

## Exercise set 9 - Functions

**NOTE! In each exercise:**

**Code all your own functions into a file called: `functions.py`**, and upload it also to codePost in each exercise's submit box along with the actual exercise.

Remember to use your **`functions.py`** -file in each exercise by using the **`import`** -command!

For example, if your functions are in the file **`functions.py`**, and it contains a function called **`greetings()`** , then:

```
from functions import *  
greetings()
```

or

```
import functions  
functions.greetings()
```

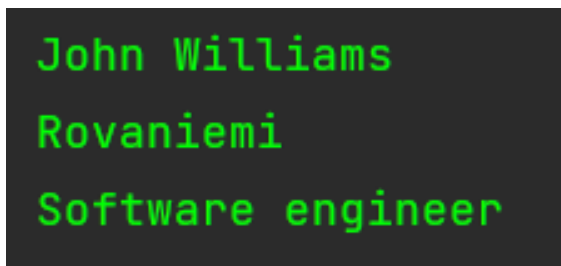
**Note:** When you submit exercises, it's okay if you have the functions of every exercise already there. Another neat thing about functions is that they do not do anything in your code, unless they are used/called somewhere in the application code! In this case: **`functions.py` is like a toolbox, which is used by different exercises whenever they need some function!**

1. Create a function called ***show\_personal\_info()*** that prints the following information of a person: ***name***, ***home city*** and ***profession***. You can use imaginary values in this exercise.

**Note:** ***show\_personal\_info()*** -function does not take any parameters and does not return anything. It only has to print three lines of text!

Finally, create an application that uses (calls) this function.

Example of the application running:



```
John Williams  
Rovaniemi  
Software engineer
```

Filename of the exercise = ***exercise9\_1.py***

Typical code amount : ***6-10 lines*** (including own function code, empty lines/comments not included)

2. Create a function: **count\_seconds(hours, minutes, seconds)** that takes three parameters: the hours, minutes and seconds.

The function **has to return** only one number as a result, which contains the total amount of seconds based on the given hours, minutes and seconds. Ask the needed values from the user, and print out the result.

**Don't print anything in the function itself!**

Example of the application running:

```
Give hours:
3
Give minutes:
47
Give seconds:
33
13653 seconds in total.
```

Filename of the exercise = *exercise9\_2.py*

Typical code amount : **7-16 lines** (including own function code, empty lines/comments not included)

#### Extra exercise:

Allow the user to input all information in one line and this format:  
**"2h 45min 33sec"**

**Tip:** This can be done by using string or list functions! (split or substring) You can return this version into the same codePost submit box without the basic version, and codePost will recognize it!



3. Create a function called **magazine\_serial\_check(serial)** that takes only one parameter as a text: **the ISSN-serial**. The function has to check if the ISSN-serial is in correct form or not. If the serial is in correct format, print "**Valid ISSN**". If the serial is not in correct format, print "**Incorrect ISSN**". You can print this text inside the function or return the value into a variable and print it elsewhere.

A valid ISSN-serial contains 8 numbers in total, with a dash in the middle.

For example, **0781-2078** is a valid ISSN-serial.

**1564XABCD** is an incorrect ISSN-serial, for example.

Examples of the application running:

```
Give an ISSN-serial:  
1234-8765  
Valid ISSN.
```

```
Give an ISSN-serial:  
1234T7UIY  
Incorrect ISSN.
```

Filename of the exercise = **exercise9\_3.py**

Typical code amount : **10-24 lines** (including own function code, empty lines/comments not included)

**Background:** The ISSN-serial (International Standard Serial Number) is used to identify such unique publications, that are continuously released by the same name (for example, magazines). ISSN-serial is globally unique.

**Tip:** String functions in Python are very useful in this exercise.

One approach could be: Check first if the middle character (index 4) is a dash. Then, remove the middle dash, and then check if the remaining serial is exactly 8 characters long. Then check if all remaining values are numbers. (check string manipulation materials for examples)

This code structure can be done in many ways, but the most straight-forward approach would probably to be **if-elif-elif-else** - structure!

**Small extra task:** Return a Boolean value in the function, which implies if the serial is valid or not (True/False). If using Booleans, don't print the result in the function, but in the application code instead.



4. Create a function called **show\_numbered\_list(title, data)** that takes two parameters: the **title text** and **a list of participant names** (one name = firstname + surname). The function has to print the title text first, and then print a numbered list of all participants given in the list.

**Note:** the code inside in the function is quite small, **and there will not be any if-statements in the function!**

**In the application, use/call this function three times:**

1. Print the list in original order by using the function  
(title: "**Original order:**")
2. Sort the list into alphabetical order in your code, and print the list by using the function.  
(title: "**Alphabetic order by first name:**")
3. Sort the list into alphabetical order based on the surname (check the examples next page), and print the list by using the function.  
(title: "**Alphabetic order by last name:**")

**Note:** **there will not be any if-statements in the function!**

Ask the user to provide the list of participants in one line, all names separated by a comma! (see the example next page):

### Example of the application running:

```
Write all participants, separated by a comma:  
James Smith, Maria Garcia, George Wilson, Sarah Miller, Jane Davis  
  
Original order:  
1. James Smith  
2. Maria Garcia  
3. George Wilson  
4. Sarah Miller  
5. Jane Davis  
  
Alphabetic order by first name:  
1. George Wilson  
2. James Smith  
3. Jane Davis  
4. Maria Garcia  
5. Sarah Miller  
  
Alphabetic order by last name:  
1. Davis Jane  
2. Garcia Maria  
3. Miller Sarah  
4. Smith James  
5. Wilson George
```

Filename of the exercise = *exercise9\_4.py*

Typical code amount : **12-18 lines** (including own function code, empty lines/comments not included)

**Note: There will be no if-statements inside the show\_numbered\_list() - function!**

### Tips:

If the user inputs the following text, for example: ***John Doe, Tina Tester and Ellie Example***, we can convert the text into a list like this:

```
# ask all the names as one string from user
people_string = input("Write all participants, separated by a comma:\n")

# convert the text into a list
people = people_string.split(",")

# remove extra space-characters from each name (from the beginning and end)
people = [p.strip() for p in people]

# the people-variable now is a list, that contains full names of each participant
```

Before we can sort the list based on the surname, we have to first convert the data into the format of "Surname Firstname". It can be done like this:

```
# let's convert the people in the list so, that last name
# comes before the first name
# this is called "list comprehension" - a feature in Python
# The logic: each name will be split in its turn, and converted to a
# list based on the space character. Because of this, each full name will
# be a list of two elements: first name and the last name.

# After this, the order of the elements is changed by using the reverse-
# function. Finally, the two elements will be built back into a string by
# using the join() -function. The end result is a list that contains the
# full names of all people, but in the format of Last Name First Name

people = [" ".join(reversed(p.split(" "))) for p in people]
```

**Note! Do all the printing inside the function! For example, if people-list now contains the names in the desired order, you just call the function:**

```
show_numbered_list("The title you want to use", people)
```

After, modify the **people**-list as you need (for example, by using `sort()`) and just call the same function again:

```
show_numbered_list("Some other title this time!", people)
```

In other words, the point of this exercise is to demonstrate how you can use the same function many times in different situations as the data gets different! **Remember, there will be no if-statements inside the function!**



5. Create a calculator application for three different volume calculations. Support these calculations:

**1. volume of a box**

-> create a new function: ***box\_volume(width, height, depth)***

- formula = width \* height \* depth
- return the result of the function, rounded to two decimals. Don't print anything in the function. **Don't print anything in the function.**

**2. volume of a ball**

-> create a new function: ***ball\_volume(radius)***

- formula =  $(4 * \pi * \text{radius}^3) / 3$
- return the result of the function, rounded to two decimals. Don't print anything in the function. **Don't print anything in the function.**

**3. volume of a pipe**

-> create a new function: ***pipe\_volume(radius, length)***

- formula =  $\pi * \text{radius}^2 * \text{length}$
- return the result of the function, rounded to two decimals. Don't print anything in the function. **Don't print anything in the function.**

**Note:** create a separate function for each calculation! **Don't print anything in any of the functions**, but return the calculated number instead (**return**), and do all the printing in the actual program code!

In the application, ask the user to give a number that determines, which calculation the user wants to perform (alternatives: 0 - 3)

**1** = box, **2** = ball and **3** = pipe, **0** = stop the application

After the user has selected the calculation, ask the user for the needed inputs, call the needed function, and print the result.

**Let the user use the application until the user selects 0.**



### Examples of the application running:

```
Select the operation (1-3), 0 stops the application:
1
Give box width:
3
Give box height:
5
Give box depth:
7
Box volume: 105 m3
```

```
Select the operation (1-3), 0 stops the application:
2
Give ball radius:
7
Ball volume: 1436.76 m3
```

```
Select the operation (1-3), 0 stops the application:
3
Give pipe radius:
6
Give pipe length:
9
Pipe volume: 1017.88 m3
```

```
Select the operation (1-3), 0 stops the application:
0
Thank you for using our application!
```

Filename of the exercise = *exercise9\_5.py*

Typical code amount : **25-50 lines** (including own function code, empty lines/comments not included)

## Extra tasks!

*Note: A good grade does not necessarily require all extra exercises! It's more important there's an interesting alternative for everyone!*

### 6. Lottery numbers

Create a function that generates a random series of lottery numbers. In this case, a series of lottery numbers contains 7 unique numbers between 1 and 40.

**Tip:** you can use a collection (list) to keep track of numbers that have been selected already. **Note:** there should not be duplicate numbers in a single series! You can check if a number exists in the list by using a conditional statement (if number in lotto\_numbers: => generate a new random number)

Finally, print the generated lotto number for the user.



Filename of the exercise = *exercise9\_6.py*

## 7. Previous exercises by using functions

Select one of the following earlier exercises, and create a new version of it by placing the application logic into own separate functions. You can also do multiple exercises if you wish. You'll get extra points up to 3 different earlier exercises done this way.

### Alternatives:

- Exercise 3 – 3
- Exercise 4 – 2
- Exercise 6 – 2
- Exercise 6 – 3
- Exercise 7 – 5
- Advanced exercise 3 – 7 or 3 – 8
- Advanced exercise 4 – 7
- Advanced exercise 5 – 8
- Advanced exercise 6 – 8

**Filename of the exercise:** Return with the same name into the original submit box of the exercise.

## 8. Currency exchanger

Create a function called "**convert\_money**" that takes three parameters: the amount of money, the original currency and the target currency. Acceptable currencies are euro (€), dollar (\$) and pound (£).

The function should convert the given sum into the new currency based on current exchange rates.

You can use these exchange rates in this exercise:

- 1 € (euro) = 1.2 \$ (dollar)
- 1 € (euro) = 0.9 £ (pound)

In the application, ask the user for the amount of money and the original currency. Then ask the user for the currency to which the money should be converted. Use your function to get the result, and print it out on the screen.



**Extra exercise: Get the current exchange rates from an internet API!**

Filename of the exercise = *exercise9\_8.py*

## 9. Recursive functions

A recursive function is a function that can also call itself, if needed. Remember, in most cases you don't have to use recursion, especially in simple applications. For example, the Fibonacci-sequence exercise we did earlier can be done only with loops as well.

For example, the Fibonacci sequence exercise by using recursion:

[https://www.python-course.eu/python3\\_recursive\\_functions.php](https://www.python-course.eu/python3_recursive_functions.php)

In this exercise, create a recursive function in Python that prints out the contents of a given folder, as well as its subfolders and their contents as well.

Every time you print a folder's name, print also three dots after that. In addition, for each level of depth in a folder, add another dash in the name. For example:

*Pictures ...*

*- Flower.jpg*

*- Pear.jpg*

*- Vacation ...*

*- - Prague1.jpg*

*- - Prague2.jpg*

*- TODO.txt*

You can use the following module in this exercise:

```
import os

for x in os.listdir("C:\\folder1\\another folder"):
    print(x)
```

**Note:** There are existing modules for this problem in Python, but the idea of this exercise is to create this code yourself.

Filename of the exercise = *exercise9\_9.py*

**Useful Google search term:**

"python 3 traverse directory contents with recursion"

## 10. Advanced function techniques

- **Create a lambda that returns a Boolean depending if the given integer is even**
  - **Tip:** You can store a lambda in a variable!
- **After this, create a collection that has a random number of both odd and even numbers**
- **Use filter() to filter out the collection by using the lambda you created earlier, and create a new list**
  - filter() returns a so-called "filter-object" which can be then converted to a list by using list()
- **Use map()- function to process the collection by using the lambda you created earlier, and create a new list**
  - map() returns a so-called "map object", which can be then converted to a list by using list()

In the end, you should have two new lists, one of which contains the odd numbers, and the other contains a list of True and False –values (even or not). Print these lists on the screen.

### Additional info:

filter() and map() might seem quite similar to each other, but the operation they do is quite opposite. filter() filters out undesired values based on the lambda from the list. Map will create a new list instead that contains the result of the lambda for each list element. Which technique is more useful really depends on the situation; whether you want to modify the original data (filter) or whether you want to collect information of each element (map).

**Note:** Remember also list comprehension if you need to filter collections!

Filename of the exercise = *exercise9\_10.py*



## 11. Sorting complex data (custom sort)

Create the following list in your application (a list of dictionaries):

- 0:
  - city : "Paris"
  - landmark: "Louvre Museum"
- 1:
  - city : "London"
  - landmark: "National Gallery"
- 2:
  - city : "Paris"
  - landmark: "Eiffel Tower"
- 3:
  - city : "London"
  - landmark: "Madame Tussauds"
- 4:
  - city : "London"
  - landmark: "British Museum"
- 5:
  - city : "Paris"
  - landmark: "Notre Dame"

Create a Python application that contains a function called **city\_landmark\_sort(data)**, which sorts the list so that first they are sorted by the city, and then by the name of the landmark. This can be done in many ways!

Filename of the exercise = *exercise9\_11.py*

**The end result should be like this in this case:**

*London: British Museum*

*London: Madame Tussauds*

*London: National Gallery*

*Paris: Eiffel Tower*

*Paris: Louvre Museum*

*Paris: Notre Dame*



**Tip:** Useful Google search term: "python 3 sort list of dictionaries by multiple keys"