A Decorator is just a function that takes another function as an argument, add some kind of functionality and then returns another function. All of this without altering the source code of the original function that you passed in.

Write a Python program to convert a given lower case string into upper case string using a decorator.

Answer:

```
def uppercase_decorator(func):
    def wrapper(input_str):
        upper_case_input = input_str.upper()
        return func(upper_case_input)
    return wrapper

@uppercase_decorator
def process_input(input_str):
    print("Processed input:", input_str)

input_str = "welcome to code paradise"
process_input(input_str)
```

Output:

Processed input: WELCOME TO CODE PARADISE

Write a Python program using a decorator to calculate the execution time of a function.

Answer:

```
import time

def timing_decorator(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
```

```
        result = func(*args, **kwargs)
        end_time = time.time()
        execution_time = end_time - start_time
        print(f"Execution time of {func.__name__}: {execution_time:.4f} seconds")
        return result
    return wrapper

@timing_decorator
def slow_function():
    total = 0
    for i in range(1, 1000000):
        total += i
    return total

slow_function()
```

Output:

Execution time of slow_function: 0.0734 seconds