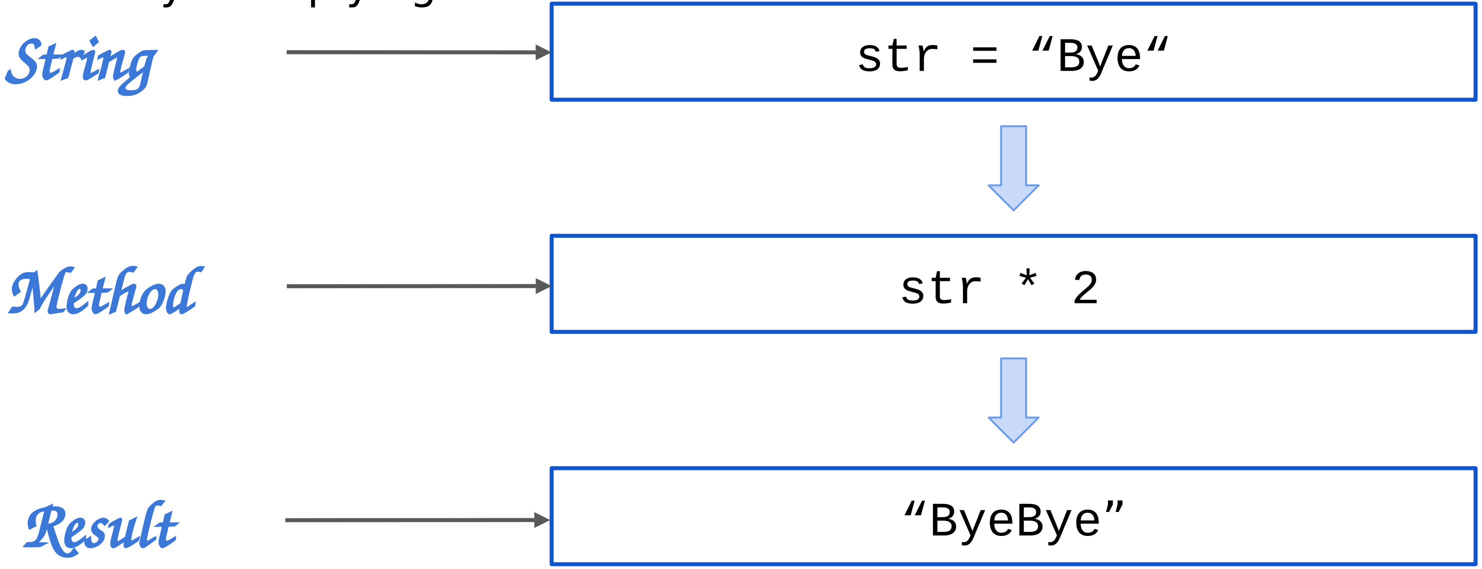
- **For Strings:**
  The `plus` (+) works as a String concatenation operator.

- **For Numbers:**
  The `plus` (+) works as a mathematical operator.

Tip: Use the plus operator carefully! Do not try to concatenate a String with a number, in this case Python will return an error.

Strings Concatenation

- **Using the multiplication operator (*):** Repeat a variable multiple times by multiplying with a number.



```
str = “Bye“
```

```
str * 2
```

```
“ByeBye”
```

String Concatenation: Formatting

- Another way of concatenating strings is using string formatting.
- String formatting can be done using the **%** operator.

Pattern

```
str = "%d. %s (%s)"
```

Format

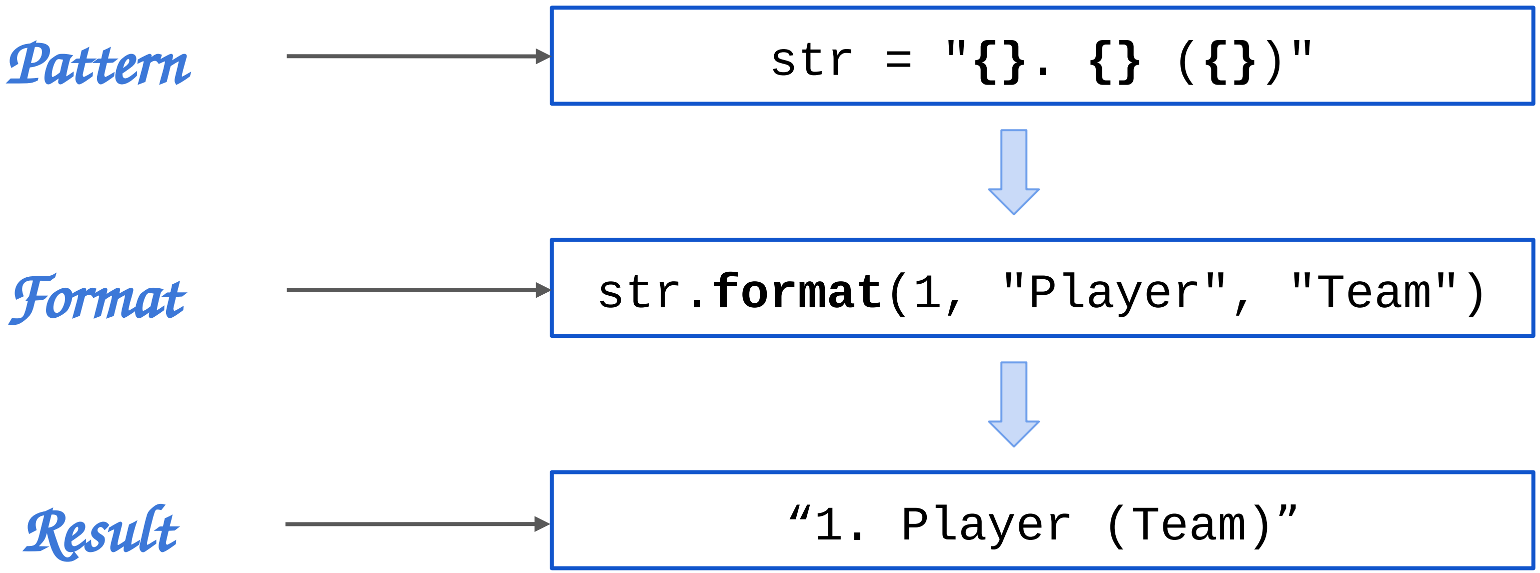
```
str % (1, "Player", "Team")
```

Result

```
“1. Player (Team)”
```

String Formatting

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



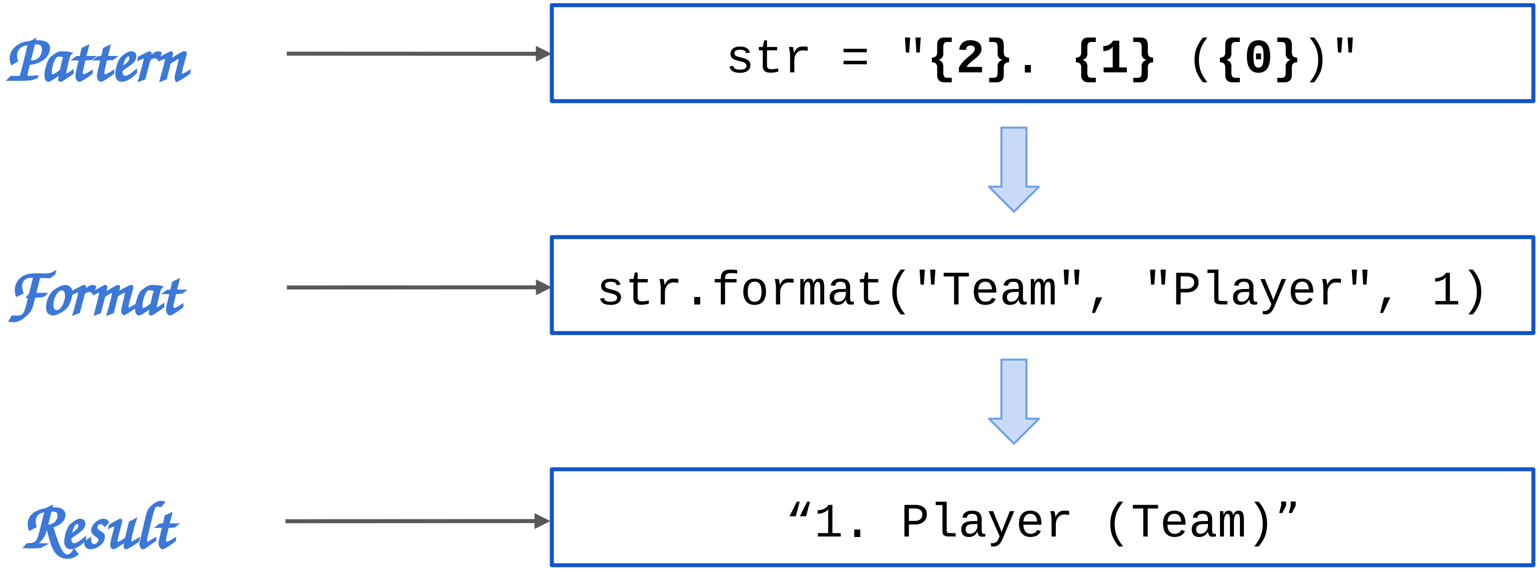
```
str = "{}. {} ({})"
```

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

```
“1. Player (Team)”
```

String Formatting

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



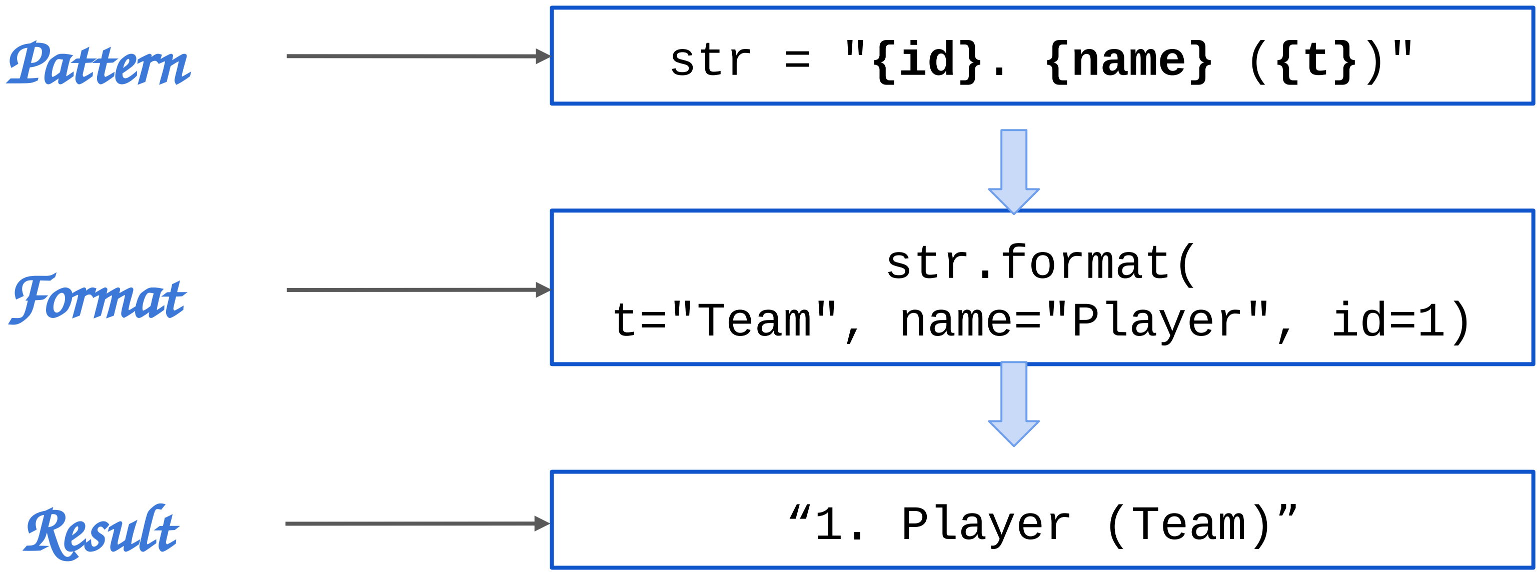
```
str = "{2}. {1} ({0})"
```

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

```
“1. Player (Team)”
```

String Formatting

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



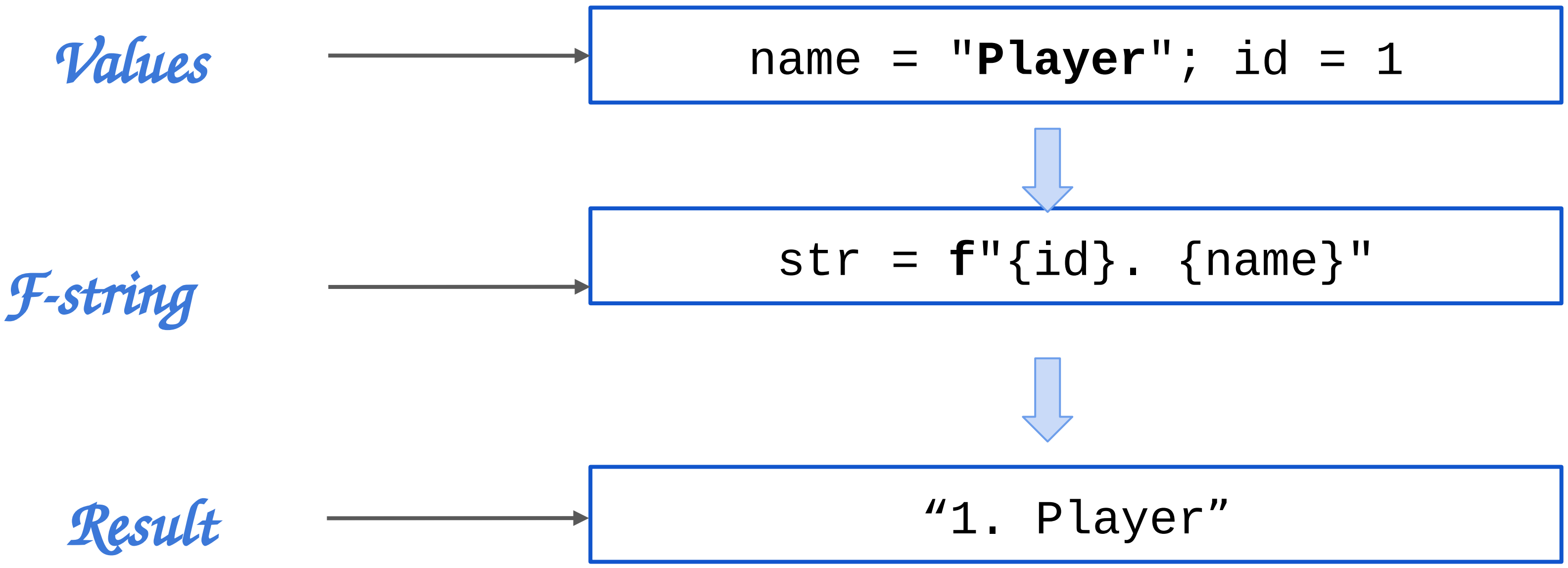
```
str = "{id}. {name} ({t})"
```

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

```
“1. Player (Team)”
```

String Formatting

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



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

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

```
“1. Player”
```

String Formatting

- 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.

Digital Career Institute

Self Study

String Methods:

- multiline
- escaping
- Template

Regular Expressions 1/2

Digital Career Institute

Regular Expressions Constructs

- 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.

Character

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.

Metacharacters

Digital Career Institute

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

Special Sequences I

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

Special Sequences II

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

Sets I

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

Sets II

Digital Career Institute

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

Digital Career Institute

Regular Expressions 2/2

Digital Career Institute

Regex methods

- 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

Regex Methods

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

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

Regex Methods

Digital Career Institute

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

Regex Methods

Method	Description
`re.start()`	The method returns the index of the `start` of the matched substring
`re.end()`	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
`string = '39801 356, 2102 1111'`	`pattern = '(\d{3}) (\d{2})'` ● The `pattern` 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

Digital Career Institute

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
`string = 'Berlin is a beautiful city'`	***`pattern = '^Berlin.*city$'`*** ● 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.

Digital Career Institute

Documentation

Digital Career Institute

Documentation

1. **Python.org documentation**
2. **W3Schools**
3. **Digital Ocean**
4. **Tutorialspoint**

THANK YOU

Digital Career Institute