**Problem statement**[#](https://www.educative.io/courses/mastering-the-art-of-programming-in-python-3/JPx51vpD58y#problem-statement)

In this exercise, we want to *time* our functions. What we want is that whenever a class method executes, it should print the time taken by itself.

Let’s get into detail. Consider the scenario below while attempting the challenge.

We have an Animal class. It has a function talk as follows:

class Animal():  
   
    def talk(self):  
        time.sleep(1)  
        print("Animal talk")

And, we have a decorator time\_taken that calculates the required time taken by any function.

Now, if we want to calculate the time for talk, we can simply add @time\_taken before its header, and the problem is solved!

But what if we create child classes of Animal in the future and want to decorate all the functions of child classes? Adding @time\_taken at the beginning of every definition would be time-taking and hectic.

To avoid such a repetitive fuss, if we know beforehand that every subclass must have this timing property, then we should look up to **metaclass** based solution.

So, the challenge is that when the classes are created, they should be immediately wrapped up by a timing method decorator. The solution requires *three* steps:

* Create a metaclass TimeMeta feeding class object to the class decorator method
* Create a class decorator method timeit that makes use of time\_taken decorator to debug class methods.
* Create time\_taken decorator that times the passed method by keeping track of *start time* and *end time*.