IIT Madras
BSc Degree

Home Lesson-1.6

Lesson-1.5

Lesson-1.5
- Strings
  - Quotes: single, double and triple
  - Length
  - Operations on strings
    - Concatenation
    - Replication
    - Comparison
  - Escape characters
  - Substrings

Strings

Quotes: single, double and triple

We briefly looked at strings in the first lesson. A string is any sequence of characters enclosed within single or double quotes. Some examples:

```
"this is a string"
'this is also a string'
'1 + 1 = 2'
"!, ?, _, @ are special characters"
"if you need to use apostrophe ('), you can use double quotes"
```

It is a good practice to stick to either single or double quotes when using strings. Interestingly, Python also supports triple quotes `'''`, especially for multi-line strings, i.e., strings that span multiple lines. Let us say that we want the following lines to be captured in a single string:

```
first line
second line
third line
```

The following code will throw a `SyntaxError`:

```
x = 'first line
second line
third line'
print(x)
```

This is where `'''` comes in:

```
x = '''first line
second line
third line'''
print(x)
```

After executing the above code, head to the console and type `x`. You will see the following output:

```
'first line\nsecond line\nthird line'
```

The `\n` character that you see above is called a newline character. Head to the section on escape characters in this lesson to know more about them.

Length

The length of a string is the number of characters in it. Python provides a built-in function called `len` to find the length of a string:

```
x = 'good'
print(len(x))
```

The code given above will give 4 as the output. If you are familiar with other programming languages, such as C, you might be aware of a character data type. Python doesn't have a separate data type for characters. A character in Python is represented by a string of length 1. In the following examples, `x` and `y` are strings of length 1.

```
x = 'a'
y = 'b'
```

We can also define empty strings:

```
x = ''
print(len(x))
```

As expected, the length of the empty string is 0.

Operations on strings

Concatenation

We can concatenate two strings using the `+` operator. Concatenation is just a fancy term for joining two strings together:

```
string1 = 'first'
string2 = ','
string3 = 'second'
string4 = string1 + string2 + string3
print(string4)
```

The output is:

```
first,second
```

Replication

We can make multiple copies of a string and string them all together using the `*` operator:

```
s = 'good'
five_s = s * 5
print(five_s)
```

The is the output:

```
goodgoodgoodgoodgood
```

The `*` operator has made the string look too good! This is a fine demonstration of that ancient adage: "multiplication is repeated addition":

```
s = 'good'
s * 5 == s + s + s + s + s  # This expression evaluates to True
```

Comparison

We can compare two strings. To begin with, we have the `==` operator:

```
x = 'python'
print(x == 'python', x == 'nohtyp')
```

The output is:

```
True False
```

Two strings are equal if and only if both of them represent exactly the same sequence of characters. Now, consider the following lines of code:

```
print('good' > 'bad')
print('nine' < 'one' )
print('a' < 'ab' < 'abc' < 'b')
```

The output is:

```
True
True
True
```

It is clear from the above examples that the length of the string is not a metric used by Python to compare strings. Instead, Python uses the familiar alphabetical ordering to compare two strings. More precisely it employs what is known as lexicographic ordering:

> **Lexicographic ordering**
>
> The first characters from the two strings are compared. If they differ this determines the outcome of the comparison. If they are equal, then the second character of both the strings are compared. This process continues until either string is exhausted.

This leads to another question. How does Python compare two characters? The answer is given in one of Python's official tutorials:

Python's string type uses the Unicode standard for representing characters, which lets Python programs work with different possible characters. What is the Unicode standard? Unicode is a specification that aims to list every character used by human languages and give each character its own unique code. The Unicode standard describes how characters are represented by **code points**. Another unfamiliar term. What is a code point? A code point value is an integer. Lexicographical ordering for strings uses the Unicode code point number to order individual characters.

Python provides a built-in function called `ord` that returns the code point of any given character. For example:

```
print(ord('a'), ord('b'))
print(ord('a'), ord('A'))
```

The output is:

```
97 98
97 65
```

Now, we clearly see why `'a' < 'b'` returns `True`. This is because the code point for `'a'` and `'b'` are 97 and 98 respectively. As 97 < 98, `'a' < 'b'`. We can also infer that `'A' < 'a'` should return `True`.

Escape characters

In Python, the backslash - `\` - is called the escape character. One of its uses is to represent certain white-space characters such as tabs and newlines. We will look at them one by one using the following examples:

```
print('This is the first sentence.\nThis is the second sentence.')
```

The output is as follows:

```
This is the first sentence.
This is the second sentence.
```

`\n` is a newline character. Its effect is to introduce a new line. Note that even though there are two separate characters: `\` and `n`, `\n` is still regarded as a single character. To verify this, execute the following code. You should get 1 as the output.

```
x = '\n'
print(len(x))
```

Another useful character is the tab: `\t`:

```
print('a\tb')
```

This will give the output:

```
a	b
```

There is also a way to escape the quotes: `\'`. This can come in handy when using the apostrophe symbol in strings with single quotes:

```
print('India\'s capital is New Delhi')
```

This gives the output:

```
India's capital is New Delhi
```

Now remove the backslash from the above string and try to print it. You will be getting an error. Why do you think that happens?

Substrings

A string is a substring of another string if the first string is contained in the second. For example, `'good'` is a substring of `'very good'`, whereas `'very good'` is not a substring of `'verygood'`. Python provides a keyword - `in` - which can be used to check if a given string is a substring of another string. For example:

```
a = 'good'
b = 'very good'
present = a in b
print(present)
not_present = b in a
print(not_present)
```

This gives the output:

```
True
False
```

`in` is a powerful keyword which has several other uses. It can also be used along with `not` in the following manner:

```
a = 'abc'
b = 'ab'
print(a not in b)
```

This gives the output:

```
True
```