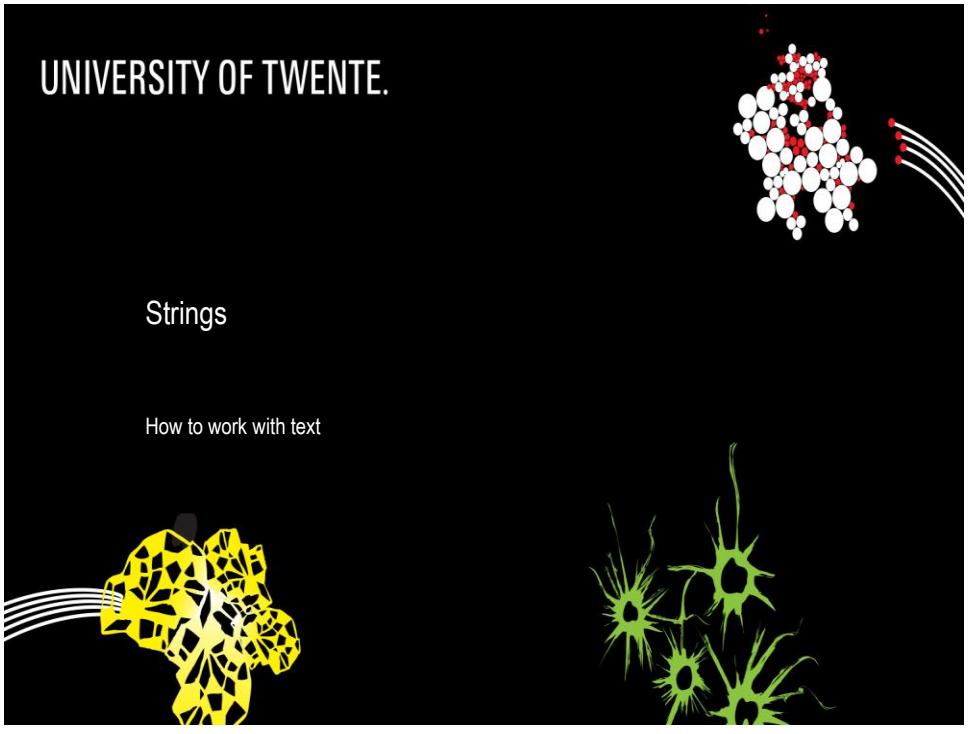
UNIVERSITY OF TWENTE.

# Strings

How to work with text



1

## Contents

- Text strings and how they are represented
- Operations with strings
- String methods
- Transformations
- Iterations over strings

ITC UNIVERSITY OF TWENTE.

2

## Built-in Python data structures

Python knows a number of built-in *compound* data types (containers), used to group objects

**Sequences**

- Types: **strings**, **lists (done!), tuples (done!)**
- Operations: Indexing, slicing, adding, multiplying, iteration & membership (also valid here)

**Dictionaries**

- Map keys to values through index
- Suitable for unstructured data

**Sets**

- Unordered and do not map keys to values



## Basics

A string is a sequence of characters delimited by quotes (" … " or ' … ')

```
"This is a string with double quotations"
'Now with single quotes'
"We can use * special & characters + inside"
'Even "quotes" within strings'

mystring =    "we can define " \
              "strings spanning " \
              "multiple lines " \
              "as one" \
print(mystring)
```

*we can define strings spanning multiple lines as one*

## Basics

Strings are immutable (like tuples), but they can be copied (through slicing)

```
fruit = "strawberry"
fruit[4]                                    # Same as in lists
w
fruit[2] = "Q"
TypeError: object does not support item assignment
```

You can calculate the length of a string, because it is a sequence

```
fruit = "coconut"
len(fruit)
7

len("1 2 3")                                # Spaces are characters also
5
```

UNIVERSITY OF TWENTE.

5

## Operations

- Strings are sequences!
- Thus, operations similar to lists can be applied
  - Concatenation
  - Multiplication
  - Slicing
  - Comparison
  - Membership

UNIVERSITY OF TWENTE.

6

## String methods

Methods are functions that are tightly associated with an object type

Similar to lists, they are called like: **`object.method(arguments)`**

Check full reference online, or in the course materials

- There are many!
- w3schools on python string methods



## String methods

| | | |
|---|---|---|
| *.count(c)* | → | *"pineapple".count("p")* |
| *.find(c)* | → | *"pineapple".find("a")* |
| *.replace(s1, s2)* | → | *"pineapple".replace("apple", "kiwi")* |
| *.upper()* | → | *"pineapple".upper()* |
| *.lower()* | → | *"PineAPPLE".lower()* |
| *.startswith(s)* | → | *"pineapple".startswith("pin")* |
| *.center(n)* | → | *"pineapple".center(21)* |
| *.zfill(n)* | → | *"pineapple".zfill(15)* |
| *.rjust(n)* | → | *"pineapple".rjust(15)* |

## String methods

| | | |
|---|---|---|
| *.isalnum()* | → | *"pine444apple".isalnum()* |
| | → | *"pine$$$apple".isalnum()* |
| *.isalpha()* | → | *"pineapple".isalpha()* |
| | → | *"pine444apple".isalpha()* |
| *.islower()* | → | *"pineapple".islower()* |
| | → | *"pineAPPLE".islower()* |
| *.isupper()* | → | *"PINEAPPLE".isupper()* |
| | → | *"pineapple".isupper()* |
| *.isspace()* | → | *"  ".isspace()* |
| | → | *"  pineapple  ".isspace()* |

UNIVERSITY OF TWENTE.

16

## Transformations: list to str

```
l = ['p', 'i', 'n', 'e', 'a', 'p', 'p', 'l', 'e]
str(l)                                          # one thinks this, but …
"['p', 'i', 'n', 'e', 'a', 'p', 'p', 'l', 'e]"
```

UNIVERSITY OF TWENTE.

17

## Transformations: list to str (merging)

So, the function *str()* does not provide a direct transformation from list to string

- Solution: Use another string method: *.join(List)*

```
l = ['p', 'i', 'n', 'e', 'a', 'p', 'p', 'l', 'e']
"".join(l)
'pineapple'
```

When *s* is a string and L is a list of strings $[l_1, l_2, l_3, \ldots]$, the expression *s.join(L)* constructs the string $l_1 s l_2 s l_3 s \ldots$ Above, s is the empty string.

Thus:

```
'', ''.join(l)
'p, i, n, e, a, p, p, l, e'
```



## Transformations: splitting

Splitting is the contrary operation to merging

- Use the string method *.split(char)* for this purpose

```
berries = "cranberry raspberry blueberry strawberry"
split_berries = berries.split(" ")
print(split_berries)
['cranberry', 'raspberry', 'blueberry', 'strawberry']
```

- The output is a list
- We split the original string by the string consisting of just the space character

## Transformations: splitting and merging

Put together:

```
berries = [ "cranberry", "raspberry", "blueberry", "strawberry"]
berries_str = "-->".join(berries)
print(berries_str)
'cranberry-->blueberry-->raspberry-->strawberry'

berries = berries_str.split("-->")
print(berries)
['cranberry', 'raspberry', 'blueberry', 'strawberry']
```



## Transformations: reversing a string

```
mandarin = "mandarin is a type of orange"
print(mandarin[::-1])
'egnaro fo epyt a si niradnam'
```

Can anyone explain how this works?

## Summary

- Strings are sequences of characters
- Characters are accesed by an index (indexing)
- String segments are accessed by slices (slicing)
- Strings can be concatenated and multiplied
- Loops (while and for) can access each character (iteration)
- Operator "in" checks for membership
- Plenty of string methods to ease its handling! (use a Google search top find alternatives)