8 String

Before dive deep in Python string, we need to have a concept of array. An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). In python, strings are arrays of bytes representing unicode characters. However, python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

```
a = "Hello, World!"
print(a[1]) # e
```

In the above example, the word "Hello, World!" is stored in the variable a. The first character of the string is accessed using the array index 1. The index starts from 0. So, the character of index 1 is "e". The index must be an integer. We can not use float or other types, this will result into errors. For a better visulization of the string, we can refer to the following figure.

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o	,		W	o	r	l	d	!

8.1 String Functions

8.1.1 len()

The `len()` function returns the length of a string.

```
a = "Hello, World!"
print(len(a)) # 13
```

8.1.2 strip()

The `strip()` method removes any whitespace from the beginning or the end.

```
a = " Hello, World! "
print(a.strip()) # "Hello, World!"
```

8.1.3 lower()

The `lower()` method returns the string in lower case.

```
a = "Hello, World!"
print(a.lower()) # "hello, world!"
```

8.1.4 upper()

The `upper()` method returns the string in upper case.

```
a = "Hello, World!"
print(a.upper()) # "HELLO, WORLD!"
```

8.1.5 replace()

The `replace()` method replaces a string with another string.

```
a = "Hello, World!"
print(a.replace("H", "J")) # "Jello, World!"
```

8.1.6 split()

The `split()` method splits the string into substrings if it finds instances of the separator. The default separator is any whitespace.

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

8.1.7 format()

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders {} are.

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age)) # My name is John, and I am 36
```

8.1.8 check string

To check if a certain phrase or character is present in a string, we can use the keywords `in` or `not in`.

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x) # True
```

8.1.9 concatenation

To concatenate, or combine, two strings you can use the + operator.

```
a = "Hello"
b = "World"
c = a + b
print(c) # HelloWorld
```

To add a space between them, add a ” ”:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c) # Hello World
```

8.2 string slicing

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string.

```
a = "Hello, World!"
print(a[2:5]) # llo
```

8.2.1 string slicing with negative index

Specify negative indexes if you want to start the slice from the end of the string.

```
a = "Hello, World!"
print(a[-5:-2]) # orl
```

8.2.2 string slicing with step

Specify the step of the slice. The step specifies the increment between the indexes to slice.

```
a = "Hello, World!"
print(a[::2]) # HloWrd
```

8.2.3 string slicing with negative step

Specify negative step to slice the string in reverse order.

```
a = "Hello, World!"
print(a[::-1]) # !dlroW ,olleH
```

8.2.4 string slicing with negative step and negative index

Specify negative step to slice the string in reverse order.

```
a = "Hello, World!"
print(a[-5:-2:-1]) #
```