Lista zadań – Dekoratory

Zadanie 1 – Hello

Napisz własny dekorator o nazwie hello. Wynikiem jego działania powinno być:

- wypisanie na ekranie napisu hello
- wywołanie funkcji przekazanej jako argument.

Przykład kodu źródłowego z zastosowaniem dekoratora:

```
@hello
def moja_funkcja():
    print("moja funkcja")

moja_funkcja()
moja_funkcja()
```

```
hello
moja_funkcja
hello
moja_funkcja
```

Zadanie 2 – Tetranacci

Zaimplementuj własną funkcję o nazwie `tetranacci`, zwracający określony element ciągu Tetranacciego[1]. Funkcja powinna posiadać parametr o nazwie `n` określający numer wyrazu ciągu do obliczenia. Obliczenia wykonuj rekurencyjnie.

Zadanie 2.1 – Tetranacci cache

Wykorzystując dekoratory, napisz cache dla funkcji `tetranacci` z poprzedniego zadania. Ten dekorator powinien zapobiegać przed ponownym obliczaniem tych samych wartości.

1 https://pl.wikipedia.org/wiki/Ci%C4%85g_Fibonacciego#Ci.C4.85g_.E2.80.9ETetranacciego.E2.80.9D

Notatki:

.....

.....

Zadanie 3 – Nazwa funkcji

Napisz własny dekorator wypisujący nazwę dekorowanej funkcji.

Zacznij od najprostszej wersji dekoratora, obsługującego wyłącznie funkcje bez argumentów. Docelowo dekorator powinien być w stanie obsługiwać funkcje przyjmujące różne argumenty oraz zwracać wynik oryginalnej funkcji.

Użyj dekoratora do przetestowania kilku funkcji o różnej liczbie argumentów, aby zobaczyć, jak działa w praktyce.

Przykład działania:

```
DEBUG: first_function function called
DEBUG: second_function function called
DEBUG: third_function function called
```

Zadanie 3.1 – Argumenty funkcji

Rozszerz dekorator z poprzedniego zadania o wypisywanie informacji o przekazanych argumentach funkcji – nazwy i wartości.

Jak w poprzednim zadaniu, dekorator powinien być w stanie obsługiwać funkcje przyjmujące różne argumenty oraz zwracać wynik oryginalnej funkcji.

Użyj dekoratora do przetestowania kilku funkcji o różnej liczbie argumentów, aby zobaczyć, jak działa w praktyce.

Przykład działania:

```
DEBUG: first_function function called with 0 args:
DEBUG: first_function function called with 0 kwargs:
DEBUG: second_function function called with 3 args:
DEBUG: second_function function called  <arg0> = 1
DEBUG: second_function function called  <arg1> = 2
DEBUG: second_function function called  <arg2> = 3
DEBUG: second_function function called with 0 kwargs:
DEBUG: third_function function called with 2 args:
DEBUG: third_function function called  <arg0> = 1
```

Notatki:

.....

.....

```
DEBUG: third_function function called  <arg1> = 2
DEBUG: third_function function called with 1 kwargs:
DEBUG: third_function function called  c = 3
```

Zadanie 3.2 – Wartość zwracana

Rozszerz dekorator z poprzedniego zadania o wypisywanie informacji o wartości zwracanej.

Jak w poprzednim zadaniu, dekorator powinien być w stanie obsługiwać funkcje przyjmujące różne argumenty oraz zwracać wynik oryginalnej funkcji.

Przykład działania:

```
DEBUG: first_function function called with 0 args:
DEBUG: first_function function called with 0 kwargs:
DEBUG: first_function function called returned value: 1
DEBUG: second_function function called with 3 args:
DEBUG: second_function function called  <arg0> = 1
DEBUG: second_function function called  <arg1> = 2
DEBUG: second_function function called  <arg2> = 3
DEBUG: second_function function called with 0 kwargs:
DEBUG: second_function function called returned value: 6
DEBUG: third_function function called with 2 args:
DEBUG: third_function function called  <arg0> = 1
DEBUG: third_function function called  <arg1> = 2
DEBUG: third_function function called with 1 kwargs:
DEBUG: third_function function called  c = 3
DEBUG: third_function function called returned value: 6
```

Zadanie 4 – Profiling

Twoim zadaniem jest stworzenie dekoratora, który będzie mierzył czas wykonywania dekorowanej funkcji i wypisywał go na standardowe wyjście. Warto zadbać o to, aby wynik pomiaru czasu był czytelny.

Dekorator powinien być w stanie obsługiwać funkcje przyjmujące różne argumenty oraz zwracać wynik oryginalnej funkcji.

Notatki:

.....

.....

Użyj dekoratora do przetestowania funkcji o różnym czasie wykonania.

Przykład działania:

```
DEBUG: first_function function execution took: 0:00:00.000009
DEBUG: second_function function execution took: 0:00:00.000003
DEBUG: third_function function execution took: 0:00:00.000002
```

Zadanie 5 – Łączenie dekoratorów

Twoim zadaniem jest zastosowanie dekoratora wypisującego nazwę dekorowanej funkcji oraz dekoratora do mierzenia czasu wykonania dekorowanej funkcji. Oba dekoratory powinny udekorować jednocześnie tą samą funkcję.

Czy dekoratory wypisują prawidłową nazwę funkcji? Czy wypiszą poprawną nazwę, gdy zmienisz ich kolejność?

Zadanie 6 – Login required

Napisz program, który do uruchomienia będzie wymagał nazwy użytkownika i hasła lub wybrania opcji logowanie anonimowe. Napisz dekorator o nazwie `login_required`, który zaaplikowany do dowolnej funkcji zweryfikuje czy użytkownik zalogował się poprawnie. W przypadku, gdy:

- użytkownik zalogował się poprawnie, dekorator powinien wywoływać otaczaną funkcję,
- użytkownik nie został zalogowany, dekorator powinien rzucić wyjątek o nazwie `NotLoggedIn`.

Przykład działania:

```
Zaloguj się
uzytkownik: jan
haslo: kowalski
Menu
1. Wypisz hello
2. Zakoncz
```

Notatki:

.....

.....

```
Wybor: 1
Hello
```

Przykład działania:

```
Zaloguj się
uzytkownik: anonim
Menu
1. Wypisz hello
2. Zakoncz
Wybor: 1
Musisz sie zalogowac
```

Notatki:

.....

.....