Working with Strings

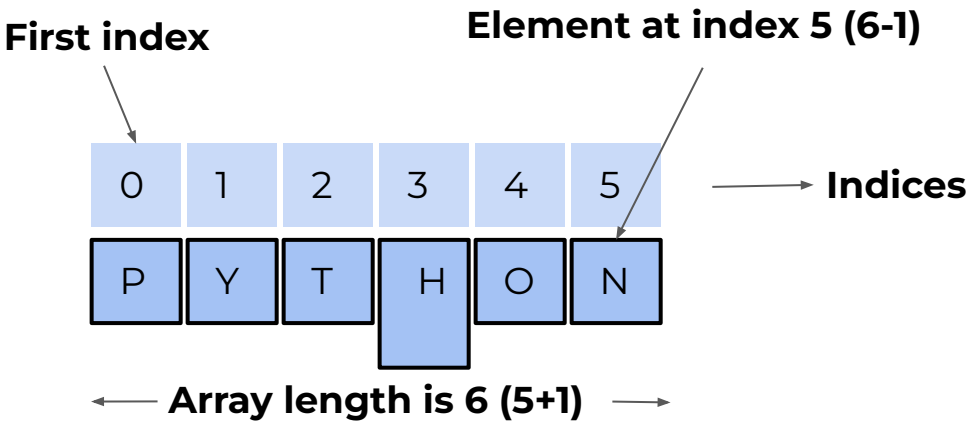
Python String Creation

- 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')	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'.

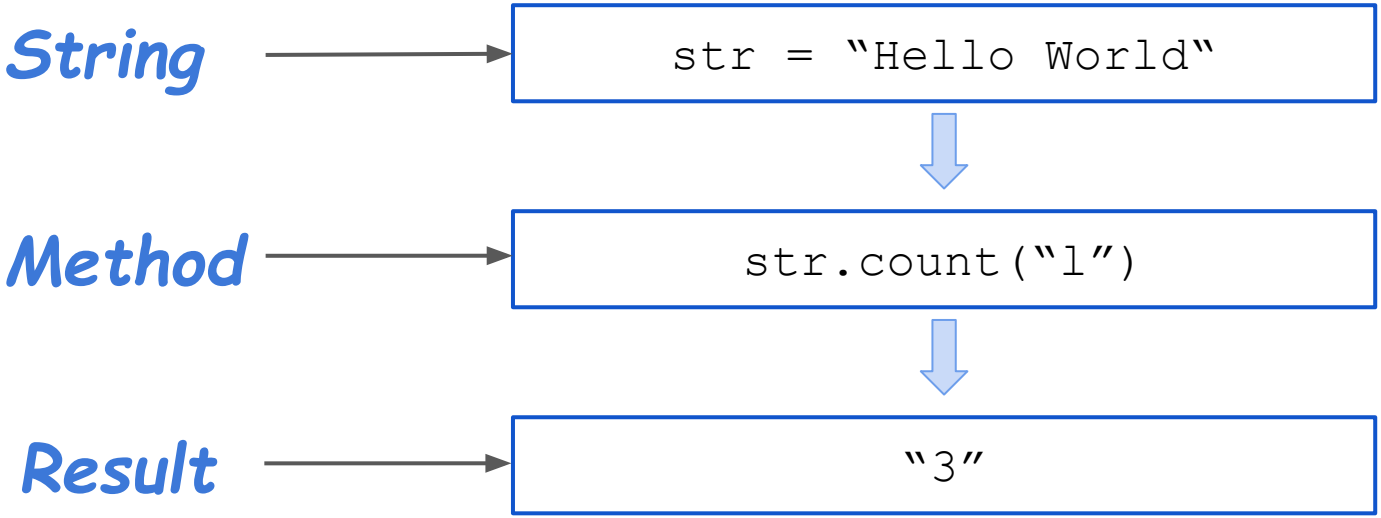
Strings are Arrays of Characters

- **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



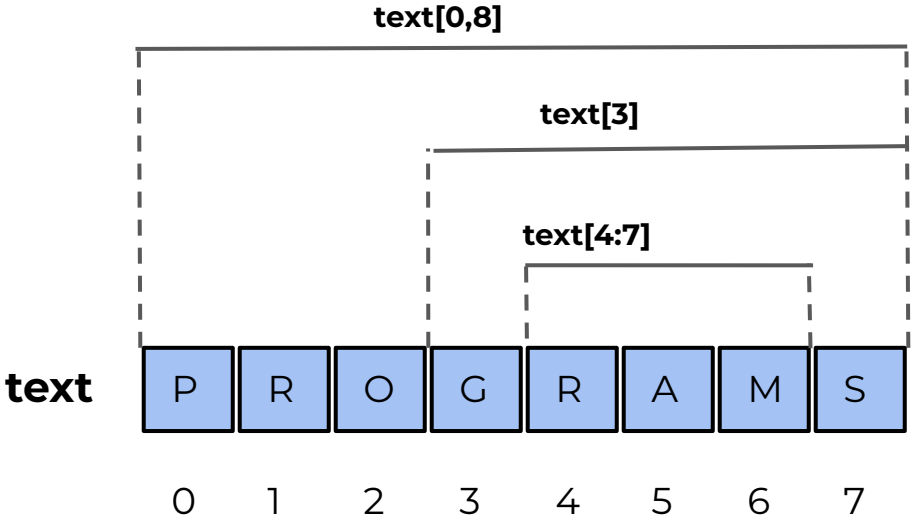
Counting characters

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



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]`

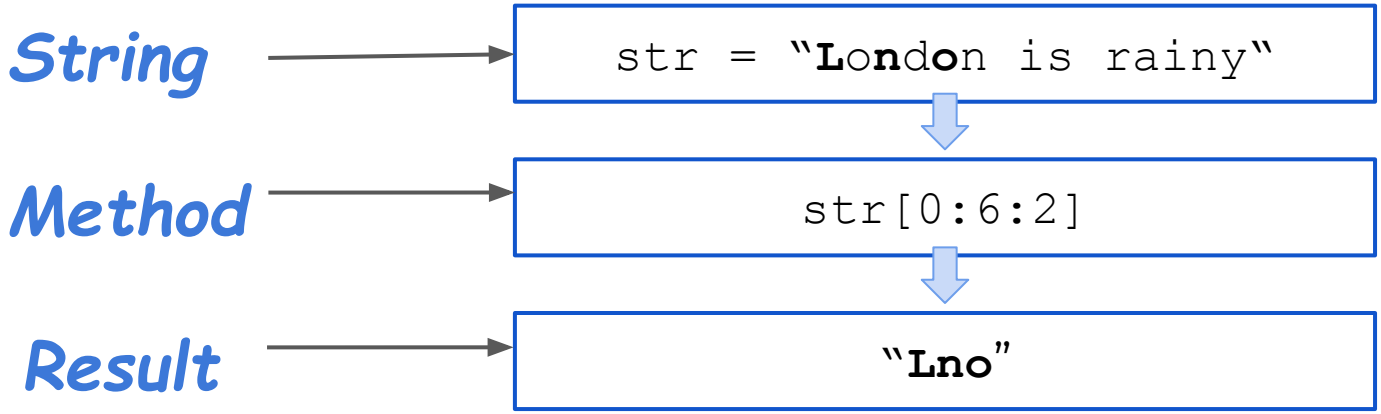
   Extract the first slice of characters until the `y` index value.

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.



Python Trim (strip) Methods

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

String → `str = "  Hello World!  "`

↓

Method → `str.strip()`

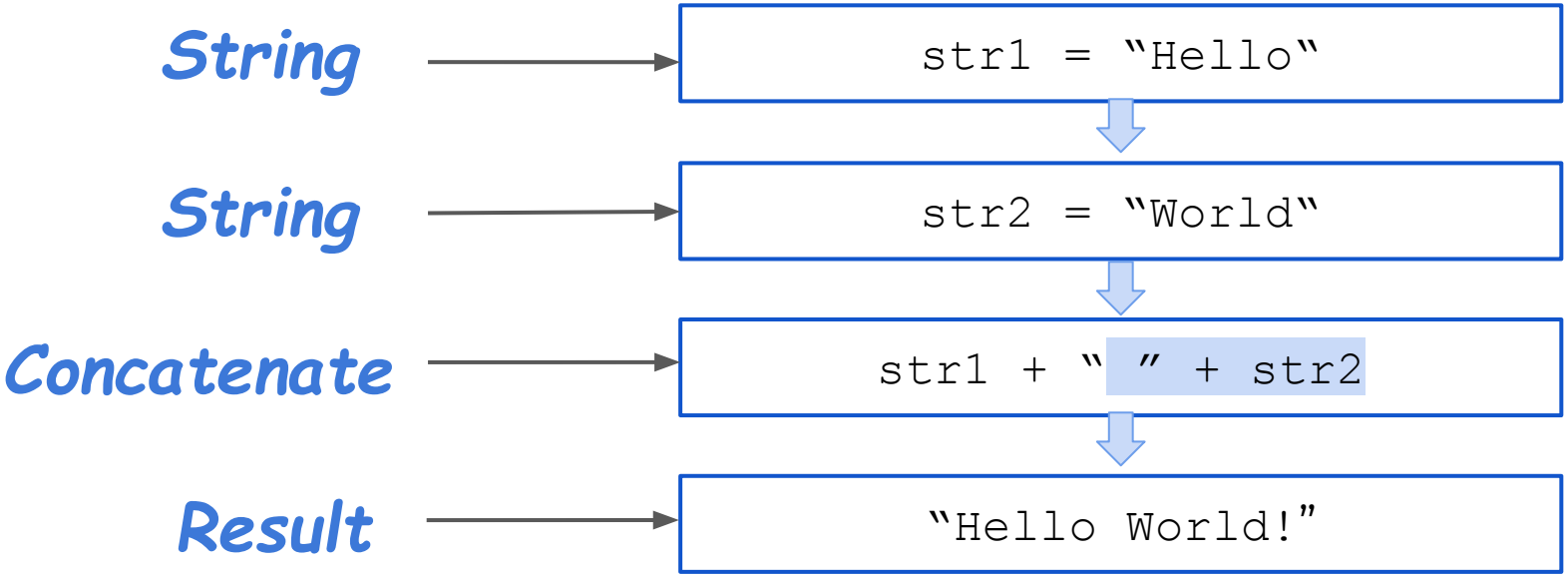
↓

Result → `"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



Strings Concatenation

- **For Strings:**

The `plus` (+) works as a String concatenation operator.

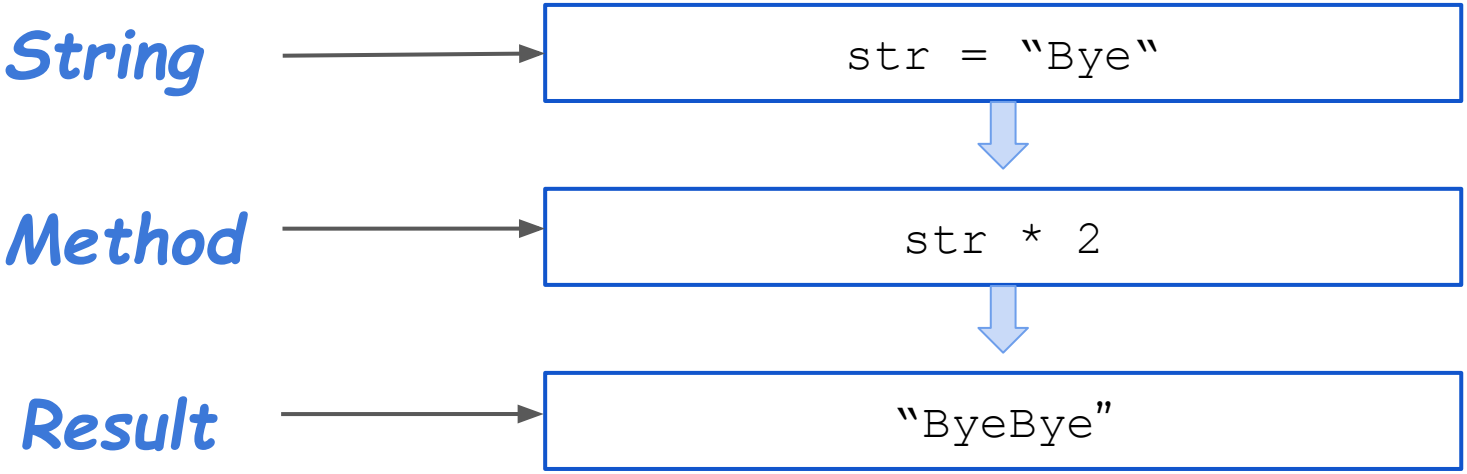
- **For Numbers:**

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.



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

Pattern →

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

↓

Format →

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

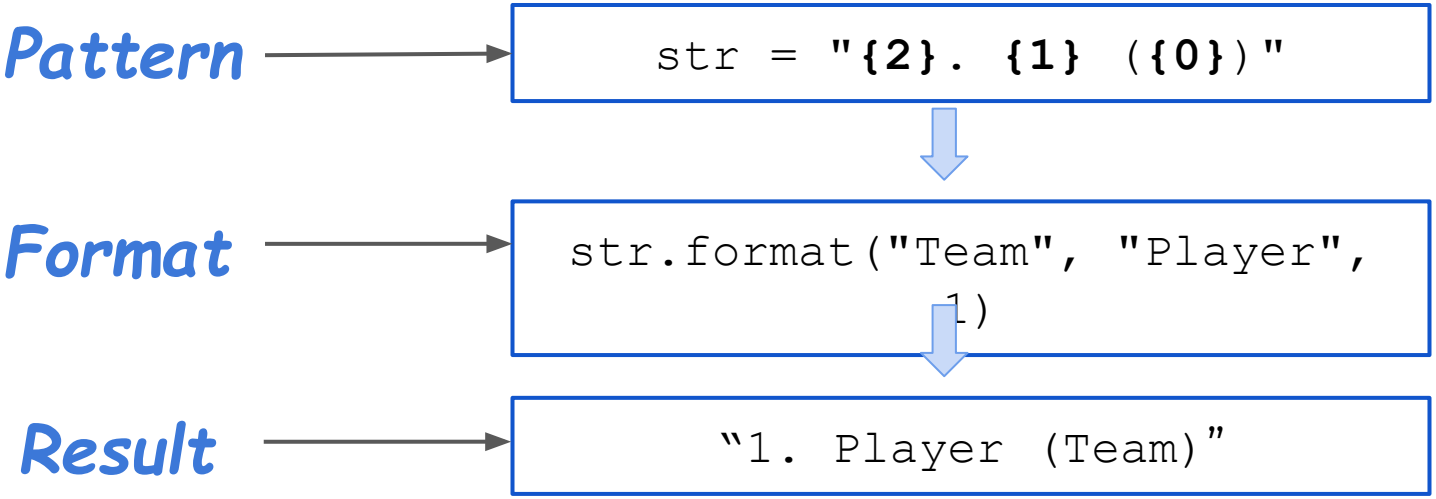
↓

Result →

```
"1. Player (Team)"
```

String Formatting

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



String Formatting

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

Pattern

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

Format

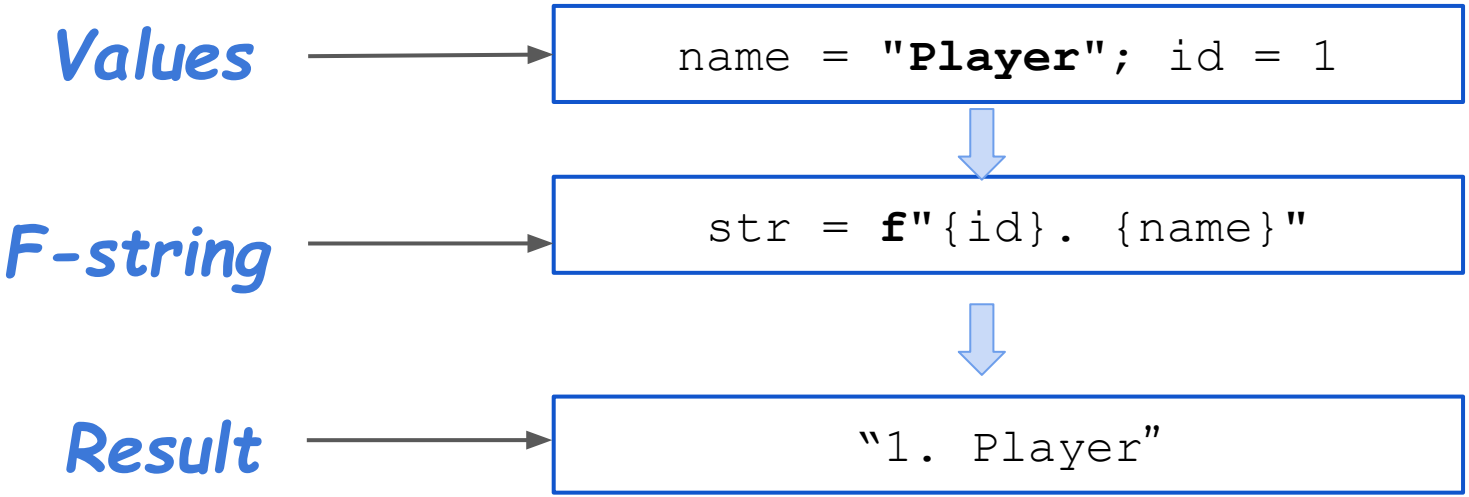
```
str.format(
t="Team", name="Player", id=1)
```

Result

```
"1. Player (Team)"
```

String Formatting

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



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.

Self Study

String Methods:

- multiline
- escaping
- Template