

# 20377199 赵芮箐 第8周作业

请用函数或类实现不同功能的装饰器并进行充分的测试。

1. 请用函数实现装饰器。部分函数往往需要将模型或者数据处理结果保存下来，但实际调用时却因为路径设置错误等原因导致文件无法存储，浪费大量的时间重复运行程序。请实现一个装饰器，接收函数的路径参数，检查路径对应文件夹是否存在，若不存在，则给出提示，并在提示后由系统自动创建对应的文件夹。

2. 请用类实现一个装饰器。部分函数可能需要花费较长时间才能完成，请实现一个装饰类，其能够在被装饰函数结束后通过声音给用户发出通知。了解并使用一下playsound或其他声音文件处理的库。另外，可否对根据返回值的类型，比如整数，元组等，来实现不同的声音通知？如果返回值有多个，可否多次按类型依次播放？

3. 请用类或者函数实现一个装饰器。部分函数可能会在运行过程中输出大量的中间状态或者中间结果，这些信息往往在程序出问题利于调试，但由于输出内容过多，可能在控制台中无法全部查看。请实现一个装饰器，其能够将装饰函数在运行过程中的所有的输出（通过print）全部保存在特定的一个文件中。

4. 实现一个类，在其中提供一些方法模拟耗时内存的一些操作，如大的数据结构生成、遍历、写入文件序列化等，并通过其体验line\_profiler、memory\_profiler、tqdm、pysnoper等装饰器的相关功能。

## 任务一：用函数实现【检查路径装饰器】

```
import os
from functools import wraps

# 用函数实现文件路径检查装饰器
def path_examine(path):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            if os.path.exists(path) == False:      # 检查路径是否存在
                print('The path does not exist! We creat for you.')
                os.mkdir(path)                      # 创建对应文件夹
            return func(*args,**kwargs)
        return wrapper
    return decorator

# 测试
path = 'test'
@ path_examine(path)

def test(path):
    print(os.path.exists(path))      # 检查路径是否存在输出布尔值
test(path)
```

- 无装饰器效果:

```
PS D:\code\mp2022> python -u "d:\code\mp2022\week8\week8.py"
False
```

- 有装饰器效果:

```
PS D:\code\mp2022> python -u "d:\code\mp2022\week8\week8.py"
The path does not exist! We creat for you.
True
```

给出了提示，并且建立了相应的文件夹

## 任务二：用类实现【声音提醒函数结束装饰器】

```
from functools import wraps
from playsound import playsound

# 用类实现声音提醒函数结束装饰器
class End_Music:
    def __init__(self):
        self.music_path_int = 'D:/code/mp2022/week8/music/int.mp3'
        self.music_path_list = 'D:/code/mp2022/week8/music/list.mp3'
        self.music_path_tuple = 'D:/code/mp2022/week8/music/tuple.mp3'

    def __call__(self, func):
        @ wraps(func)
        def wrapper(*args, **kwargs):
            ans_list = func(*args, **kwargs)
            for ans in ans_list:
                if isinstance(ans, int):
                    playsound(self.music_path_int)
                elif isinstance(ans, list):
                    playsound(self.music_path_list)
                elif isinstance(ans, tuple):
                    playsound(self.music_path_tuple)
            return func(*args, **kwargs)
        return wrapper

# 测试
@ End_Music()
def test():
    return 1, [1,2], (1,2)

test()
```

- 装饰器效果：

突然一时间不知道该怎么展示结果哈哈哈，就是按类型听到音乐了！

## 任务三：用类或函数实现【保存输出装饰器】

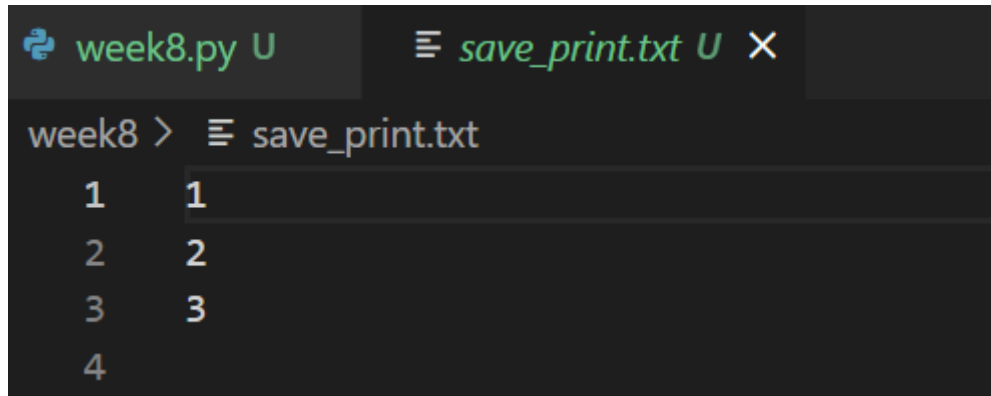
```
import sys
from functools import wraps

# 用函数实现保存输出装饰器
def save_print(save_path):
    def decorator(func):
        @ wraps(func)
        def wrapper(*args, **kwargs):
            sys.stdout = open(save_path, 'w')
            return func(*args, **kwargs)
        return wrapper
    return decorator

# 测试
@ save_print('week8/save_print.txt')
def test():
```

```
print('1')
print('2')
print('3')
test()
```

- 装饰器效果:



```
week8 > save_print.txt
1      1
2      2
3      3
4      4
```

将运行过程中的输出全部存入特定文件

## 任务四：用类模拟耗时耗内存的操作，并体验相关装饰器功能

```
import time
import random
import pysnooper
from tqdm import tqdm
from memory_profiler import profile
from line_profiler import LineProfiler

# 用类模拟耗时耗内存的操作
# 这里用生成一个大字典（结构为"ID:成绩"）模拟耗内存操作，用遍历字典查询考某成绩的学生模拟耗时操作
class Grade:
    def __init__(self):
        self.dic = {}

    @ profile                                     # memory profile
    @ pysnooper.snoop('week8/dic.log')           # pysnooper
    def input_grade(self):
        """
        录入成绩的方法，即生成大字典，模拟耗内存操作
        """
        for i in range(10 ** 2):
            self.dic[i] = random.randint(0,100)

    def search(self, grade):
        """
        查询成绩的方法，即遍历大字典，模拟耗时操作
        """
        stu = []
        for key in tqdm(self.dic):               # tqdm
            time.sleep(0.000001)
            if self.dic[key] == grade:
                stu.append(key)
        return stu
```

```
# 测试
g = Grade()
g.input_grade()

lp = LineProfiler()
lp.enable_by_count()
lp_wrapper = lp(g.search)
stu = g.search(90)
lp.print_stats()
```

- 装饰器效果:

```

Filename: d:\code\mp2022\week8\week8.py

Line #      Mem usage      Increment  Occurrences   Line Contents
=====
89          43.2 MiB          43.2 MiB           1       @ profile
90                                     def input_grade(self):
91                                     """
92                                     录入成绩的方法，即生成大字典，模拟耗内存操作
93                                     """
94          51.3 MiB           0.0 MiB       100001       for i in range(10 ** 5):
95          51.3 MiB           8.1 MiB       100000           self.dic[i] = random.randint(0,100)

100% | 100000/100000 [25:56<00:00, 64.25it/s]
Timer unit: 1e-07 s

Total time: 1555.34 s
File: d:\code\mp2022\week8\week8.py
Function: search at line 97

Line #      Hits          Time  Per Hit   % Time  Line Contents
=====
97                                     def search(self, grade):
98                                     """
99                                     查询成绩的方法，即遍历大字典，模拟耗时操作
100                                    """
101          1          24.0      24.0      0.0      stu = []
102       100001  252822995.0  2528.2    1.6      for key in tqdm(self.dic):
103       100000  15280535593.0  152805.4  98.2          time.sleep(0.000001)
104       100000          19983006.0    199.8    0.1          if self.dic[key] == grade:
105          1028          83425.0    81.2    0.0              stu.append(key)
106          1          17.0      17.0    0.0      return stu

```

memory\_profiler装饰器：显示执行该行解释器的内存使用情况以及当前行相对于上一行的内存差

line\_profiler装饰器：显示每一行的运行次数、时间及占整个程序的比重

tqdm装饰器：显示运行的进度条

```

week8.py M  ≡ dic.log  U X
week8 > ≡ dic.log
1 Source path:... d:\code\mp2022\week8\week8.py
2 Starting var:... self = <__main__.Grade object at 0x0000018348B50070>
3 14:23:52.644355 call 92 def input_grade(self):
4 14:23:52.645351 line 96 for i in range(10 ** 2):
5 New var:..... i = 0
6 14:23:52.645351 line 97 self.dic[i] = random.randint(0,100)
7 14:23:52.646355 line 96 for i in range(10 ** 2):
8 Modified var:... i = 1
9 14:23:52.646355 line 97 self.dic[i] = random.randint(0,100)
10 14:23:52.647346 line 96 for i in range(10 ** 2):
11 Modified var:... i = 2
12 14:23:52.647346 line 97 self.dic[i] = random.randint(0,100)
13 14:23:52.648346 line 96 for i in range(10 ** 2):
14 Modified var:... i = 3
15 14:23:52.650348 line 97 self.dic[i] = random.randint(0,100)
16 14:23:52.650348 line 96 for i in range(10 ** 2):
17 Modified var:... i = 4
18 14:23:52.650348 line 97 self.dic[i] = random.randint(0,100)
19 14:23:52.651335 line 96 for i in range(10 ** 2):
20 Modified var:... i = 5
21 14:23:52.651335 line 97 self.dic[i] = random.randint(0,100)
22 14:23:52.651335 line 96 for i in range(10 ** 2):

```

pysnooper装饰器：自动识别到语句和变量，将其值print出来或其输出到log文件中

## 代码:

[https://github.com/rachhhhing/mp2022\\_python/blob/master/week8/week8.py](https://github.com/rachhhhing/mp2022_python/blob/master/week8/week8.py)

## Ref:

- os模块: <https://zhuanlan.zhihu.com/p/150835193>
- 声音文件处理: [https://blog.csdn.net/via\\_2020/article/details/108270536](https://blog.csdn.net/via_2020/article/details/108270536)
- line\_profiler: <https://www.cnblogs.com/dechinphy/p/line-profiler.html>
- memory\_profiler: <https://zhuanlan.zhihu.com/p/121003986>
- tqdm: <https://zhuanlan.zhihu.com/p/163613814>
- pysnooper: <https://zhuanlan.zhihu.com/p/408470180>