Strings

September 8, 2018

1 Strings

Strings are basically text data. It is sequence of different characters. e.g. "Hello" is sequence of 'H','e','l','l' and 'o'. We can use indexing and do slicing on strings.

Either single quote ' or double qoute "" can be used to form a string

```
In [1]: #Single quote
        'hello'

Out[1]: 'hello'

In [2]: 'hello world'

Out[2]: 'hello world'

In [3]: #double quote
        "hello world, this is a long sentence"

Out[3]: 'hello world, this is a long sentence'

In [4]: #if you have a double quote within your string - use string within single quote and vi
        'here is "double quote"'

Out[4]: 'here is "double quote"'

In [5]: "here is 'single quote'"

Out[5]: "here is 'single quote'"

In [8]: #Introducing new line in string
        "this is first line \n this is next line"

Out[8]: 'this is first line \n this is next line'

In [9]: "this is first line"
        "this is next line"

Out[9]: 'this is next line'
```

```
In [10]: #print function
         print("this is first line")
         print("this is next line")
```

```
this is first line
this is next line
```

```
In [11]: print("this is first line \n this is next line")
```

```
this is first line
 this is next line
```

```
In [12]: print("this is first line \t this is next line")
```

```
this is first line          this is next line
```

```
In [13]: print("this is first line \\ this is next line")
```

```
this is first line \ this is next line
```

```
In [17]: #ASCII carriage return (CR). Moves all characters after ( CR ) to
         #the beginning of the line while overriding same number of characters moved.
         print("12345678901234567891234567890 \r this is next line")
```

```
 this is next line
```

```
In [18]: #unicode database name -- \N{unicode database name}
         print("\N{Double exclamation mark}")
```

```
In [19]: print("\N{DAGGER}")
```

2 String Variable

```
In [4]: st = "Hello World!"
```

```
In [5]: st
```

```
Out[5]: 'Hello World!'
```

```
In [6]: print(st)
```

```
Hello World!
```

3 Indexing

Each character in the string gets an index number and it starts from 0

```
In [7]: st

Out[7]: 'Hello World!'

In [8]: st[0]

Out[8]: 'H'

In [9]: st[1]

Out[9]: 'e'

In [10]: st[8]

Out[10]: 'r'
```

4 Slicing

Getting parts of your string

```
In [11]: st

Out[11]: 'Hello World!'

In [12]: st[0:4]

Out[12]: 'Hell'

In [13]: st[0:5] #it is upto 5 but not inclusing 5th index

Out[13]: 'Hello'

In [15]: st[:5]

Out[15]: 'Hello'
```

it does not change your original string

```
In [14]: st

Out[14]: 'Hello World!'
```

slicing can be done backward also

```
In [18]: st[-1]

Out[18]: '!'
```

```
In [16]: st[:-1]

Out[16]: 'Hello World'

In [17]: st[:]

Out[17]: 'Hello World!'
```

slicing step wise

```
In [19]: st[:] #st[::1]

Out[19]: 'Hello World!'

In [20]: st[::1]

Out[20]: 'Hello World!'

In [21]: st[::2]

Out[21]: 'HloWrd'

In [22]: st[::-1]

Out[22]: '!dlroW olleH'
```

string concatenation - using '+'

```
In [24]: st + ' concatenated'

Out[24]: 'Hello World! concatenated'
```

until now we have not changed our original string

```
In [26]: st

Out[26]: 'Hello World!'

In [27]: st = st + ' concatenated'

In [28]: st

Out[28]: 'Hello World! concatenated'
```

string repetition using '*'

```
In [29]: st*3

Out[29]: 'Hello World! concatenatedHello World! concatenatedHello World! concatenated'
```