**Dekoratory:**

**Zad 1.**

Napisz dekorator, który służyć będzie do logowania, z jakimi argumentami dana funkcja została wywołana. Skorzystaj z \*\*kwargs, \*args oraz zmiennej specjalnej \_\_name\_\_, aby logować również nazwę funkcji, którą wywołujemy.

Kod:

@logged

def func(\*args):

return 3 + len(args)

func(4, 4, 4)

Output:

you called func(4, 4, 4) it returned 6

**Zad 2.**

Stwórz dekorator, który będzie służył do przyozdabiania wyświetlanego tekstu gwiazdkami. Dowolny tekst ten ma być wyświetlany z poziomu dekorowanej funkcji.

**Efekt:**

\*\*\*\*\*\*\*\*\*\*\*\*

Hello World!

\*\*\*\*\*\*\*\*\*\*\*\*

**Zad 3.**

W szkoleniu nie zostało o tym wspomniane, ale możemy również określać przesyłane argumenty dekoratorów! Jedyna konieczność jaka będzie do zrealizowania, to dodanie kolejnej funkcji wrappującej, czyli np:

def arg\_check(arg):

def check(old\_func):

def new\_func():

# do sth with arg and call old\_func as examp

return new\_func

return check

@arg\_check(arg)

def examp(num):

# do sth

Twoje zadanie to stworzyć dekorator, który sprawdzać będzie, czy określony w dekoratorze typ jest zgodny z typem zmiennej przesłanej do funkcji.

Podpowiedź:

* Przesyłaj jako argument do dekoratora obiekt typu: int, float itd
* Sprawdzaj, czy typy są zgodne przy użyciu isinstance(zmienna, typ\_oczekiwany)
* Jeżeli typ będzie niezgodny, rzucaj wyjątkiem

**Zad 4.**

Utwórz dekorator @timethis mierzący czas wykonania dekorowanej funkcji. Wykorzystaj moduł time i metodę time.time().

**Zad 5.**

Zapoznaj się, do czego służą wbudowane w standard Pythona poniższe dekoratory: @property, @dataclass, @classmethod i @staticmethod.

Zbuduj proste programy przedstawiające realizację tych dekoratorów i różnice między nimi.