1. String methods

Like any programming language, Python allows many operations on strings. Finding sub-strings, splitting, joining, etc. You can find a list of the available methods here.

Exercise

Use the appropriate methods to make the following lines of code work.

In [206…

```
string = "In computer programming,  a string is traditionally  a sequence of charac

print(string.find('c'))                          # index of the first 'c'
print(len(string)-1 - int(string[::-1].find('c'))) # index of the last 'c' (see r{
print(len(string.removesuffix('  ')))            # length of the string without trailin
print(string.startswith('In'))                   # whether the string starts with "In"
print(string.lower())                            # string as all lower-case
print(string.split(','))                         # list of parts of the sentence, split
print(string.replace('  ', ' '))                 # all double whitespaces replaced by a
print(string.replace(' traditionally ', ''))  # without the word "traditionally" (
```

```
3
72
78
True
in computer programming,  a string is traditionally  a sequence of characters.
['In computer programming', '  a string is traditionally  a sequence of character
s.  ']
In computer programming, a string is traditionally a sequence of characters.
In computer programming,  a string is a sequence of characters.
```

2. String formatting</h2>

Formatting a string allows you to export or print data. For example, printing the string `Client name: %s` where `%s` is formatted to be the name of a client given as a string. Besides substituting strings at `%s`, other data types can also be formatted in to the string. See here for a list of all formatting conversions. This includes formatting/rounding numbers.

A general way to format a string is given below. Note the `%d` for an integer. In case of a single argument, the `( )` are not nessecary.

In [207…

```
client_name = "Obelix"
client_age = 32                                          # [years]
string = "Client %s is %d years old." % (client_name, client_age) # the format is:
print(string)
```

```
Client Obelix is 32 years old.
```

Exercise

Use the appropriate format to make the following lines of code work.

In [208…

```
value = 1.73456
print("%d      " % value)   # 2         (see "5. Precision", why can't you use %d?)
print("%.1f " % round(value, 1))    # 1.7
print("%.2f   " % round(value, 2))    # 1.73
print("%f    " % round(value, 2))   #    1.73  (with a total length of 7, see "4. I
print("000%.2f     " % value)  # 0001.73  (see Flag '0')
print("+%.2f    " % value)   # +1.73    (see Flag '+')
print("+00%.2f     " % value) # +001.73
print("%.2e    " % value)   # 1.73e+00 (exponential format)
```

```
1
1.7
1.73
1.730000
0001.73
+1.73
+001.73
1.73e+00
```

3. Regular expressions

Regular expressions are used to find patterns in text, without exactly specifying each character. For example to find words, to find numbers that were formatted in a particular way, etc.
A single digit can for example be matched with `\d`. That would match at 4 locations in the string `The width of the car is 2m, and the height is 1.65m.`.
Another example is that we can match a set of characters. This can be matched using `[xyz]`. That would match at 4 locations in the string `If x = 2y, than y = 6z.`.
At Python Regular Expressions more information can be found on matching string patterns in Python. Using this information, make the following assignment.

Exercise
Consider the 12 lines in the code box below. You will have to find a pattern that:

- Matches the first 10 lines with a decimal number.
- Does not match the integer in the 11th line.
- Does not match the text in the 12th line.

1. Before you read the documentation, write the pattern down in words under 'pattern in words:'

   Tip: The lines to match have a general pattern of 3 elements (first this, than that, and finally that).

Open regex101.com.
On the left-hand side, select the "Python" flavor.
Copy the 12 lines from the code box below in the "TEST STRING" box.
In the "REGULAR EXPRESSION" text box, test and write your pattern.

1. Write the pattern down in code, regexp = ...

    Tip: Some elements may be "zero or more of X", where X is some type.

    Tip: Some elements may be "any of Y", where Y is a set of cases it can be.

```
0001,2345
1,2345
1,23
,2345
1,
001.2345
1.2345
1.23
.2345
1.
1
thisisnotanumber
```

pattern in words:

first ... than ... finally

In [209…

```
import re
regexp = "\d*[,\.]\d*"
```

4. Counting characters

Exercise

Print all non-zero frequencies of each character from the alphabet in the text given in the code box.

- Treat accented characters as normal characters.
- Combine uppercase and lowercase characters in a single count.
- Print in alphabetical order.
  </ul> *Hint: Have one step where you prepare and filter some data, and a second step with a loop.*
  Hint: sets have unique values, and lists are indexed and can thus be sorted (sort()).

In [210…

```
import string
from unidecode import unidecode

text = "For the movie The Theory of Everything (2014), Jóhann Jóhannsson composed 

alphabet = dict.fromkeys(string.ascii_lowercase , 0)
text_decoded = unidecode(text).lower()

#print(text_decoded)

for letter in text_decoded:
    for key, value in alphabet.items():
        if letter == key:
            alphabet[letter] = alphabet[letter] + 1
        else:
            continue


print(alphabet)
```

```
{'a': 3, 'b': 0, 'c': 1, 'd': 2, 'e': 12, 'f': 3, 'g': 2, 'h': 8, 'i': 3, 'j': 2,
'k': 0, 'l': 1, 'm': 3, 'n': 8, 'o': 12, 'p': 1, 'q': 0, 'r': 4, 's': 5, 't': 6,
'u': 1, 'v': 3, 'w': 0, 'x': 0, 'y': 2, 'z': 0}
```

5. Good... afternoon?

The code below generates a random time in the day. Suppose we want to present a user a welcoming message when the user opens a program at that time.

Exercise

- *Print a message with the (pseudo) format: Good {part of day}, the time is hh:mm*
- *Parts of the day are night [0-5], morning [6-11], afternoon [12-17] or evening [18-23].*
- *Hour or minute values below 10 should have a leading 0.*

Hint: you can use if-elif-else for the part of the day, but you can also have a fixed list of parts of the day and use clever indexing from the hour value.

In [211…

```
import random

h = random.randint(0, 23) # hour of the day
m = random.randint(0, 59) # minute in the hour
day_part = ''

if h in [i for i in range(6, 11 + 1)]:
     day_part = 'morning'

if h in [i for i in range (12, 17 + 1)]:
     day_part = 'afternoon'

if h in [i for i in range (18, 23 + 1)]:
     day_part = 'evening'

if h in [i for i in range(0, 5 + 1)]:
    day_part = 'night'

if h < 10:
    h = '0' + str(h)
else:
    h=h

if m < 10:
    m = '0' + str(m)
else:
    m=m

#print(day_part)
#print(h)


print(" Good {day_part}, the time is {h}:{m}".format(day_part=day_part, h=h, m=m))
```

```
Good morning, the time is 09:36
```