Strings

In [1]:
```
my_string = 'Python is my favorite programming language!'
```

In [2]:
```
my_string
```

Out[2]:
```
'Python is my favorite programming language!'
```

In [3]:
```
type(my_string)
```

Out[3]:
```
str
```

In [4]:
```
len(my_string)
```

Out[4]:
```
43
```

In [5]:
```
'Long years ago we made a tryst with destiny, and now the time comes when we shall redeem our pledge.'
' India will awake to life and freedom.'
```

Out[5]:
```
'Long years ago we made a tryst with destiny, and now the time comes when we shall redeem our pledge. India will awake to life and freedom.'
```

Respecting PEP8 with long strings

In [6]:
```
long_story = ('Long years ago we made a tryst with destiny, and now the time comes when we shall redeem our pledge.'
              'not wholly or in full measure, but very substantially.At the stroke of the midnight hour, when the world sleeps,'
              ' India will awake to life and freedom.'
              'A moment comes, which comes but rarely in history, when we step out from the old to new, '
              'when an age ends, and when the soul of a nation, long suppressed, finds utterance...')
long_story
```

Out[6]:
```
'Long years ago we made a tryst with destiny, and now the time comes when we shall redeem our pledge.not wholly or in full measure, but very substantially.At the stroke of the midnight hour, when the world sleeps, India will awake to life and freedom.A moment comes, which comes but rarely in history, when we step out from the old to new, when an age ends, and when the soul of a nation, long suppressed, finds utterance...'
```

`str.replace()`

If you don't know how it works, you can always check the `help`:

In [7]:
```
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /)
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

This will not modify `my_string` because replace is not done in-place.

In [8]:
```
my_string.replace('a', '?')
print(my_string)
```

```
Python is my favorite programming language!
```

You have to store the return value of `replace` instead.

In [9]:
```
my_modified_string = my_string.replace('is', 'will be')
print(my_modified_string)
```

```
Python will be my favorite programming language!
```

`str.format()`

In [12]:
```
secret = '{} is cool'.format('C')
print(secret)
#my_string.format
```

```
C is cool
```

In [18]:
```
print('My name is {1} {0}, you can call me {0}.'.format('John', 'Doe', 'John'))
# is the same as:
print('My name is {first} {family}, you can call me {first}.'.format(first='John', family='Doe'))
```

```
My name is Doe John, you can call me John.
My name is John Doe, you can call me John.
```

`str.join()`

In [19]:
```
pandas = 'pandas'
numpy = 'numpy'
requests = 'requests'
cool_python_libs = ', '.join([pandas, numpy, requests])
```

In [20]:
```
print('Some cool python libraries: {}'.format(cool_python_libs))
```

```
Some cool python libraries: pandas, numpy, requests
```

Alternatives (not as Pythonic and slower):

In [21]:
```
cool_python_libs = pandas + ', ' + numpy + ', ' + requests
print('Some cool python libraries: {}'.format(cool_python_libs))

cool_python_libs = pandas
cool_python_libs += ', ' + numpy
cool_python_libs += ', ' + requests
print('Some cool python libraries: {}'.format(cool_python_libs))
```

```
Some cool python libraries: pandas, numpy, requests
Some cool python libraries: pandas, numpy, requests
```

`str.upper()`, `str.lower()`, `str.title()`

In [23]:
```
mixed_case = 'PyTHoN hackER'
```

In [24]:
```
mixed_case.upper()
```

Out[24]:
```
'PYTHON HACKER'
```

In [25]:
```
mixed_case.lower()
```

Out[25]:
```
'python hacker'
```

In [26]:
```
mixed_case.title()
```

Out[26]:
```
'Python Hacker'
```

In [28]:
```
mixed_case
```

Out[28]:
```
'PyTHoN hackER'
```

`str.strip()`

In [29]:
```
ugly_formatted = ' \n \t Some story to tell '
stripped = ugly_formatted.strip()

print('ugly: {}'.format(ugly_formatted))
print('stripped: {}'.format(ugly_formatted.strip()))
```

```
ugly:
         Some story to tell
stripped: Some story to tell
```

`str.split()`

In [30]:
```
sentence = 'three different words'
words = sentence.split()
print(words)
```

```
['three', 'different', 'words']
```

In [31]:
```
type(words)
```

Out[31]:
```
list
```

In [32]:
```
secret_binary_data = '01001,101101,11100000'
#Marchant_Calculate_Str_SecretBinaryData
binaries = secret_binary_data.split(',')
print(binaries)
```

```
['01001', '101101', '11100000']
```

Calling multiple methods in a row

In [34]:
```
ugly_mixed_case = '   ThIS LooKs BAd '
pretty = ugly_mixed_case.strip().lower().replace('bad', 'good')
print(pretty)
#print("%d%d%d", a++, ++a,++a++)
```

```
this looks good
```

Note that execution order is from left to right. Thus, this won't work:

In [35]:
```
pretty = ugly_mixed_case.replace('bad', 'good').strip().lower()
print(pretty)
```

```
this looks bad
```

Escape characters

In [36]:
```
two_lines = 'First line\nSecond line'
print(two_lines)
```

```
First line
Second line
```

In [37]:
```
indented = '\tThis will be indented'
print(indented)
```

```
        This will be indented
```

In [ ]:
```

```