Exercises

1. Timer Decorator

Mark as done

Write a decorator that prints the time it takes a function to be executed.

Example for running 2 functions that sum numbers in a range:

sum1 = sum\_many(10000000)

sum2 = sum\_many\_more(50000000)

your decorator should print something like:

function runtime: 0:00:01.184106

function runtime: 0:00:05.506641

1. Print Function Information

Mark as done

Write a decorator that prints information about a function execution.

For each function call the decorator should :

* 1. print some data about the function call
  2. Name of the function
  3. all arguments (regular, \*args, \*\*kwargs)
  4. execute the original function
  5. print all the returning data. (value and type)

Here is an example.

**Calling:**

add(1, 2, age=3, text="hello")

**Should return**:

func name: add

args: (1, 2)

kwargs: {'age': 3, 'text': 'hello'}

return value: 3

return type: <class 'int'>

1. Slow Down

Mark as done

Write a decorator that slows down the function by 1 second.

1. Number of calls to a func

Mark as done

* 1. Write a decorator that keeps count for how many calls have been done to a function.
  2. Print the number of calls each time the function finishes to execute.

1. Challenging

Cache Decorator

Mark as done

Write a decorator that implements a cache mechanism.

The decorator should store results of function calls that already have been made. If a function is being called and the result was already calculated, then the function should return the result immediately, instead of calculate it again.

Test it on a Fibonacci function.

You can write in the first line of the Fibonacci function a print that will tell you which number you are calculating:

def fibonacci(num):

print("calculating: ", num)

Then when you run the function, each value should be calculated only once.

So running :

print(fibonacci(2))

print(fibonacci(4))

Should output:

calculating:  2

calculating:  1

calculating:  0

1

calculating:  4

calculating:  3

3

Running the function without the decorator would print:

calculating:  2

calculating:  1

calculating:  0

1

calculating:  4

calculating:  3

calculating:  2

calculating:  1

calculating:  0

calculating:  1

calculating:  2

calculating:  1

calculating:  0

3

Make sure your code is generic and you can use the decorator for different functions.

**Bonus**: support more cases.

For example:

def add(x, \*, addition=0, more=0):

print("running add")

return x + 1 + addition + more

Note: the \* make the kwargs "mandatory kwargs", which mean we must pass them as a key value pair.

You can read more about it [here](https://www.python.org/dev/peps/pep-3102/).

Then when you run the function, each value should be calculated only once.

So running :

add(1, addition=0, more=0)

add(1, more=0, addition=0)

add(1, more=0)

add(1)

Should print:

running  add