Exercise: Functions Advanced

Problems for exercise and homework for the Python Advanced Course @SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/1839

1. 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

2. 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]`

3. Function Executor

Create a function called **func_executor** that can receive different amount 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
| ```python
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] |

Follow us:

4. 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

5. Age Assignment

Create a function called **age_assignment** that receives different amount 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}`

6. 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`

7. 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))`	10000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000

Follow us: