# Exercise: Functions Advanced

Problems for exercise and homework for the [Python Advanced Course @SoftUni](https://softuni.bg/courses/python-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1839>

## Negative vs Positive

You will receive a **list of numbers**. **Separate** the negative numbers from the positive. Find the **total sum of the negatives and positives**, **replace** the negative number with its **absolute value** and **print the following:**

If the **absolute negative number** is bigger than the **positive number**:  
 **"The negatives are stronger than the positives"**

If the **positive number** is bigger than the **absolute negative number**:  
 **"The positives are stronger than the negatives"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 -3 -4 65 -98 12 57 -84 | -189  137  The negatives are stronger than the positives |

## Odd or Even

You will receive a **command** and a **list of numbers**:

If the command is **"Odd"**: Print the **sum of the Odd** numbers multiplied by the **length** of the initial list.

If the command is **"Even"**: Print the **sum of the Even** numbers multiplied by the **length** of the initial list.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Odd  1 3 5 34 7 9 12 11 13 10 | 490 |

## Arguments Length

Create a function called args\_length that returns the **number of arguments**. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(args\_length(1, 32, 5)) | 3 |
| print(args\_length("john", "peter")) | 2 |
| print(args\_length([1, 2, 3])) | 1 |

## Even or Odd

Create a function called even\_odd that can receive different **amount of numbers** and a **command** at the end. The command can be **"**even**"** or **"**odd**"**. **Filter** the numbers depending on the command and **return** them in a **list**. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(even\_odd(1, 2, 3, 4, 5, 6, "even")) | [2, 4, 6] |
| print(even\_odd(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "odd")) | [1, 3, 5, 7, 9] |

## Function Executor

Create a function called func\_executor that can receive different number of **tuples**, each of which will have exactly **2 elements**: first will be a **function** and the second will be a **tuple of the arguments** that need to be passed to that function. Create a **list** which will contain all the **results** of the **executed functions** with its corresponding **arguments**. For more clarification, see the examples below. Submit only your function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| def sum\_numbers(num1, num2):  return num1 + num2  def multiply\_numbers(num1, num2):  return num1 \* num2  print(func\_executor((sum\_numbers, (1, 2)), (multiply\_numbers, (2, 4)))) | [3, 8] |

## Keyword Arguments Length

Create a function called kwargs\_length which can receive different amount of **keyword arguments** and **returns** their **length**. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| dictionary = {'name': 'Peter', 'age': 25}  print(kwargs\_length(\*\*dictionary)) | 2 |

## Age Assignment

Create a function called age\_assignment that receives different number of **names** and then different amount of **key-value** pairs. The **key** will be a **single letter** (first letter of a name), and the **value** a **number** (age). For each **name**, find its **first letter** in the **key-value** pairs and **assign** the **age to the persons name**. At the end **return a dictionary** with all the **names and ages** as shown in the example. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(age\_assignment("Peter", "George", G=26, P=19)) | {'Peter': 19, 'George': 26} |
| print(age\_assignment("Amy", "Bill", "Willy", W=36, A=22, B=61)) | {'Amy': 22, 'Bill': 61, 'Willy': 36} |

## Recursion Palindrome

Write a **recursive** function called palindrome which will receive a **word** and an **index** (**always 0**). Implement the function, so it returns **"{word} is a palindrome"** if the word is a palindrome and **"{word} is not a palindrome"** if the word is not a palindrome using **recursion**. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(palindrome("abcba", 0)) | abcba is a palindrome |
| print(palindrome("peter", 0)) | peter is not a palindrome |

## Recursive Power

Create a **recursive** function called recursive\_power which should receive a **number** and a **power**. Using **recursion** **return** the result of **number \*\* power**. Submit only the function in the judge system.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(recursive\_power(2, 10)) | 1024 |
| print(recursive\_power(10, 100)) | 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |