# EPAM Python Software Engineer Training

## Lesson 2: Python strings

# Contents



# 1 Course

## 1.1 String types

In python there are two string types *str* and *unicode* which are both derived from a *basestring*. It is posible to define a custom string type by inheriting from one of those classes. For most trivial tasks *str* is enough, even for a unicode string in encoded form, while a *unicode* type shall be used when an encoding shall be taken into account (e.g. when a string slicing shall occur). All strings in python are immutable meaning that they are hashable.

There is a *StringIO* class that implements a string buffer, however, if one really needs a fast implementation for some algorithm, the best way would be to off-load this task to some C module (e.g. written in Cython).

## 1.2 String literals

There are 6 ways in total to represent a string in python:

1. Quoted:

```
s = 'It\'s a test python string\nwith a "newline" character.'
```

1. Raw quoted:

```
# Note that strings are concatenated if put one after one.
s = r'It\'s a test python string\nwith two backslashes an no "newline" ' \
    r'character.'
```

1. Tripple quoted:

```
s = '''It's a test python string
       with a "newline" character and leading white-space.'''
```

1. Double-quoted:

```
s = "It's a test python string\nwith a \"newline\" character."
```

1. Raw double-quoted:

```
s = r"It's a test python string with two \"backslash\" characters."
```

1. Tripple double-quoted:

```
s = """It's a test python string\nwith a "newline" character."""
```

For string literals it is preferred to use *quoted* strings even if an escaping shall be done while for doc-strings *tripple double-quoted* strings are preferred. Unicode string literals follow the same rules and are prefixed with an *u* character (e.g. `ur'test unicode: \u1234'`).

# 2 Tasks

## 2.1 Cheshire Cat

Print this ASCII graphic using 4 different string literal types:

```
_-_-_-_-_-_-_-_
\           /
|  ^_____^  |
/  (.) (.)  \
|  ( t   )  |  Miaowww
\    ==/    /
|           |
'"'"'"'"'"'"'"'
```

## 2.2 String Joining

Write a function that takes arbitrary number of positional and named arguments and returns a string in the following format (use only a `str.join` method for this):

```
pos1_value, pos2_value, pos3_value
named1=named1_value, named2=named2_value
```

## 2.3 String Formatting

Implement the above task using a `str.join` and a `str.format` methods.

## 2.4 String Splitting

Write a function that restores function arguments off the output of the previous function into a tuple of positional and a dictionary of named arguments. For simplicity assume that argument values are all strings that does not contain special characters. Use a `str.split` method for this task.

## 2.5 Advanced String Splitting

Write a function that takes a comma-separated string and returns a last element (separated by a last comma) or the entire string if there is no comma in it.

## 2.6 String replace

Write a function which takes a string and replaces all vocal letters in it to an uppercase using a `str.replace` method.

## 2.7 String slicing

Write a function which takes a string and returns the first 10 characters off it concatenated with the last 10 characters.

## 2.8 Unicode

Write a function which takes a unicode string and encodes it into a printable US-ASCII character set. Use some real *UTF-8* characters to test this function.