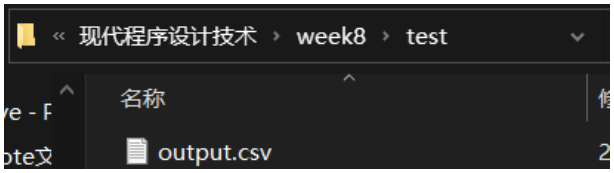
```python
import os
import sys
import time
import pandas as pd

from tqdm import tqdm
from functools import wraps, partial
from collections import Iterable
from line_profiler import LineProfiler
from memory_profiler import profile
from pysnooper import snoop
```

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。

```python
def check(fun):
    @wraps(fun)
    def wrapper(*args,**kwargs):
        folder_path = args[1]
        if not os.path.exists(folder_path):
            print('--Attention: 文件夹不存在，将自动创建--')
            os.mkdir(folder_path)
        return fun(*args,**kwargs)
    return wrapper

@check
def output(df, folder_path):
    df.to_csv(folder_path+'output.csv')


df = pd.DataFrame({k:[k for i in range(4)] for k in range(3)})
output(df, 'test/')
```



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？

```python
#2
class Inform:
    def __call__(self, fun):
        @wraps(fun)
        def wrapper(*args,**kwargs):
            sound = {'NoneType': 'Bottle.ogg',
                     'bool': 'Flute.ogg',
                     'int': 'WindChime.ogg',
                     'float': 'FadeOut.ogg',
                     'str': 'Guitar.ogg',
                     'list':'AcousticGuitar浪人吉他.ogg',
                     'dict': 'Reveries回转记忆.ogg',
                     'tuple': 'IceLatte冰拿铁.ogg',
                     }
            result = fun(*args,**kwargs)
            result_type =  type(result).__name__
            os.system('sound/' + sound[result_type])
            if result_type == 'tuple' or result_type == 'list':
                for i in range(3):
                    result_type = type(result[i]).__name__
                    os.system('E:/北航/课业/大三上/现代程序设计技术/week8/sound/'
                              + sound[result_type])
                    time.sleep(2)
            return result
        return wrapper

@Inform()
def inform_test(x):
    return x

# 测试
inform_test([None, 'str', {0:0}])
```

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时利于调试，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，其能够将被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。

```python
def intermediate(fun):
    @wraps(fun)
    def wrapper(*args,**kwargs):
        savedStdout = sys.stdout  # 标准输出流
        with open("printlog.txt","w") as print_log:
            sys.stdout = print_log  # 输出流指向pring_log
            result = fun(*args,**kwargs)
            sys.stdout = savedStdout  # 重置输出流
        return result
    return wrapper

@intermediate
def intermediate_test():
    for i in range(5):
        print(i+1)

# 测试
intermediate_test()
```

printlog.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
1
2
3
4
5
```

4. 实现一个类，在其中提供一些方法模拟耗时耗内存的一些操作，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、pysnooper 等装饰器的相关功能。

```python
class simulating:
    def __init__(self, n):
        self.dimension = n

    @profile
    def generate(self):
        self.data = [[i+1 for i in range(self.dimension)] for j in range(self.dimension)]

    def iterate(self, fun = lambda x: None):
        for i in range(self.dimension):
            for j in range(self.dimension):
                if fun(self.data[i][j]):
                    self.data[i][j] = fun(self.data[i][j])

    @snoop()
    def cal(self):
        a = sum(self.data[0])
        b = sum(self.data[1])
        return a / b

# 测试
simulation = simulating(100)
simulation.generate()  # 测试profile

square = partial(pow, exp=2)
lp = LineProfiler()
lp.add_function(square)
lp_wrapper = lp(simulation.iterate)
lp_wrapper(square)
lp.print_stats()  # 测试LineProfiler

bar = tqdm(simulation.data[0][:3])  # 测试tqdm
for x in bar:
    print(x)
    time.sleep(1)

simulation.cal()  # 测试snoop
```

memory_profiler.rofile

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    18     92.2 MiB     92.2 MiB           1       @profile
    19                                             def generate(self):
    20     92.2 MiB      0.0 MiB       10303           self.data = [[i+1 for i in
range(self.dimension)] for j in range(self.dimension)]
```

line_profiler.LineProfiler

```
Timer unit: 1e-07 s

Total time: 0.0371508 s
File: E:/北航/课业/大三上/现代程序设计技术/week8/week8_homework.py
Function: iterate at line 98

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    98                                           def iterate(self, fun = lambda x: None):
    99       101        538.0      5.3      0.1          for i in range(self.dimension):
   100     10100      52802.0      5.2     14.2              for j in range(self.dimension):
   101     10000     148184.0     14.8     39.9                  if fun(self.data[i][j]):
   102     10000     169984.0     17.0     45.8                      self.data[i][j] =
fun(self.data[i][j])
```

tqdm.tqdm

```
  0%|          | 0/3 [00:00<?, ?it/s]1
 33%|███▎      | 1/3 [00:01<00:02,  1.01s/it]4
 67%|██████▋   | 2/3 [00:02<00:01,  1.00s/it]9
100%|██████████| 3/3 [00:03<00:00,  1.00s/it]
```

pysnooper.snoop

```
Source path:... E:/北航/课业/大三上/现代程序设计技术/week8/week8_homework.py
Starting var:.. self = <__main__.simulating object at 0x000002294FE0A940>
21:37:29.800675 call       105     def cal(self):
21:37:29.801885 line       106         a = sum(self.data[0])
New var:....... a = 5050
21:37:29.801885 line       107         b = sum(self.data[1])
New var:....... b = 5050
21:37:29.801885 line       108         return a / b
21:37:29.802676 return     108         return a / b
Return value:.. 1.0
Elapsed time: 00:00:00.002001
```