Appendix 1: Strings

Advanced string operations

Concatenation: `+`

For string the `+` operator is used for concatenation, joining multiple strings together.

In [1]:

```
word1 = 'Hello'
word2 = 'Python'
greet = word1 + ' ' + word2 + '!'
print(greet)
```

```
Hello Python!
```

Multiplication: `*`

The `*` operator is used for "multiplying" a string, repeating and concatenating it the given times.

In [2]:

```
greet3times = greet * 3
print(greet3times)
```

```
Hello Python!Hello Python!Hello Python!
```

Length: `len()`

The `len()` statement returns the length of the string.

In [3]:

```
print(len(greet))
```

```
13
```

String indexing and slicing: `[]`

A single charcter of a string can be access by indexing it, starting from zero:

In [4]:

```
print(greet[0])
```

```
H
```

Question: what will happen if we index with a negative number?

In [5]:

```
print(greet[-1])
```

```
!
```

Question: what will happen if we with a number larger than the length of the string?

In [6]:

```
print(greet[100])
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-6-eb8fbb2c6e43> in <module>
----> 1 print(greet[100])

IndexError: string index out of range
```

We can also create substrings by fetching a slice of a string.
Note that the end index is exclusive, so if the slice is given as `[4:6]` , then the characters with the index 4 and 5 will be sliced.

In [7]:

```
print(greet[0:5])
print(greet[6:7])
```

```
Hello
P
```

The first (start) index can be omitted, by default it will be zero:

In [8]:

```
print(greet[:5])
```

```
Hello
```

The second (end) index can also be omitted, by default it will be the end of the string:

In [9]:

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

```
Python!
```

Question: what happens if we omit both the start and the end index?

In [10]:

```
print(greet[:])
```

```
Hello Python!
```

Question: what happens if we use negative indices?

In [11]:

```
print(greet[-7:])
print(greet[1:-2])
```

```
Python!
ello Pytho
```

Question: what happens if the end index is larger than the length of the string?

In [12]:

```
print(greet[6:100])
```

```
Python!
```

Built-in string functions

A comprehensive list of the built-in functions can be found in the ['string library' (https://docs.python.org/3/library/stdtypes.html#string-methods)](https://docs.python.org/3/library/stdtypes.html#string-methods) reference documentation.

These string functions are *methods*, which means they can be called on a string instance (value or variable) in a form `stringvar.method(parameters)` . They do not modify the original string, but return a new instance.

Lowercase: `lower`

Replace all letters to lowercase.

In [13]:

```
print(greet)
greet_lower=greet.lower()
print(greet_lower)
```

```
Hello Python!
hello python!
```

Uppercase: `upper`

Replace all letters to uppercase.

In [14]:

```
print(greet)
greet_upper=greet.upper()
print(greet_upper)
```

```
Hello Python!
HELLO PYTHON!
```

Capitalization: `capitalize` and `title`

Replace the very first letter or the first letter of each words to uppercase. The rest will be turned to lowecase.

In [15]:

```
print(greet_lower)
greet_capital=greet_lower.capitalize()
print(greet_capital)

greet_title=greet_lower.title()
print(greet_title)
```

```
hello python!
Hello python!
Hello Python!
```

Substring search: `find`

Looks up the first occurance of a character or a substring in a string. The result is the starting index position of the first occurance as an `integer` . Keep in mind that the first index is `0` ! The returned value is `-1` if the substring was not found.

In [16]:

```
print(greet)
location = greet.find('Python')
print(location)

print(greet)
location = greet.find('java')
print(location)
```

```
Hello Python!
6
Hello Python!
-1
```

The starting index of the search can also be passed to the function. This way multiple occurances of a substring can be looked up.

In [17]:

```
print(greet3times)
location = greet3times.find('Python')
print(location)

location = greet3times.find('Python', location + 1)
print(location)
```

```
Hello Python!Hello Python!Hello Python!
6
19
```

This function is case-sensitive.
If you would like to search for both lower and uppercase variants, you may convert the string to lowercase first!

In [18]:

```
print(greet)
location = greet.find('python')
print(location)

print(greet.lower())
location = greet.lower().find('python')
print(location)
```

```
Hello Python!
-1
hello python!
6
```

Substring replace: `replace`

Replace **all** occurances of a substring to another substring.

This function is also case-sensitive.

In [19]:

```
greet_alternative = greet3times.replace('Hello', 'Hi')
print(greet_alternative)
```

```
Hi Python!Hi Python!Hi Python!
```

Stripping: `lstrip`, `rstrip`, `strip`

All functions are used to trim unrequired whitespace characters (spaces, tabulators, newlines) from a string.

- `lstrip` - remove whitespace characters from the lefthand side.
- `rstrip` - remove whitespace characters from the righthand side.
- `stri` - remove whitespace characters from both sides.

In [20]:

```
greet_world = '   --== Hello World  ==-- '
print(greet_world.lstrip())
print(greet_world.rstrip())
print(greet_world.strip())
```

```
--== Hello World  ==--
   --== Hello World  ==--
--== Hello World  ==--
```

The characters to remove can also be specified otherwise:

In [21]:

```
print(greet_world.strip(' -='))
```

```
Hello World
```

Prefix and suffix check: `startswith`, `endswith`

These functions verifies whether a string starts or ends with the given substring. The result is a *boolean* value (`True` or `False` .)

This function is also case-sensitive.

In [22]:

```
print(greet.startswith('Hello'))
print(greet.startswith('Hi'))
```

```
True
False
```

Splitting: `split`

Split a string into a list of substring by defining a so-called *separator* or *delimiter* character or string. The *separator* is removed from the string.

In [23]:

```
print(greet3times)
words = greet3times.split('!')
print(words)
```

```
Hello Python!Hello Python!Hello Python!
['Hello Python', 'Hello Python', 'Hello Python', '']
```

Question: why is there an empty string at the end of the result list?

Logical operations on strings

Containment check: `in`

Verify whether a letter or a substring occures *anywhere* inside a string. The result is a *boolean* value (`True` or `False` .)

In [24]:

```
print('p' in greet)
print('P' in greet)
print('Python' in greet)
```

```
False
True
True
```

In [25]:

```
if 'P' in greet:
    print('Contains a letter P!')
```

```
Contains a letter P!
```

Equality check: `==`

Perform a case-sensitive equality check between two strings.

In [26]:

```
if word2 == 'Python':
    print('It was Python.')
else:
    print('It was not Python.')
```

```
It was Python.
```

Summary exercise on strings

Task: request the name, birth year, email address and spoken languages of the user. The spoken languages are requested as a string, separated by commas.

Check whether the following validation rules are matched. If any of the data is invalid, display an error message and request a repeated entry of the data.

- The name must contain at least 2 parts. (There must be a space inside it.)
- The birth year must be a number, between 1900 and 2019.
- The email address must contain a `@` letter and must end with a `elte.hu` domain.

When the data was given successfully, trim any unncceseary whitespaces and display it in a corrected format:

- The name shall be displayed with each part starting with a capital letter.
- Beside the birth year, calculate the (possible) age of the current user.
- The email address shall be lowercase.
- The spoken languages shall be displayed as a list of languages instead of a single string.

In [27]:

```python
import datetime

def valid_name(name):
    name = name.strip()
    return len(name.split(' ')) >= 2

def valid_birthyear(year):
    year = year.strip()
    try:
        year_num = int(year)
        return year_num >= 1900 and year_num <= 2019
    except:
        return False

def valid_email(email):
    email = email.strip()
    return '@' in email and email.endswith('elte.hu')

def format_name(name):
    return name.strip().title()

def format_age(year):
    now = datetime.datetime.now()
    age_max = now.year - int(year)
    age_min = max(age_max - 1, 0)
    if age_max != age_min:
        return str(age_min) + "/" + str(age_max)
    else:
        return str(age_max)

def format_email(email):
    return email.strip().lower()

def format_langs(langs):
    langs = langs.strip().split(',')
    langs = list(map(str.strip, langs))
    return langs

name = input("Name: ")
while not valid_name(name):
    print("Incorrect format for name.")
    name = input("Name: ")

birthyear = input("Birth year: ")
while not valid_birthyear(birthyear):
    print("Incorrect format for birth year.")
    birthyear = input("Birth year: ")

email = input("Email: ")
while not valid_email(email):
    print("Incorrect format for email.")
    email = input("Email: ")

langs = input("Spoken languages: ")

print("Name: %s" % format_name(name))
print("Birth year: %s (age: %s)" % (birthyear, format_age(birthyear)))
print("Email: %s" % format_email(email))
print("Languages: %s" % format_langs(langs))
```

```
Incorrect format for name.

Incorrect format for birth year.

Incorrect format for email.

Incorrect format for email.

Name: John Smith
Birth year: 1985 (age: 35/36)
Email: johnsmith@elte.hu
Languages: ['english', 'german', 'hungarian']
```