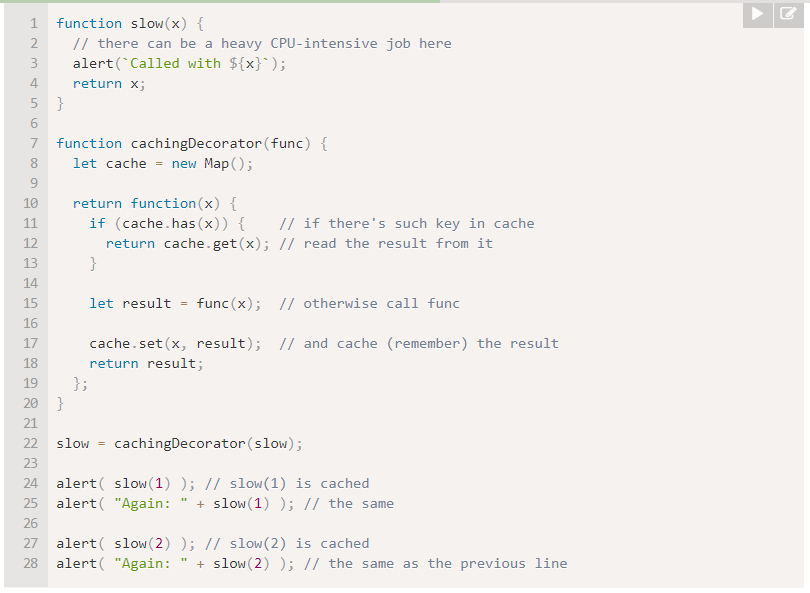
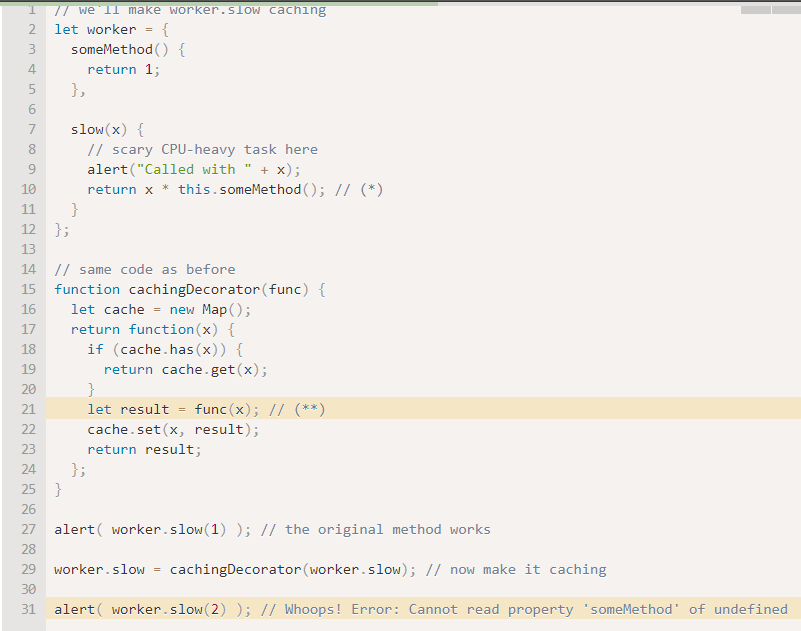
**\*\*\*\*\*\*DECORATORS AND FORWARDING, CALL/APPLY\*\*\*\***

* **TRANSPARENT CACHING :**
* Giả sử ta có 1 hàm như slow(x) sẽ phải chạy rất lâu và tốn cpu. Nhưng kết quả là stable tức là cùng 1 input sẽ cho ra cùng 1 output.
* Khi đó ta muốn bắt và lưu kết quả đã có trước đó, để khi gọi lại sẽ chỉ cần trả về kết quả đó. Tiết kiệm được time.
* Thay vì dùng 1 hàm bên trong slow() thì ta sẽ dùng hàm wrapper để caching nó.



* Ở trên, cachingDecorator sẽ là decorator, là 1 hàm đặc biệt sẽ nhận 1 hàm khác và thay đổi hành vi của hàm nó nhận vào.
* Ý tưởng là có thể gọi cachingDecorator cho bất cứ hàm nào, và sẽ trả về 1 caching wrapper.
* Caching không làm phức tạp hàm lên, xem kỹ phần code để hiểu.
* **USING “func.call” FOR THE CONTEXT**
* Caching decorator sẽ không phù hợp với object.



* Ta thấy khi pass method vào ta chỉ pass than hàm nên this sẽ là undefined.
* Có 1 special built-in function method là func.call(context, arg1, arg2,…) gọi 1 hàm với this được xác định hay setting.

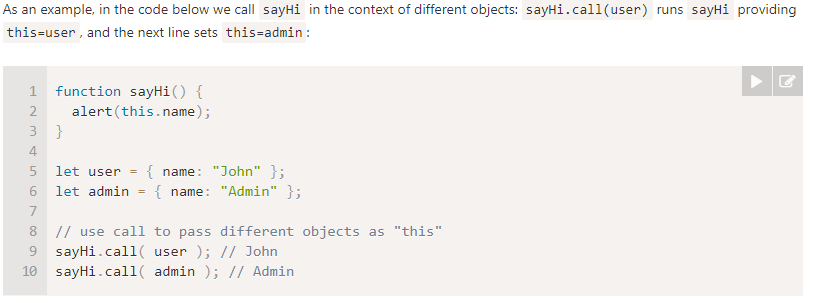
**Func.call(context, arg1, arg2,… )**

* Context sẽ là this trong func, các giá trị còn lại sẽ là arguments của func
* VD :

func(1,2,3) ;

func.call(obj,1,2,3)

* ở vd dưới, func.call sẽ xét thêm cho this là object



* Sửa lỗi ở ví dụ trên :



* Sau decorator worker.slow thì lúc này sẽ là function(x) {….}
* Khi worker.slow(2) được thực thì, nhận 2 làm argument và this=worker.
* Với trường hợp có 2 arguments, xem thêm ở web
* **func.apply()**
* Ngoài func.call ta cũng có thể sử dụng func.apply(). Sẽ chỉ khác ở chỗ, func.apply() sẽ nhận 1 array là các đối số trong khi func.call() nhận danh sách các đối số riêng lẻ.