# 02-string

September 16, 2022

## 1 Strings

- Textual data in Python is handled with `str` objects, more commonly known as strings.
- They are `immutable` sequences of unicode code points.
- When it comes to store textual data though, or send it on the network, you may want to encode it, using an appropriate encoding for the medium you're using.
- String literals are written in Python using single, double or triple quotes (both single or double).

4 ways to define a string

[1]:
```
str1 = 'This is a string. We built it with single quotes.'
```

[2]:
```
 str2 = "This is also a string, but built with double quotes."
```

[3]:
```
str3 = '''This is built using triple quotes,
so it can span multiple lines.'''
```

[4]:
```
str4 = """This too
is a multiline one,
built with triple double-quotes."""
```

[5]:
```
str4
```

[5]:
```
'This too \nis a multiline one, \nbuilt with triple double-quotes.'
```

[6]:
```
print(str4)
```

```
This too
is a multiline one,
built with triple double-quotes.
```

[7]:
```
len(str4)
```

[7]:
```
63
```

As these are instances of the `str`class, they have associated methods and properties

[8]:
```
dir(str4)
```

```
[8]: ['__add__',
      '__class__',
      '__contains__',
      '__delattr__',
      '__dir__',
      '__doc__',
      '__eq__',
      '__format__',
      '__ge__',
      '__getattribute__',
      '__getitem__',
      '__getnewargs__',
      '__gt__',
      '__hash__',
      '__init__',
      '__init_subclass__',
      '__iter__',
      '__le__',
      '__len__',
      '__lt__',
      '__mod__',
      '__mul__',
      '__ne__',
      '__new__',
      '__reduce__',
      '__reduce_ex__',
      '__repr__',
      '__rmod__',
      '__rmul__',
      '__setattr__',
      '__sizeof__',
      '__str__',
      '__subclasshook__',
      'capitalize',
      'casefold',
      'center',
      'count',
      'encode',
      'endswith',
      'expandtabs',
      'find',
      'format',
      'format_map',
      'index',
      'isalnum',
      'isalpha',
      'isascii',
```

```
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

which can be called in a traditional OOP way

[9]:
```
str1.lower()
```

[9]:
```
'this is a string. we built it with single quotes.'
```

[10]:
```
str1.upper()
```

[10]:
```
'THIS IS A STRING. WE BUILT IT WITH SINGLE QUOTES.'
```

[11]:
```
str1.title()
```

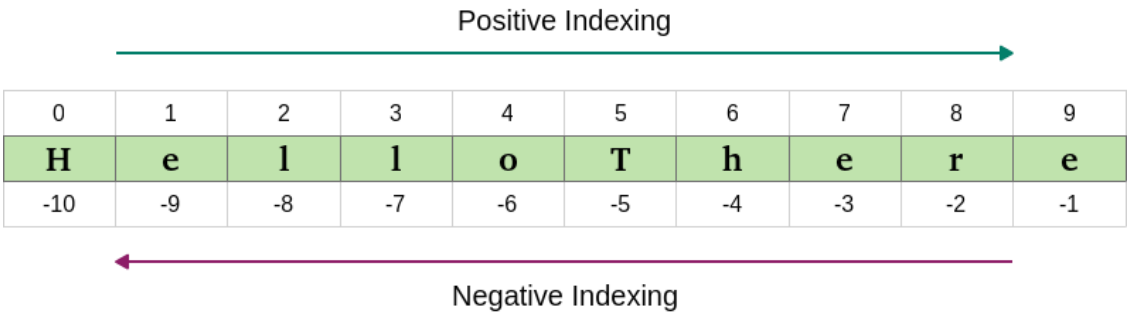
[11]:
```
'This Is A String. We Built It With Single Quotes.'
```

[12]:
```
str1.split()
```

[12]:
```
['This',
 'is',
 'a',
 'string.',
 'We',
 'built',
 'it',
 'with',
 'single',
 'quotes.']
```

### 1.1 Indexing and slicing

- When manipulating sequences, it's very common to have to access them at one precise position (indexing), or to get a subsequence out of them (slicing).
- When dealing with immutable sequences, both operations are read-only.
- When you get a slice of a sequence, you can specify the start and stop positions, and the step: `my_sequence[start:stop:step]`.




[13]:
```
s = 'Are you suggesting that coconuts migrate?'
```

[34]:
```
s[0] = 't'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [34], line 1
----> 1 s[0] = 't'

TypeError: 'str' object does not support item assignment
```

[35]:
```
s[0]
```

[35]:
```
'A'
```

[36]:
```
s[5]
```

[36]:
```
'o'
```

[37]:
```
s[:4]
```

[37]:
```
'Are '
```

[38]:
```
s[4:]
```

[38]:
```
'you suggesting that coconuts migrate?'
```

[39]:
```
s[4:14]
```

[39]:
```
'you sugges'
```

[40]:
```
list(zip(s, range(len(s))))
```

[40]:
```
[('A', 0),
 ('r', 1),
 ('e', 2),
 (' ', 3),
 ('y', 4),
 ('o', 5),
 ('u', 6),
 (' ', 7),
 ('s', 8),
 ('u', 9),
 ('g', 10),
 ('g', 11),
 ('e', 12),
 ('s', 13),
 ('t', 14),
 ('i', 15),
 ('n', 16),
 ('g', 17),
 (' ', 18),
 ('t', 19),
 ('h', 20),
 ('a', 21),
 ('t', 22),
 (' ', 23),
 ('c', 24),
 ('o', 25),
 ('c', 26),
 ('o', 27),
 ('n', 28),
 ('u', 29),
```

```
 ('t', 30),
 ('s', 31),
 (' ', 32),
 ('m', 33),
 ('i', 34),
 ('g', 35),
 ('r', 36),
 ('a', 37),
 ('t', 38),
 ('e', 39),
 ('?', 40)]
```

[41]:
```
s[4:14:3]        # slicing, start, stop and step (every 3 chars)
```

[41]:
```
'y gs'
```

[42]:
```
s
```

[42]:
```
'Are you suggesting that coconuts migrate?'
```

[43]:
```
s[-1]            # indexing at last position
```

[43]:
```
'?'
```

[44]:
```
s[-5:]
```

[44]:
```
'rate?'
```

[45]:
```
s[:-5]
```

[45]:
```
'Are you suggesting that coconuts mig'
```

[46]:
```
s[5:-5]
```

[46]:
```
'ou suggesting that coconuts mig'
```

[47]:
```
s[:]
```

[47]:
```
'Are you suggesting that coconuts migrate?'
```

[48]:
```
r = s
```

[49]:
```
id(s)
```

[49]:
```
2829234534320
```

[50]:
```
id(r)
```

[50]:
```
2829234534320
```

[51]:
```
s_copy = s[:5] + s[5:]
print(s_copy)
id(s_copy)
```

```
Are you suggesting that coconuts migrate?
```

[51]: 2829252726576

[52]:
```
s_copy = s[:]
id(s_copy)
```

[52]: 2829234534320

[53]:
```
s_copy = 'Are you suggesting that coconuts migrate?'
id(s_copy)
```

[53]: 2829234534800

[54]:
```
import copy
t = copy.deepcopy(s)
```

[55]:
```
id(t)
```

[55]: 2829234534320

### 1.2 Encode and decoding strings (optional)

- Using the encode/decode methods, we can encode unicode strings and decode bytes objects.
- Utf-8 is a variable length character encoding, capable of encoding all possible unicode code points.
- Notice also that by adding a literal b in front of a string declaration, we're creating a bytes object.

[56]:
```
s = "This is üñíçõe"            # unicode string: code points
s
```

[56]: 'This is üñíçõe'

[57]:
```
type(s)
```

[57]: str

[58]:
```
encoded_s = s.encode('utf-8')  # utf-8 encoded version of s
type(encoded_s)
```

[58]: bytes

[59]:
```
encoded_s
```

[59]: b'This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o'

[60]:
```
encoded_s.decode('utf-8')
```

[60]: 'This is üñíção'

[61]:
```
b"This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o"
```

[61]: b'This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o'

[62]:
```
b"This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o".decode('utf-8')
```

[62]: 'This is üñíção'

[63]:
```
"This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o".decode('utf-8')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In [63], line 1
----> 1 "This is \xc3\xbc\xc3\xb1\xc3\xad\xc3\xa7\xc3\xa3o".decode('utf-8')

AttributeError: 'str' object has no attribute 'decode'
```

## 2 Exercises

Go here...