Author: Dr Ambikesh Jayal ambi1999@gmail.com Demo code to show an example of decorator function to show execution time of other functions.

Code also shows the efficiency of NumPy which is a scientific computing package in Python.

Created on Wed May 12 2021, Modified on Tuesday June 15 2021

Output oexecuting this program on my computer is below. As you can see the function processUsingNumpy takes far less time than the function processUsingDataFrame. Both function do the same task.

Function 'processUsingNumpy' starts executing Function 'processUsingNumpy' has finished execution. Execution time equals to 799.1797924041748 millisecondss

Function 'processUsingDataFrame' starts executing Function 'processUsingDataFrame' has finished execution. Execution time equals to 15878.308773040771 millisecondss

In [1]:

```
# import pandas and numpy libraries
import pandas as pd
import numpy as np
import time

from functools import wraps
def timer(func):
    @wraps(func)
    def wrapper(*args,**kwargs):
        print(f"Function {func.__name__!r} starts executing")
        startTime = time.time()
        result = func(*args,**kwargs)
        endTime=time.time()
        #execution_time in milliseconds
        executionTime=round((endTime-startTime)*1000)
        print(f"Function {func.__name__!r} has finished execution. \n****Exec
        return result
    return wrapper


def timer_simpler(func):
    @wraps(func)
    def wrapper(param1):
        print(f"Function {func.__name__!r} starts executing")
        startTime = time.time()
        result = func(param1)
        endTime=time.time()
        #execution_time in milliseconds
        executionTime=(endTime-startTime)*1000
        #print(f"{func.__name__!r} finished. Execution time (in ) in {time.ti
        print(f"Function {func.__name__!r} has finished execution. \nExecutio
        return result

    return wrapper


#@timer_simpler
@timer
def processUsingDataFrame(df):
    min=df.min(axis="rows").min()
    max=df.max(axis="rows").max()
    average=df.mean(axis="rows").mean()
    return min,max,average
```

```
#@timer_simpler
@timer
def processUsingNumpy(df):
    #ndarray1=df.values
    ndarray1=df.to_numpy()
    min=ndarray1.min(axis=1).min()
    max=ndarray1.max(axis=1).max()
    average=ndarray1.mean(axis=1).mean()
    return min,max,average


#arrayrandomnumbers= np.random.randint(1,1000,10)
#array2d=arrayrandomnumbers.reshape(2,-1)

#Generate a large array of random numbers, convert it into multidimensional a
#arrayrandomnumbers= np.random.randint(1,100000000,20000000)
arrayrandomnumbers= np.random.randint(1,10000000000,200000000)
array2d=arrayrandomnumbers.reshape(100,-1)


df=pd.DataFrame(array2d)
#ndarray1=df11.values
#ndarray2=df11.to_numpy()
res1=processUsingNumpy(df)
res2=processUsingDataFrame(df)
```

```
Function 'processUsingNumpy' starts executing
Function 'processUsingNumpy' has finished execution.
****Execution time of 'processUsingNumpy' function equals to 785 milliseconds
s
Function 'processUsingDataFrame' starts executing
Function 'processUsingDataFrame' has finished execution.
****Execution time of 'processUsingDataFrame' function equals to 18577 millis
econdss
```

In [ ]:

In [ ]: