作业一：

import time

import random

def test\_tuple\_performance():

# 创建10000x10000的tuple矩阵

matrix = tuple(tuple(0 for \_ in range(10000)) for \_ in range(10000))

start\_time = time.time()

# 进行10000轮修改，每次修改一个元素

for \_ in range(10000):

i = random.randint(0, 9999)

j = random.randint(0, 9999)

# 由于tuple不可变，需要创建新的tuple

new\_row = tuple(v if idx != j else random.random() for idx, v in enumerate(matrix[i]))

matrix = tuple(row if idx != i else new\_row for idx, row in enumerate(matrix))

end\_time = time.time()

return end\_time - start\_time

def test\_list\_performance():

# 创建10000x10000的list矩阵

matrix = [[0 for \_ in range(10000)] for \_ in range(10000)]

start\_time = time.time()

# 进行10000轮修改，每次修改一个元素

for \_ in range(10000):

i = random.randint(0, 9999)

j = random.randint(0, 9999)

matrix[i][j] = random.random()

end\_time = time.time()

return end\_time - start\_time

# 运行测试

tuple\_time = test\_tuple\_performance()

list\_time = test\_list\_performance()

print(f"Tuple 执行时间: {tuple\_time:.4f} 秒")

print(f"List 执行时间: {list\_time:.4f} 秒")

print(f"List 比 Tuple 快: {tuple\_time/list\_time:.2f} 倍")

作业二：

import random

import string

import numpy as np

def generate\_random\_sample(structure, \*\*kwargs):

"""

生成指定结构的随机样本数据

参数:

structure: 描述数据结构的字典或列表

kwargs: 可选参数，用于控制样本数量和其他属性

返回:

生成的随机样本数据

"""

# 控制样本数量的参数

n\_samples = kwargs.get('n\_samples', 1)

# 定义基本数据类型的生成函数

def generate\_primitive(primitive\_type):

if primitive\_type == 'int':

return random.randint(0, 100)

elif primitive\_type == 'float':

return random.uniform(0, 100)

elif primitive\_type == 'bool':

return random.choice([True, False])

elif primitive\_type == 'str':

length = random.randint(3, 10)

return ''.join(random.choice(string.ascii\_letters) for \_ in range(length))

elif primitive\_type == 'none':

return None

else:

raise ValueError(f"不支持的原始类型: {primitive\_type}")

# 递归生成嵌套结构

def generate\_nested(struct):

if isinstance(struct, dict):

return {k: generate\_nested(v) for k, v in struct.items()}

elif isinstance(struct, list):

if len(struct) == 0:

return []

elif isinstance(struct[0], dict) or isinstance(struct[0], list):

# 列表中的元素是复杂类型

return [generate\_nested(struct[0]) for \_ in range(random.randint(1, 5))]

else:

# 列表中的元素是原始类型

return [generate\_primitive(struct[0]) for \_ in range(random.randint(1, 5))]

elif isinstance(struct, str):

# 假设字符串表示原始类型

return generate\_primitive(struct)

else:

raise ValueError(f"不支持的结构类型: {type(struct)}")

# 生成指定数量的样本

samples = [generate\_nested(structure) for \_ in range(n\_samples)]

# 如果只需要一个样本，直接返回而不是列表

return samples[0] if n\_samples == 1 else samples

# 示例使用

if \_\_name\_\_ == "\_\_main\_\_":

# 定义一个嵌套结构

structure = {

"name": "str",

"age": "int",

"is\_student": "bool",

"grades": ["float"],

"address": {

"street": "str",

"city": "str",

"zip": "str"

},

"friends": [

{

"name": "str",

"age": "int"

}

]

}

# 生成样本

sample = generate\_random\_sample(structure, n\_samples=3)

print(sample)

作业三：

import numpy as np

from functools import wraps

def statistics\_decorator(operations=None):

"""

统计修饰器，用于计算嵌套数据结构中所有数值型数据的统计值

参数:

operations: 要执行的统计操作列表，可选值为 'sum', 'avg', 'var', 'rmse'

返回:

修饰后的函数，该函数除了返回原始结果外，还会打印统计信息

"""

# 设置默认操作

if operations is None:

operations = ['sum', 'avg', 'var', 'rmse']

def decorator(func):

@wraps(func)

def wrapper(\*args, \*\*kwargs):

# 调用原始函数获取结果

result = func(\*args, \*\*kwargs)

# 收集所有数值

numbers = []

def collect\_numbers(data):

if isinstance(data, (int, float, np.number)):

numbers.append(data)

elif isinstance(data, dict):

for value in data.values():

collect\_numbers(value)

elif isinstance(data, (list, tuple, set)):

for item in data:

collect\_numbers(item)

# 处理单个样本或样本列表

if isinstance(result, list):

for sample in result:

collect\_numbers(sample)

else:

collect\_numbers(result)

# 执行统计操作

stats = {}

if not numbers:

print("警告: 未找到数值型数据")

return result

if 'sum' in operations:

stats['sum'] = sum(numbers)

if 'avg' in operations:

stats['avg'] = np.mean(numbers)

if 'var' in operations:

stats['var'] = np.var(numbers)

if 'rmse' in operations:

stats['rmse'] = np.sqrt(np.mean(np.square(numbers)))

# 打印统计结果

print("统计结果:")

for op, value in stats.items():

print(f"{op.upper()}: {value}")

return result

return wrapper

return decorator

# 示例使用

if \_\_name\_\_ == "\_\_main\_\_":

# 导入作业2的函数

from random\_data\_generator import generate\_random\_sample

# 定义一个嵌套结构

structure = {

"name": "str",

"age": "int",

"is\_student": "bool",

"grades": ["float"],

"address": {

"street": "str",

"city": "str",

"zip": "str"

},

"friends": [

{

"name": "str",

"age": "int"

}

]

}

# 使用修饰器修饰函数

@statistics\_decorator(operations=['sum', 'avg'])

def get\_samples():

return generate\_random\_sample(structure, n\_samples=3)

# 调用修饰后的函数

samples = get\_samples()

print("\n生成的样本:")

print(samples)