Introduction to string manipulation

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

You will learn

- **String manipulation**
  - e.g. replace and find specific substrings
- **String formatting**
  - e.g. interpolating a string in a template
- **Basic and advanced regular expressions**
  - e.g. finding complex patterns in a string

Why it is important

- Clean dataset to prepare it for text mining or sentiment analysis
- Process email content to feed a machine learning algorithm that decides whether an email is spam
- Parse and extract specific data from a website to build a database

3. Why it is important
Learning to manipulate strings and master regular expressions will allow you to perform these tasks faster and more efficiently.

Strings

- Sequence of characters
- Quotes

```
my_string = "This is a string"
my_string2 = 'This is also a string'
```

```
my_string = 'And this? It's the wrong string'
```

```
my_string = "And this? It's the correct string"
```

4. Strings
The first step of our journey is strings, a data type used to represent textual data. Python recognizes any sequence of characters inside quotes as a single string object. As shown on the slide, both single or double quotes can be used. You should use the same quote type to open and close the string. If a quote is part of the string as seen in the code, we need to use the other quote type to enclose the string. Otherwise, python recognizes the second quote as a closing one.

More strings

- Length

```
my_string = "Awesome day"
len(my_string)
```

```
11
```

- Convert to string

```
str(123)
```

```
'123'
```

Concatenation

- Concatenate: `+` operator

```
my_string1 = "Awesome day"
my_string2 = "for biking"
```

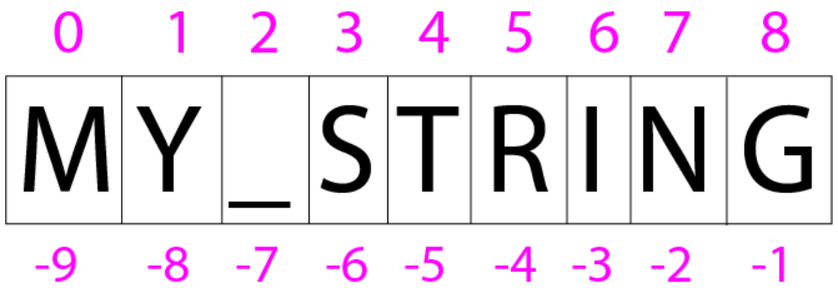
```
print(my_string1+" "+my_string2)
```

```
Awesome day for biking
```

5. More strings
Python has built-in functions to handle strings. Suppose we define the following string. We can get the number of characters in the string by applying the function len() which returns eleven as shown in the output. The function str() returns the string representation of an object as seen in the code.

Indexing

- **Bracket notation**




```
my_string = "Awesome day"
```

```
print(my_string[3])
```

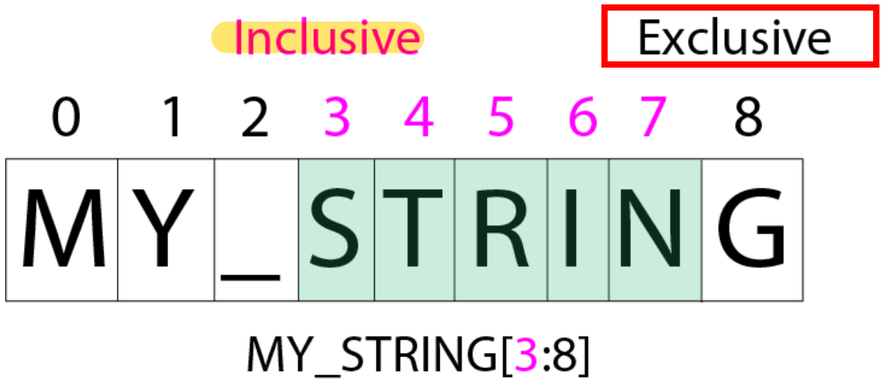
```
s
```

```
print(my_string[-1])
```

```
y
```

Slicing

- Bracket notation




```
my_string = "Awesome day"
print(my_string[0:3])
```

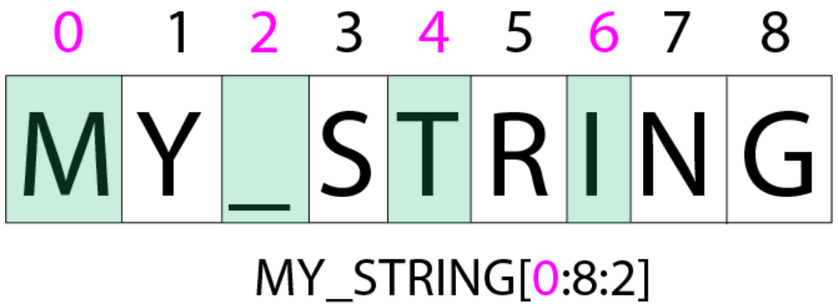
```
Awe
```

```
print(my_string[:5])
print(my_string[5:])
```

```
Aweso
me day
```

Stride

- Specifying stride




```
my_string = "Awesome day"
print(my_string[0:6:2])
```

```
Aeo
```

Reverse order!

```
print(my_string[::-1])
```

```
yad emosewA
```

9. Stride
String slicing also accepts a third index which specifies how many characters to omit before retrieving a character. In the example, the specified indices returns the following output. They are the characters retrieved between positions zero and six, skipping two characters in between. Interestingly, omitting the first and second indices and designating a minus one step returns a reversed string as shown in the output.

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

String operations

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

datacamp

Adjusting cases

```
my_string = "tHis Is a niCe StriNg"
```

- Converting to lowercase

```
print(my_string.lower())
```

```
this is a nice string
```

- Converting to uppercase

```
print(my_string.upper())
```

```
THIS IS A NICE STRING
```

```
my_string = "tHis Is a niCe StriNg"
```

- Capitalizing the first character

```
print(my_string.capitalize())
```

```
This is a nice string
```

Splitting

```
my_string = "This string will be split"
```

- Splitting a string into a list of substrings

```
my_string.split(sep=" ", maxsplit=2)
```

```
['This', 'string', 'will be split']
```

```
my_string.rsplit(sep=" ", maxsplit=2)
```

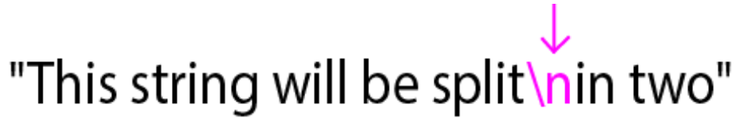
```
['This string will', 'be', 'split']
```

4. Splitting
We want to split the string into a list of substrings. Python provides us with two methods: dot split and dot rsplit. Both of them return a list.They both take a separating element by which we are splitting the string, and a maxsplit that tells us the maximum number of substrings we want. As we can see in the code, the difference is that split starts splitting at the left. rsplit begins at the right of the string. If maxsplit is not specified both methods behave in the same way. They give as many substrings as possible. If you want the split to be done by the whitespace you don't have to specify the sep argument.

```
my_string = "This string will be split\nin two"
print(my_string)
```

```
This string will be split
in two
```

Escape Sequence	Character
\n	Newline
\r	Carriage return



- Breaking at line boundaries

```
my_string = "This string will be split\nin two"
```

```
my_string.splitlines()
```

```
['This string will be split', 'in two']
```

6. splitlines
For this aim, Python has the method splitlines(). As we can see in the code, the string is split at the slash n sequence returning a list of two elements.

Joining

- Concatenate strings from list or another iterable

sep.join(iterable)

```
my_list = ["this", "would", "be", "a", "string"]
print(" ".join(my_list))
```

```
this would be a string
```

7. Joining
Some methods can paste or concatenate together the objects in a list or other iterable data. It first takes the separating element. Inside the call, we specify the list or iterable element. We can observe in the example, that whitespace is specified as a separator and the data type is a list. The result is a single string containing all the objects in the list separated by whitespace.

Stripping characters

- Strips characters from left to right: `.strip()`

```
my_string = " This string will be stripped\n"
```

```
my_string.strip()
```

```
'This string will be stripped'
```

8. Stripping characters
The dot strip method will remove both leading and trailing characters. Inside the call, we can specifying a character. If we don't do it, whitespace will be removed. We get a string where both the leading space and the trailing escape sequence were removed.

```
my_string = " This string will be stripped\n"
```

- Remove characters from the right end

```
my_string.rstrip()
```

```
' This string will be stripped'
```

- Remove characters from the left end

```
my_string.lstrip()
```

```
'This string will be stripped\n'
```

9. Stripping characters2
We can apply dot rstrip method and it will return a string where the trailing slash n was removed. If we apply the dot lstrip method, we'll get a string with the leading whitespace eliminated.

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Finding and replacing

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data scientist

datacamp

Finding substrings

- Search target string for a specified substring.

string.find(substring, start, end)

optional

```
my_string = "Where's Waldo?"
my_string.find("Waldo")
```

```
8
```

```
my_string.find("Wenda")
```

```
-1
```

2. Finding substrings
In the example code, we search for Waldo in the string Where's Waldo?. The dot find method returns the lowest index in the string where it can find the substring, in this case, eight. If we search for Wenda, the substring is not found and the method returns minus one.

Finding substrings

- Search target string for a specified substring.


```
my_string = "Where's Waldo?"
```

```
my_string.find("Waldo", 0, 6)
```

```
-1
```

3. Finding substrings
Now, we check if we can find Waldo between characters number zero and five. In the code, we specify the starting position, zero, and the ending position, six, because this position is not inclusive. The dot find method will not find the substring and returns minus one as we see in the output.

Index function

- Similar to `.find()`, search target string for a specified substring.


```
my_string = "Where's Waldo?"
my_string.index("Waldo")
```

```
8
```

4. Index function
The dot index method is identical to dot find. In the slide, we see that it takes the desired substring as mandatory argument. It can take optional starting and ending positions as well. In the example, we search again for Waldo using dot index. We get eight again. When we look for a substring that is not there, we have a difference. The dot index method raises an exception, as we can see in the output.

```
my_string.index("Wenda")
```

```
File "<stdin>", line 1, in <module>
ValueError: substring not found
```

Index function

- Similar to `.find()`, search target string for a specified substring.


```
my_string = "Where's Waldo?"
```

```
try:
    my_string.index("Wenda")
except ValueError:
      print("Not found")
```

5. Index function
We can handle this using the try except block. In the slide, you can observe the syntax. The try part will test the given code. If any error appears the except part will be executed obtaining the following output as a result.

```
"Not found"
```

Counting occurrences

- Return number of occurrences for a specified substring.


```
my_string = "How many fruits do you have in your fruit basket?"
my_string.count("fruit")
```

```
2
```

```
my_string.count("fruit", 0, 16)
```

```
1
```

6. Counting occurrences
The dot count method searches for a specified substring in the target string. It returns the number of non-overlapping occurrences. In simple words, how many times the substring is present in the string. The syntax of dot count is very similar to the other methods as we can observe. In the example, we use the dot count method to get how many times fruit appears. In the output, we see that is two. Then, we limit the occurrences of fruit between character zero and fifteen of the string as we can observe in the code. The method will return 1. Remember that starting position is inclusive, but the ending is not.

Replacing substrings

- Replace occurrences of substring with new substring.


```
my_string = "The red house is between the blue house and the old house"
print(my_string.replace("house", "car"))
```

```
The red car is between the blue car and the old car
```

```
print(my_string.replace("house", "car", 2))
```

```
The red car is between the blue car and the old house
```

7. Replacing substrings
As we see in the slide, it takes three arguments: the substring to replace, the new string to replace it, and an optional number that indicates how many occurrences to replace. In the example code, we replace the substring house with car. The method will return a copy with all house substrings replaced. In the next example, we specified that we only want 2 of the occurrences to be replaced. We see in the output that the method return a copy of the string with the first two occurrences of house replaced by car.

Wrapping up

- **String manipulation:**
  - Slice and concatenate
  - Adjust cases
  - Split and join
  - Remove characters from beginning and end
  - Finding substrings
  - Counting occurrences
  - Replacing substrings

8. Wrapping up
In this chapter, we walked through learning how to manipulate strings, a valuable skill for any data science project. You saw how to slide, concatenate and adjust cases of strings, how to split them into pieces and join them back together, how to remove characters and finally how to find, count and replace occurrences of substrings.

Let's practice!

REGULAR EXPRESSIONS IN PYTHON