Practical 5

Functions

1. Create a function that takes 3 integers as arguments and returns their average value.

2. Create a function that takes a list of integers as an argument and returns the number of even values in the given list.

3. Create a function that takes **password** of type String as an argument and check if the given **password** corresponds to the password rules, returning True or False, accordingly:

   Password rules:
   The length of the password should be at least 10
   The password should contain at least 2 numbers (try to find how to check this on your own)

4. Create a function that takes a required argument **name** of type String and an optional argument **greeting** with a default value "Welcome to our company!". The function prints "**name**, **greeting**", using the values of the arguments **name** and **greeting**.

5. Create a function that takes one required argument **name** of type String and an undefined number of optional non-keyword arguments. The function should calculate the average value of the optional arguments (if at least one optional argument is given) and print the following: if at least one optional argument is given print "**X**, your average grade is: **Y**", where **X** is the value of the argument **name** and **Y** is the average value of the optional arguments. Otherwise, print "No grades available for **X**", where **X** is the value of the argument **name.**

6. Create a function that takes one required argument **user** of type String and an undefined number of optional non-keyword arguments. If the value of the argument **user** is "admin", the function should print all the optional arguments in the following format:

   "Argument1 name": "argument1 value"
   "Argument2 name": "argument2 value"
   "Argument3 name": "argument3 value"
   etc.
   Otherwise, if the value of the argument **user** is not "admin", the function should print "access denied to the user X", where X is the value of the argument **user.**

Modules

7. Create the python file **calc.py** that contains the following 2 functions:
   calculate_cube(x) - which takes an integer **x** as an argument and returns the cube of the given number x^3;
   calculate_square(x) - which takes an integer **x** as an argument and returns the square of the given number x^2;
   Create the python file **pretty_print.py** that contains the following 2 functions:
   simple_print(x) - which takes an integer **x** as an argument and prints "Result: **x**", using the value of the argument **x**;
   pro_print(x) - which takes an integer **x** as an argument and prints "The result of the operation is **x**", using the value of the argument **x**;
   You should create a function **main()** inside the **calc.py** file. Inside the **main** function, you should calculate the square of the number 2, using the appropriate function and print the result using the function **simple_print(2)**. You should then calculate the cube of the number 4, using the appropriate function and print the result using the function **pro_print(4)**.

Decorators

8. Create the following list outside the functions **list1** = ['Anna', 'Edgar']. Create the function **add_values(list2)** which takes a list **list2** which contains some names of type String as an argument and adds the names which are in **list2** but not in **list1** to the list **list1.** Create a decorator which will print the list **list1** before and after calling the function **add_values(list2).**

9. Create a function that doesn't take any arguments that returns the following text "HI EVERYONE". Create 2 different decorators, one of them should make the letters of a string lowercase and the other one should add the following text to the string "!!! Welcome to the party.". Use the decorators on the function you have created in the first step. The final result should look like this: "Hi everyone!!! Welcome to the party."

Generators

10. Create the generator function **list_func(list1)** which takes the list **list1** as an argument and yields its values one by one.

11. Create the generator function **iter_num(n)** which takes an integer **n** as an argument and yields integers from 1 to **n.**

12. Create the generator function **power(max)** which calculates and yields the powers of 2, from 0 to the given integer **max** one by one**.** Example: if we call power(3) with max=3 it should yield the values 2^0, 2^1, 2^2, 2^3 one by one.

Homework 5

1. Create the function **max** which gets an undefined number of non-keyword arguments and returns the maximum of those. In case the function is called without arguments, it should return the text “no numbers given”. Don’t use the in-built functions for calculating the maximum value.

2. Create a function that gets a list as an argument and returns a new list, which contains only the unique values from the original list.

3. Create a function that doesn’t take any arguments and returns the text “Inside the function”. Create a decorator which will change the function so that the final result looks like this:

   “Before function call”
   “Inside the function”
   “After the function call”

   So, the decorator should add a string values before and after the function call.

4. Create a function that doesn’t take any arguments and returns the string “Hi”. Create 2 different decorators: one of them adds the string “, it’s me!” to the value returned by the function, the other one adds “<u>” and “</u>” to the value returned by the function. Use the decorator so that the final result looks like this:

   <u> Hi, it’s me! </u>

5. Create the function **my_range(n)** which gets the value **n** of type int as an argument and yields the values 0, 1, … , n-1, n until it reaches the value n+1 and prints “there are no values left”.

6. Create the modules **Productcheck.py** and **Customer.py** following the instructions below:

Productcheck.py

At the beginning of the file, outside the functions, create the dictionary products = {“candy”: 10, “juice”: 5, “pen”: 50}.

check(product, num): Gets the name of the product and its quantity as arguments and checks if the product with the given quantity is present in the dictionary **products** (you should check if there is a key with the product name and if the value at that key is >= the given quantity). The function should return True or False accordingly.

Customer.py

buy(product, num, price): Gets the name of the purchased product, the quantity and the price as arguments and checks if the product with the given quantity is present using the function **check(product, num)** from the module **Productcheck**. If the product with the given quantity is present, the function prints “You bought ***product*** and spent ***num*price***”, otherwise it prints “Sorry! We are out of this product.”:

main(): The function calls the function **buy(product, num, price)** with some values of your choice.