1. Fill missing pieces

Fill `____` pieces below to have correct values for `lower_cased`, `stripped` and `stripped_lower_case` variables.

In [18]:

```
original = ' Python strings are COOL! '
lower_cased = original.lower()
stripped = original.strip()
stripped_lower_cased = original.strip().lower()
```

Let's verify that the implementation is correct by running the cell below. `assert` will raise `AssertionError` if the statement is not true.

In [19]:

```
assert lower_cased == ' python strings are cool! '
assert stripped == 'Python strings are COOL!'
assert stripped_lower_cased == 'python strings are cool!'
```

2. Prettify ugly string

Use `str` methods to convert `ugly` to wanted `pretty`.

In [20]:

```
ugly = ' tiTle of MY new Book\n\n'
```

In [21]:

```
# Your implementation:
pretty = ugly.title().format().strip()

print(pretty)
```

```
Title Of My New Book
```

Let's make sure that it does what we want. `assert` raises `AssertionError` (https://docs.python.org/3/library/exceptions.html#AssertionError) if the statement is not `True`.

In [22]:

```
print('pretty: {}'.format(pretty))
assert pretty == 'Title Of My New Book'
```

```
pretty: Title Of My New Book
```

3. Format string based on existing variables

Create `sentence` by using `verb`, `language`, and `punctuation` and any other strings you may need.

In [23]:

```
verb = 'is'
language = 'Python'
punctuation = '!'
```

In [24]:

```
# Your implementation:
fword = 'Learning'
fun = 'fun'

sentence = fword + " " + language + " " + verb + " " + fun + punctuation
```

In [25]:

```
print('sentence: {}'.format(sentence))
assert sentence == 'Learning Python is fun!'
```

```
sentence: Learning Python is fun!
```

In [ ]:

```

```