Exercise set 5

Repetition statements and collections (**for, while - list, tuple, dict**)

1. a) Create an application that prints the numbers 1-50 on separate lines. Use the **while**-statement.

   b) Do the same as in the previous task, but use **for**-statement instead.

   c) Create a loop which sums up all numbers in the range of 1-30. Print **ONLY** the end result. Use any repetition statement.

   d) Create a loop which prints all years between 2005 and 2010 **on the same line**. Use any repetition statement.

   Example of the application running:

```
42
43
44
45
46
47
48
49
50
Sum: 465
2005 2006 2007 2008 2009 2010
```

   Filename of the exercise = ***exercise5_1.py***

   Typical code amount : ***14-20 lines*** (empty lines/comments not included)

2. Create an application that asks the user 12 times the rain amount of the month (basically each month of a year). Based on the 12 values, calculate and print the average rain amount rounded to one decimal.

   Note: Use a loop to ask the rain amounts! (for example, a for-loop is perfect). Check out the repetition statement materials, page 41.

   Tip: average = total rain amount / amount of months

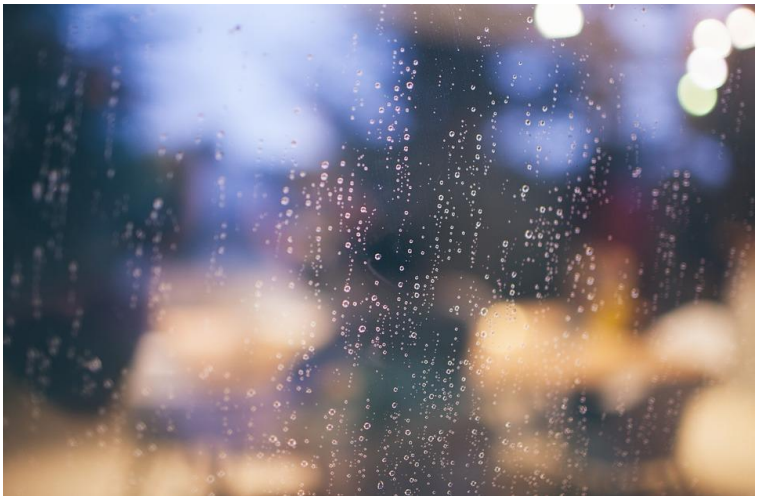
   The easiest way is to increase the total rain amount -variable in a for-loop! (see exercise 5-1, part c))

   Example of the application running

```
Give rain amount of month:
75
Give rain amount of month:
60
Give rain amount of month:
56
Give rain amount of month:
50
Give rain amount of month:
40
Year average for rain = 47.7 mm
```

   Filename of the exercise = ***exercise5_2.py***

   Typical code amount : ***7-12 lines*** (empty lines/comments not included)



3. Create an application that **asks the user for the radius of a circle**, and then calculates the area of the circle and prints the result on screen. **After this, ask the user whether they want to continue using the program or not (y/n).**

   If the user inputs anything else than **"n",** start the application from the start (ask the radius etc.)

   If the user inputs **"n"**, stop the application.

   Round the circle area into two decimals.

   Reminder: Circle area: πr^2

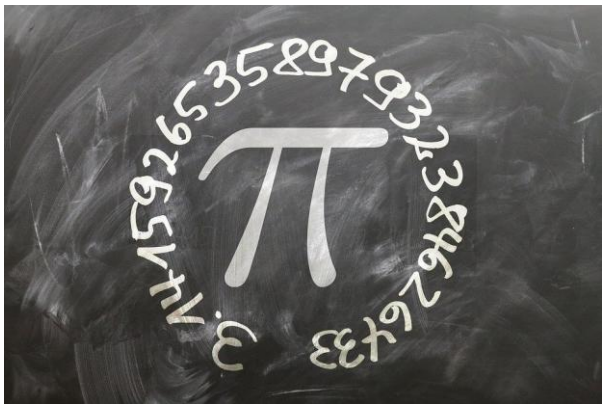
   Example of the application running:

```
Give radius:
15
Circle area: 706.86
Do you want to continue? (y/n)
y
Give radius:
7
Circle area: 153.94
Do you want to continue? (y/n)
n

Process finished with exit code 0
```

   Filename of the exercise = ***exercise5_3.py***

   Typical code amount : ***10-16 lines*** (empty lines/comments not included)



4. Create an application that contains the following **list**:
    - Rome
    - Athens
    - Stockholm
    - London
    - Dublin
    - Paris

   Print the contents of the list **by using a loop**. Print also the row number of each city, starting from 1.

   Tip: You can use the index number to calculate the row number!

   Small extra task: Sort the list alphabetically by using Python before printing!

   Example of the application running:

```
1: Rome
2: Athens
3: Stockholm
4: London
5: Dublin
6: Paris
```

   Filename of the exercise = ***exercise5_4.py***

   Typical code amount : ***4-12 lines*** (empty lines/comments not included)

5. Create an application that contains a **tuple,** which contains the names of months (January, February, March etc.). Ask the user for a number (1-12), and print the corresponding month name by using the **tuple.**

   Note: collections start from 0, not 1!

   Tips: When the user's input number has been converted to int(), you can substract - 1 from it and the index will match with the name of the month!

   You could technically also add an "empty" element as the first element of the tuple, but it's considered a less optimal solution and can be a bit confusing for other programmers.

   Example of the application running

```
Give the number of month:
4
April
```

   Filename of the exercise = ***exercise5_5.py***

   Typical code amount : ***3-6 lines*** (empty lines/comments not included)

6. Create an application that contains the following ***dictionary*** (**dict**):
   - **name**: ”Pulp Fiction”
   - **year**: 1994
   - **director**: ”Quentin Tarantino”
   - **genre:** ”Crime, Drama”
   - **duration:** 154

   Print the contents of this dictionary **without a loop** as in the example below.

   For this exercise, **check out Repetition statements materials, page 66**.

Example of the application running:

```
Pulp Fiction
1994
Quentin Tarantino
Crime, Drama
154
```

Filename of the exercise = ***exercise5_6.py***

Typical code amount : ***8-16 lines*** (empty lines/comments not included)

Advanced exercises!

Note: A good grade does not need all advanced exercises to be done. You can choose those advanced exercises that interest you the most!

7. Create an application that contains a list with the following product identifiers as text:
    - **”PHILIPS_Boiler_HD4646_2020_09_21_C_1”**
    - **”KENWOOD_KitchenMachine_KVL8300S_2015_09_22_C_3”**
    - **”BOSCH_Blender_MMB65G5M_2016_05_07_C_3”**
    - **”WHIRLPOOL_Microwave_MCP345WH_2019_01_15_C_1”**
    - **”ROSENLEW_Freezer_RPP2330_2017_01_29_C_2”**
    - **”ELECTROLUX_Fridge_ERF4114AOW_2017_11_07_C_2”**

    A single product identifier (or id) consists of the following formula:

    MANUFACTURER_PRODUCTNAME_MODEL_YEAR_MONTH_DAY_C_CATEGORY

    The categories of the products are numbers between 1-3, and they mean the following names:

    1 = Small electronics
    2 = Appliances
    3 = Blenders

    Loop through all product identifiers, and split all the needed information into their own variables (check out Repetition statements materials, page 95.)

    Print each product in the format:

    Brand: PHILIPS
    Name and model: Boiler (HD4646)
    Category: Small electronics
    Addition date: 21.9.2020

    Print also an empty line after each product description.

Tip: The names of the categories could be in a separate list, for example:

```
categories = ["Other", "Small electronics", "Appliances", "Blenders"]
```

After this, it's possible to get the name of the category by using the category number quite easily, for example:

```
# testing, if category is 1. this should come from the
# product identifier
c = 1
description = categories[c]
```

This structure works well even if there were lots of different categories!

Filename of the exercise = ***exercise5_7.py***

Typical code amount : ***20-40 lines*** (empty lines/comments not included)

8. Create an application that asks the user for a word. After this, ask the user for a shift number. Based on the word and the shift number, transform the word according to Caesar's cipher, and print the result.

   More details on Caesar's cipher:
   https://en.wikipedia.org/wiki/Caesar_cipher

   For example: if the original word is "***banana***", and the shift number is 3, then the ciphered version would be "**edqdqd**".

   Tip: There are many ways on how to create this, but these are some useful functions that might help:

   for example:

```
# this gives you the position of a certain letter in the
# alphabet (character set)
index = ord("a")

# this gives you a certain letter based on
# character position number
char = chr(99)
```

P.S. Caesar's cipher is extremely easy to break in real life, so it's not useful for security purposes, but it's a good programming exercise!

Filename of the exercise = ***exercise5_8.py***
Typical code amount : ***10-20 lines*** (empty lines/comments not included)

9. Python has a feature called ***list comprehension***, which is helpful when performing complex searches or operations towards lists or collections- Check out examples:
   - https://www.programiz.com/python-programming/list-comprehension
   - https://www.digitalocean.com/community/tutorials/understanding-list-comprehensions-in-python-3

In this extra exercise:

1. Create an application that contains a list, which contains at least 10 names of different persons (imaginary)

2. Use ***list comprehension***, and print only those names that follow **all these conditions at the same time**:
   - Names that do not contain the letter s
   - Names that do not contain the letter e
   - Names that are less than 8 characters long
   - **Extra challenge:** Names that contain a maximum of 2 vowels!
     - These Python 3-functions might be useful in this extra challenge:
       1. sum(), list(), map()

Filename of the exercise = ***exercise5_9.py***

Typical code amount : ***12-24 lines*** (empty lines/comments not included)