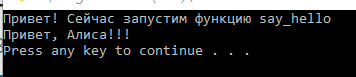
**Декораторы в Python**

Цель:

Научиться создавать и применять декораторы для модификации поведения функций, улучшения читаемости и повышения эффективности кода.

Задания:

1. Создайте декоратор greet\_decorator, который добавляет приветственное сообщение перед вызовом функции.



1. Создайте декоратор timer\_decorator, который измеряет время выполнения функции и выводит его.



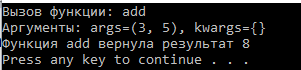
1. Создайте декоратор access\_control\_decorator, который проверяет разрешение пользователя перед выполнением функции.

У декоратора описываем функцию с тремя параметрами   
def wrapper(role, \*args, \*\*kwargs):

У декорируемой функции один параметр  
def delete\_user(user\_id):

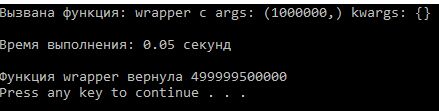


1. Создайте декоратор, который будет логировать вызовы декорированной функции, печатая имя функции и аргументы, с которыми она была вызвана.



Дополнительные задания:

Композиция декораторов: Попробуйте применить несколько декораторов к одной функции и посмотрите, как они взаимодействуют.



Примеры применения декораторов

#Пример 1 Декорирование функции с параметрами

def check(input\_func):

def output\_func(\*args, \*\*kwags):

print(args)

print(kwags["age"])

input\_func(\*args, \*\*kwags) # вызов оригинальной функции

return output\_func # возвращаем новую функцию

# определение оригинальной функции

@check

def print\_person(name, age):

print(f"Name: {name} Age: {age}")

# вызов оригинальной функции

print\_person("Tom", age=38)

#Пример 2 Декорирование функции без параметров. Применение одного декоратора к разным функциям.

def select(input\_func):

def output\_func():

print(f"Запуск функции {input\_func.\_\_name\_\_}")

input\_func()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

return output\_func

@select

def hello():

print("Пример работы декоратора!")

@select

def summa():

print(5+10)

hello()

summa()

Ссылка на теорию:

https://metanit.com/python/tutorial/2.28.php