

Osnova

# Lesson Overview

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [LESSON OVERVIEW](#)

Welcome to lesson 4, we missed you!

In last lesson you have learned how to use:

- dictionaries
- & sets.

And now, we will focus on **while loops**.

This concept is very useful, because automatization saves your time. You can automatize some routine action, e.g. repetitive tasks. A tiny part of code can ensure that you don't have to ask Python to print some information for 10 times, but just once.

At the end of this lesson, a voluntary project for self-study is being introduced. We highly recommend you to work on it. You can test, what you have learned so far. Go for it :)

Osnova

00:58 |  


## REVIEW EXERCISES

### Dictionary

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [REVIEW EXERCISES](#) / [DICTIONARY](#)

Goal of this task is to put different dictionaries into our main dictionary.

100% z Lekce 4

```
''' Here is a pair dictionary: database = {'id123': {}, 'id124': {}, 'id125':  
''' Osnova  
''' }
```

And here are the others:

```
FirstDict = {'name': 'Thomas', 'age': 45, 'Country': 'Czechia', 'City':  
'Brno'}  
SecondDict = {'name': 'Daniel', 'age': 34, 'Country': 'Czechia', 'City':  
'Prague'}  
ThirdDict = {'name': 'Eva', 'age': 43, 'Country': 'Czechia', 'City':  
'Olomouc'}
```

Example of running script:

```
{'id123': {'name': 'Thomas', 'age': 45, 'Country': 'Czechia', 'City':  
'Brno'}, 'id124': {'name': 'Daniel', 'age': 34, 'Country': 'Czechia',  
'City': 'Prague'}, 'id125': {'name': 'Eva', 'age': 43, 'Country':  
'Czechia', 'City': 'Olomouc'}}
```

## Online Python Editor

1 |

Osnova  
spustit kod

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

## Sets

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [REVIEW EXERCISES](#) / [SETS](#)

At the beginning we have dictionary with tram stations. Our goal is to identify which stations are included in each list.

```
1 TramStations = {  
2 'No.1' : ['Reckovice', 'Semilasso', 'Husitska', 'Jungmannova',  
    'Kartouzská', 'Sumavská', 'Hrnicrska', 'Pionyrská', 'Antoninska',  
    'Moravske nam.', 'Malinovske nam', 'Hlavni nadr.', 'Nove sady',  
    'Hybesova', 'Vaclavska', 'Mendlovo nam.', 'Vystaviste main',  
    'Vystaviste G2', 'Lipova', 'Pisarky'],  
3 'No.2' : ['Zidenice', 'Kuldova', 'Vojenska nemocnice', 'Tkalcovska',  
    'Kornerova', 'Malinovske nam.', 'Hlavni nadr.', 'Nove Sady',  
    'Hybesova', 'Vaclavska', 'Mendlovo nam.', 'Porici', 'Nemocnice UM',  
    'Celni', 'Hluboka', 'Ustredni hrbitov'],  
    ...  
}
```

100% z Lekce 4

```
Osnova 'venskeho nam.', 'Obilni trh', 'Uvoz']  
5 }
```

Example running script:

```
{'Hlavni nadr.'}
```

## Online Python Editor

1 |

spustit kód

## Code Solution

100% z Lekce 4

# ONSITE PROJECT

## Our Goal

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [OUR GOAL](#)

Today we will work on a virtual shopping cart application. We want our shopping cart to be able to perform the following **actions**:

1. Add new items to it

100% z Lekce 4

3. Calculate the total price

Osnova

4. Run the program until the user decides to terminate it

Optionally:

5. get basic statistics about counts of individual items in the basket

6. retrieve prices by name

7. compare the contents of our cart to a list items in offer

8. depict the cart contents in a neat way

9. remove the cart items



## Before while loop

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [BEFORE WHILE LOOP](#)

Create a program that will:

100% z Lekce 4

• create a list representing the shopping cart,  
Osnova

- calculate the total price,
- print the cart contents and the total price.

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



## Repetition

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [REPETITION](#)

What if we wanted to change the number of items to be entered? We would have to go back and change the code, adding new lines.

What if we wanted our program to collect 100 price tags? We would have to add 100 lines.



The solution is to use **loops**. Loop allows us to tell Python, which set of instructions to we want it to execute **repeatedly**.

Good candidates for change are these lines of code:

100% z Lekce 4



```
2 item1 = float(input('Enter the price: '))
3 item2 = float(input('Enter the price: '))
4
5 cart.append(item1)
6 cart.append(item2)
7 cart.append(item3)
```

What we are actually doing is repeating the following two commands:

```
1 item = float(input('Enter the price: '))
2 cart.append(item)
```

## Action 1 - Adding items

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [ACTION 1 - ADDING ITEMS](#)

So in general, if want to say **while we have not collected 3 items, keep collecting them** the structure would look like this:

```
1 while we have not collected 3 items:
2     keep collecting them
```

OR

```
1 while there are less than 3 items in a cart:
2     keep collecting them
```

## Action 1

Let's dive into our action list, starting with: 1. Add new items to the cart

Click to see our solution



3. Osnova 4. Solution

## Code Task

How do we:

- get number of items in a cart
- and express less than 3 items in a cart?

1 |

spustit kód

... and ... below if you want to see, how we wrote the code.

Osnova

Click to see our solution

## Action 2 & 3 - List content & Total price

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [ACTION 2 & 3 - LIST CONTENT & TOTAL PRICE](#)

Ok, so far we have the following code:

```
1 cart = []
2 while len(cart) < 3:
3     item = float(input('Enter the price: '))
4     cart.append(item)
```

We can now get to the 2nd and 3rd action point:

2. List the cart's content
3. Calculate the total price

The first one shouldn't be such a problem ;). Also, we'd be able to calculate the price f.e. by using indexing:

```
1 total_price = cart[0] + cart[1] + cart[2]
```

However, this is not a lesson on indexing! **Use a while loop** to sum all the 3 prices :)

```
1 |
```

Osnova

[spustit kód](#)

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

[Click to see our solution](#)



## Action 4 - Infinite asking

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [ACTION 4 - INFINITE ASKING](#)

So we have already completed 3/4 tasks for this lesson, good job!:

1. **Add new items to it**
2. **List its content**

100% z Lekce 4

4.  **Osnova**  until the user decides to terminate it

So, our last task is to run the program until the user decides to terminate it. For this purpose we can use infinite loop. However, there is **bad and good** infinite loop.

## Bad infinite loop

When using while loops, a infinite loop can occur. This can be bad, when **we do not have it under control**:

```
1 total_price = 0
2 i = 0
3 while i < len(cart):
4     total_price = total_price + cart[i]
```

We need to be able to **change the variable**, that is being assessed in the loop's header.

```
1 total_price = 0
2 i = 0
3 while i < len(cart):
4     total_price = total_price + cart[i]
5
6     i = i + 1
```

## Good infinite loop

Good infinite loop grant's the user possibility to **terminate the program by allowing for input**. What does that mean? A use case in our program is when we want to allow our users to add as many prices as they want and enter 'q' to stop the process of price collection.

To create an **infinite loop** we need a a condition that will always evaluate **True** , unless we change the tested variable value inside the loop body.

## Code Solution Summary

Use dropdown feature below if you want to see, how we wrote the code.

[CLICK TO SEE OUR SOLUTION](#)

Osnova

## Action 4 - Infinite listing

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [ACTION 4 - INFINITE LISTING](#)

Additionally, we could keep listing the contents of the cart (you can use the cart below), until the user tells the program to stop, by entering letter 'q'. Once we are at the end of the cart, we want to return to its beginning and show the first item and so forth.

Before we incorporate it into our program, let's try this first with the following cart:

```
1 cart = [1.02, 3.45, 6.82]
```

```
1 |
```

[spustit kód](#)

100% z Lekce 4

Osnova

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



## Code Summary

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [CODE SUMMARY](#)

Let's now try to incorporate the infinite listing into our program:

```
1 i = 0
2 repeat = True
3 # Infinite listing
4 while repeat:
5
6     index = i % 3
7     print(cart[index])
8
9     answer = input('Press enter to continue or "q" to quit: ')
10
11     if answer == 'q':
12         repeat = False
13     else:
14         i = i + 1
```

So the program should:

2. 'Osnova' until the key 'q' is pressed

Osnova

3. using while loop to **calculate the total price**

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



## Food Database

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [FOOD DATABASE](#)

Till now, we have been collecting prices into our shopping cart. Now we will turn to maintaining the food database.

Our database will get more and more sophisticated. But first, we will keep it very simple - in the form of **food\_name : food\_price** pairs

For example:

**'mustard': 10**

Python allows us to keep data in this form thanks to the **dictionary** data type. Why dictionary? In real world language dictionaries, we are able to search for a word. Once we find it, we get the associated meaning.

And this is how it works. We take so called key - the first member of the pair - mustard - and we get its price:

Dictionaries are collections of **key : value** pairs enclosed in **curly braces {}**:



- The first member of the pair following the colon is the **key** Osnova

- The second member of the pair following the colon is the **value** (10)

## Keys have to be unique

```
{'mustard': 10, 'ketchup':25, 'salami':34}
```

# Getting the values

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [GETTING THE VALUES](#)

Similarly to lists, we get the values from dictionaries using square brackets `[]`, but inside them we put the key name, which value we want to retrieve:

```
food = {'mustard': 10, 'ketchup':25, 'salami':34}
```

```
salami_price = food['salami']
```

## What if the key is not present?

```
milk_price = food['milk']
```

## Use `dict.get()` method

```
milk_price = food.get('milk',0)
```

# Adding new keys

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [ADDING NEW KEYS](#)

```
food = {'mustard': 10, 'ketchup': 25, 'salami': 34}
Osnova = 14.90
```

Or we can check, whether milk is present in the dictionary and if not, then add it using **get()** method:

```
if not food.get('milk'):
    food['milk'] = 14.90
```

Or even simpler:

```
food['milk'] = food.get('milk', 14.90)
```

Or if we had the name and the price in another dictionary, we could use **update()** method:

```
food.update({'milk': 14.90})
```

## Changing the values

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [CHANGING THE VALUES](#)

To change the value associated with a key, we do the same thing as if we added a new item:

```
food = {'mustard': 10, 'ketchup': 25, 'salami': 34, 'milk': 14.90}
food['milk'] = 15.90
```

Or we can work with **update()**:

```
food.update({'milk': 15.9})
```

## Removing keys

... We would like to remove milk from our database as we no longer sell it. We do it using `pop()`.  
Osnova: In the parentheses, we put the key, which we want to remove:

```
food = {'mustard': 10, 'ketchup': 25, 'salami': 34, 'milk': 15.9}  
food.pop('milk')
```

## More sophisticated database

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [ONSITE PROJECT](#) / [MORE SOPHISTICATED DATABASE](#)

We would like to track more information about one type of food. For example, these are the categories, which we want to track:

- name
- producer
- product
- price
- unit
- category
- quantity

For example:

```
{'name': 'mustard', 'producer': 'Senf', 'product': 'Kremzska', 'price': 10,  
'unit': 100, 'category': 'condiment', 'quantity': 50}
```

So how would our database look like?

We can keep all the records in a list:

```
food = [  
{ 'name': 'mustard', 'producer': 'Senf', 'product': 'Kremzska', 'price': 10,
```



# WHILE LOOPS

## Loops introduction

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [WHILE LOOPS](#) / [LOOPS INTRODUCTION](#)

Computers are good friends when speaking about doing repetitive and boring tasks on our behalf. Repetitions in programming are called loops. For example, if we want to find and print out all the numbers between 3 and 1358979677 divisible by 3, it is better to leave this task to our PC. Actually a computer is designed to work like that, to run in cycles - its processor runs in cycles. The number of cycles per second is called Hertz. So these guys are nowadays doing billions of

One of the main performance of repeating tasks is called looping and Python recognizes 2 kinds of loops.

Osnova

- **while loop**
- **for loop**

Loops are **compound statements** that means they consist of header and suite similarly to conditional statements.

### Template for the while loop:

```
1 while test:
2     your code
```

### Template for the for loop:

```
1 for item in iterable:
2     your code
```

Header contains the reserved keyword - **for** or **while**. Suite contains at least one statement. Statements that compose the suite (body) of the loop are repeatedly executed until a defined end state is reached.

## The principle of While

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [WHILE LOOPS](#) / [THE PRINCIPLE OF WHILE](#)

While loop is a more general loop than for loop. Meanwhile Python has to perform some magic behind the scenes of for loop, while loop's principle is very straightforward.

With **while** loop, the code is repeatedly executed inside the while body as long as the test in the loop's header evaluates **True** :

```
1 while test:
2     statements
```



1. Once the while statement header is encountered by program execution, the test in the header is evaluated as `bool(test)` by Python..
2. If the result of the boolean test is `True`, then the statements in the loop's suite are executed. Otherwise the suite is skipped.
3. Program execution returns back to the `while` header and again evaluates the test it contains.
4. In the moment, the test evaluates to `False`, the loop terminates and the program execution continues with lines after the while block.

As an **example**, try to run the following code on your computer:

```
1 num = 5
2
3 while num > 0:
4     print('My number is ' + str(num))
5     num = num - 1
6 print('The loop has terminated')
```

spustit kód

# Infinite loop

PYTHON ACADEMY / 4. WHILE LOOPS / WHILE LOOPS / INFINITE LOOP

Using while loop brings with itself the danger of getting stuck in an infinite loop. Infinite loop never terminates and is caused by poor code design - when the test in the header can never evaluate to **False**.

Example of such situation is, when we count from 1 up and want to stop loops execution once the counter is equal to 0. The value in the variable **num** will never reach the limit 0 as it moves away from it.

```
1 num = 1
2 while num > 0:
3     print(num)
4     num += 1
```

If you will run the code above, the program execution will enter infinite loop! In order to stop such a program we need to use keyboard shortcut **Ctrl+C**.

## Important

One of the statements executed inside the loop's suite has to **manipulate variable** that forms part of the header test expression, otherwise:

- If this variable was not changed, the loop would run infinitely long.
- If value of this variable does not approach the limit, the loop runs infinitely long

## Use case 1



That's very useful, **when we want to count number of occurrences** of items. In the below code, we need to know something about the **while** loop.

We could do it this way:

```
1 colors = ['green', 'blue', 'black', 'red', 'red', 'yellow', 'blue',  
  'grey', 'black', 'red', 'green']  
2 color_counts = {}  
3  
4 while colors:  
5     color = colors.pop()  
6  
7     if color not in color_counts:  
8         color_counts[color]=0  
9  
10    color_counts[color] = color_counts[color] + 1
```

or this way

```
1 colors = ['green', 'blue', 'black', 'red', 'red', 'yellow', 'blue',  
  'grey', 'black', 'red', 'green']  
2 color_counts = {}  
3  
4 while colors:  
5     color = colors.pop()  
6     color_counts[color] = color_counts.get(color,0) + 1
```

The **color\_counts** variable should be at the end refer to a dictionary:

```
>>> color_counts  
{'grey': 1, 'blue': 2, 'black': 2, 'red': 3, 'green': 2, 'yellow': 1}
```

## Use case 2

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [WHILE LOOPS](#) / [USE CASE 2](#)

100% z Lekce 4

while loop is used in cases **when we do not know in advance, how many repetitions** will be executed. For example, when the program requires user's input which is unpredictable.

The program below requests a number and then counts down 1 cycle per second:

```
1 from time import sleep
2
3 num_seconds = int(input('How many seconds to you need?: '))
4
5 while num_seconds:
6     sleep(1)
7     print(num_seconds)
8     num_seconds = num_seconds - 1
```

Output:

```
5
4
3
2
1
```

## Iteration techniques

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [WHILE LOOPS](#) / [ITERATION TECHNIQUES](#)

Let's go over 3 iteration techniques to further demonstrate the use of this loop.

### Looping using a number

Loop will go on until the variable value is not equal to 0:

```
1 number = 10
```

```
4     number- 1
5     Osnova  'New Year')
```

If number variable acquires value 0, the loop body is not executed and the loop code block is skipped. Program execution then continues with the lines following the loop's code block

## Looping over iterables

Cutting of pieces of a string (or other iterable):

```
1 my_str = 'while loops are more genEral'
2 while my_str:
3     if my_str[0].isupper():
4         print('I have found capital:',my_str[0])
5     my_str = my_str[1:]
```

The program above checks whether there are any capital letters in the string my\_str. Similarly to number being equal to 0, if iterable is empty (all the chars cut off), then the loop exits.

[Check it out in Python Tutor.](#)

## Using index to retrieve an item

**Less elegant and less Pythonic** than the previous two examples:

```
1 my_str = 'while loops are more genEral'
2 index = 0
3 while index < len(my_str):
4     if my_str[index].isupper():
5         print('I have found capital:',my_str[index])
6     index += 1
```

Osnova

# QUIZ

## While Loop

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [QUIZ](#) / [WHILE LOOP](#)

1/9

What is the purpose of the while loop?

- ☐ A. To repeat designated set of commands until the tested condition is not met
- ☐ B. To pause the code execution
- ☐ C. To repeat designated set of commands until the tested condition is met

100% z Lekce 4

# HOME EXERCISES

## Student Names

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [STUDENT NAMES](#)

We have a class of students. All the student names are stored in the list `students`.

```
1 students = ['Adam, Baker', 'Chelsea, Archer',  
2           'Marcus, Archer', 'Oliver, Cook',  
3           'Alex, Dyer', 'Sandra, Turner',  
4           'Ann, Fisher', 'Glenn, Hawkins',
```

```
7      Osnova      'Clara, Woodman', 'Abraham, Mason',  
8                  'Marcus, Sawyer', 'Alex, Glover',  
9                  'Glenn, Cook', 'Clara, Fisher',  
10                 'Alfred, Dyer', 'Curt, Head',  
11                 'Oliver, Slater', 'Alex, Mason',  
12                 'Tyler, Fisher', 'Ann, Parker',  
13                 'Samuel, Hawkins', 'Ann, Woodman',  
14                 'Sandra, Slater', 'Curt, Dyer']
```

Our task is to extract an overview of what unique names and surnames do we have in the class.

```
~/PythonBeginner/Lesson2 $ python student_names.py
```

```
Extracting ...
```

```
Unique names:
```

```
{'Ann', 'Curt', 'Clara', 'Abraham', 'Chelsea', 'Oliver', 'Glenn',  
'Samuel', 'Alfred', 'Marcus', 'Alex', 'Adam', 'Tyler', 'Sandra'}
```

```
Unique surnames:
```

```
{'Woodman', 'Head', 'Dyer', 'Smith', 'Cook', 'Hunt', 'Slater', 'Baker',  
'Parker', 'Turner', 'Fisher', 'Sawyer', 'Mason', 'Archer', 'Glover',  
'Hawkins'}
```

## Online Python Editor

1 |

Osnova

[spustit kód](#)

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

[Click to see our solution](#)

## Difference - Odd vs.Even

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [DIFFERENCE - ODD VS.EVEN](#)

Write a Python script that will sum all the even numbers and odd numbers separately. At the end the program should print to terminal **the absolute value of the difference** between the two sums of odd and even numbers.

Example of how the script should work:

1. We have a list of numbers: `[1,2,3,4,5,6,7,8]`
2. Now we will sum all the even numbers and the result store in the variable `even = 2 + 4 + 6 + 8`
3. Then we will sum all the odd numbers and the result store in the variable `odd = 1 + 3 + 5 + 7`

100% z Lekce 4





## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



## Echo

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [ECHO](#)

Write a Python program that will create "echo sentences". Each word in the sentence we will feed in, should be repeated n number of times. The number of repetitions and the sentence to be manipulated are inputs provided by the user.

Example:

If the supplied number of repetitions is 3 and the sentence: **'I do not want to work today'**.

Output:

**'I I I do do do not not not want want want to to to work work work today today today'**

The resulting sentence cannot begin with space, unless the input sentence contained it.

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python echo.py
```

```
1 # Print out the words in the list so that you only print the words you want to print today
2 # Hint: Use a while loop to loop over the list and print the words you want to print today
3 # Osnova
4 # not not want want want to to to work work work today
```

## Online Python Editor

1 |

spustit kód

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



100% z Lekce 4

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [LONGEST WORD](#)

Write a program that will take a list of words as input and will print to the terminal the longest word and its length in one tuple.

Please use the following list:

```
1 words = ['Python', 'is', 'a', 'widely', 'used',  
2         'high-level', 'programming', 'language',  
3         'for', 'general-purpose', 'programming',  
4         'created', 'by', 'Guido', 'van', 'Rossum',  
5         'and', 'first', 'released', 'in', '1991.']
```

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python longest_word.py  
('general-purpose', 15)
```

## Online Python Editor

1 |

Osnova

spustit kód

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution



## Sum powers

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [SUM POWERS](#)

Write a Python script that will ask the user to enter a number from which it will compute the result. The result should be the sum of numbers less than or equal to the input number, each raised to power of its value. Then the script should print the result to the terminal.

For example:

- if the user enters number 5, the program should compute the sum as:  $1**1 + 2**2 + 3**3 + 4**4 + 5**5$ .
- if the user enters 6, then it should be:  $1**1 + 2**2 + 3**3 + 4**4 + 5**5 + 6**6$
- ...and so on.

Example of running the script:

100% z Lekce 4

# Online Python Editor

1 |

[spustit kód](#)

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

[Click to see our solution](#)

100% z Lekce 4

## Osnova 'a dict

[PYTHON ACADEMY](#) / [4. WHILE LOOPS](#) / [HOME EXERCISES](#) / [LOOPING OVER A DICT](#)

We have a dictionary:

```
1 film = {'name': 'Forrest Gump',
2         'made': 1994,
3         'director': 'Robert Zemeckis',
4         'soundtrack': 'Multiple',
5         'starring': 'Tom Hanks',
6         'fun_fact': '''The house used in Forrest Gump is
7                       the same house used in The Patriot
8                       (2000). Some paneling was changed
9                       for interior shots in the latter
10                      film.'''}
```

Create a script that will print each key - value pair to the terminal in format: **Key: <value> | Value: <value>**"

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python list_items.py
Key: starring | Value: Tom Hanks
Key: director | Value: Robert Zemeckis
Key: made | Value: 1994
Key: name | Value: Forrest Gump
Key: soundtrack | Value: Multiple
Key: fun_fact | Value: The house used in Forrest Gump is the same house
used in The Patriot (2000). Some paneling was changed for interior shots
in the latter film.
```

## Online Python Editor

1 |

## Osnova

[spustit kód](#)

## Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

[Click to see our solution](#)[DALŠÍ LEKCE](#)