****南昌大学实验报告

学生姓名：马星 学号：5418122020 专业班级：计算机科学与技术(卓越工程师计划)221班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

**实验三 列表与元组**

1. 实验目的

1.了解 Python 序列类型的基本概念与常用操作;

2.掌握列表、元组、字符串和数字序列等常用序列类型的使用方法

3.理解列表与元组的区别。

1. 实验内容

1.练习列表、元组、字符串和数字序列对象的创建方法。

2.利用多种方法实现斐波那契数列的输出，并比较不同实现方法的性能

3.完成教材 93 页的课后练习第 1、5、6题

1. 实验步骤

1.练习列表、元组、字符串和数字序列对象的创建方法。

# 列表对象创建

list1 = []

list2 = list()

list3 = [0, 1, 2, 3, 4]

list4 = [i for i in range(5)]

# 列表对象创建

tuple1 = ()

tuple2 = (1,)

tuple3 = tuple([1, 2])

# 字符串对象创建

s1 = ""

s2 = ''

s3 = str(1)

s4 = str([1, 2, 3])

s5 = str((1, 2))

s6 = [1, 2, 3].\_\_str\_\_()

# 数字序列对象创建

r1 = range(5)

r2 = range(3, 8)

r3 = range(6, 2, -1)

2.利用多种方法实现斐波那契数列的输出，并比较不同实现方法的性能

import time

n = 30

def printTime(func):

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

start = time.perf\_counter()

result = func(\*args, \*\*kwargs) # 调用目标函数

end = time.perf\_counter()

print(f"\033[32m {func.\_\_name\_\_} 运行耗时 {(end - start) \* 1000} ms \033[0m")

return result

return wrapper

@printTime

def method1(n):

if n < 0:

raise Exception("项数非负")

"""打印斐波那契数列前n项"""

print("[", end="")

def fun(a, b, n):

"""从a和b开始向后递推n项"""

if n == 0:

return

print(a, end=', ')

fun(b, a + b, n - 1)

fun(1, 1, n) # 使用函数栈进行递推

if n != 0:

print('\b\b', end='')

print(']')

method1(n)

@printTime

def method2(n):

if n < 0:

raise Exception("项数非负")

a, b = 0, 1

print("[", end="")

for i in range(n):

print(b, end=', ')

a, b = b, a + b # 递推

if n != 0:

print('\b\b', end='')

print("]")

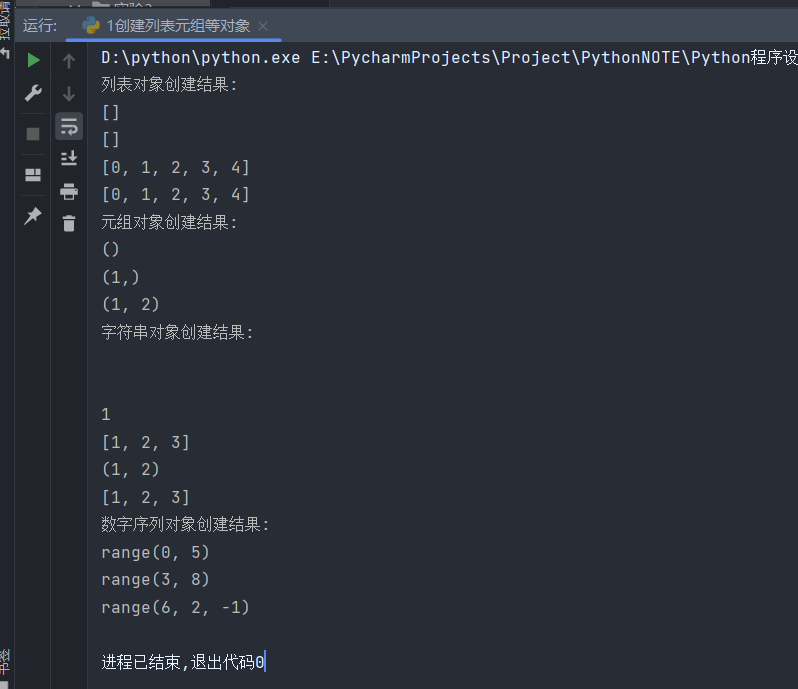
method2(n)

method1使用了递归的方式实现打印斐波那契数列, 通过前两项的参数迭代, 求解出后一项, 以及项数n的每次减1控制了递归深度

method2使用动态规划/递推的方式实现,两个滚动变量不断向后递推,循环n次, 即可求出前n项

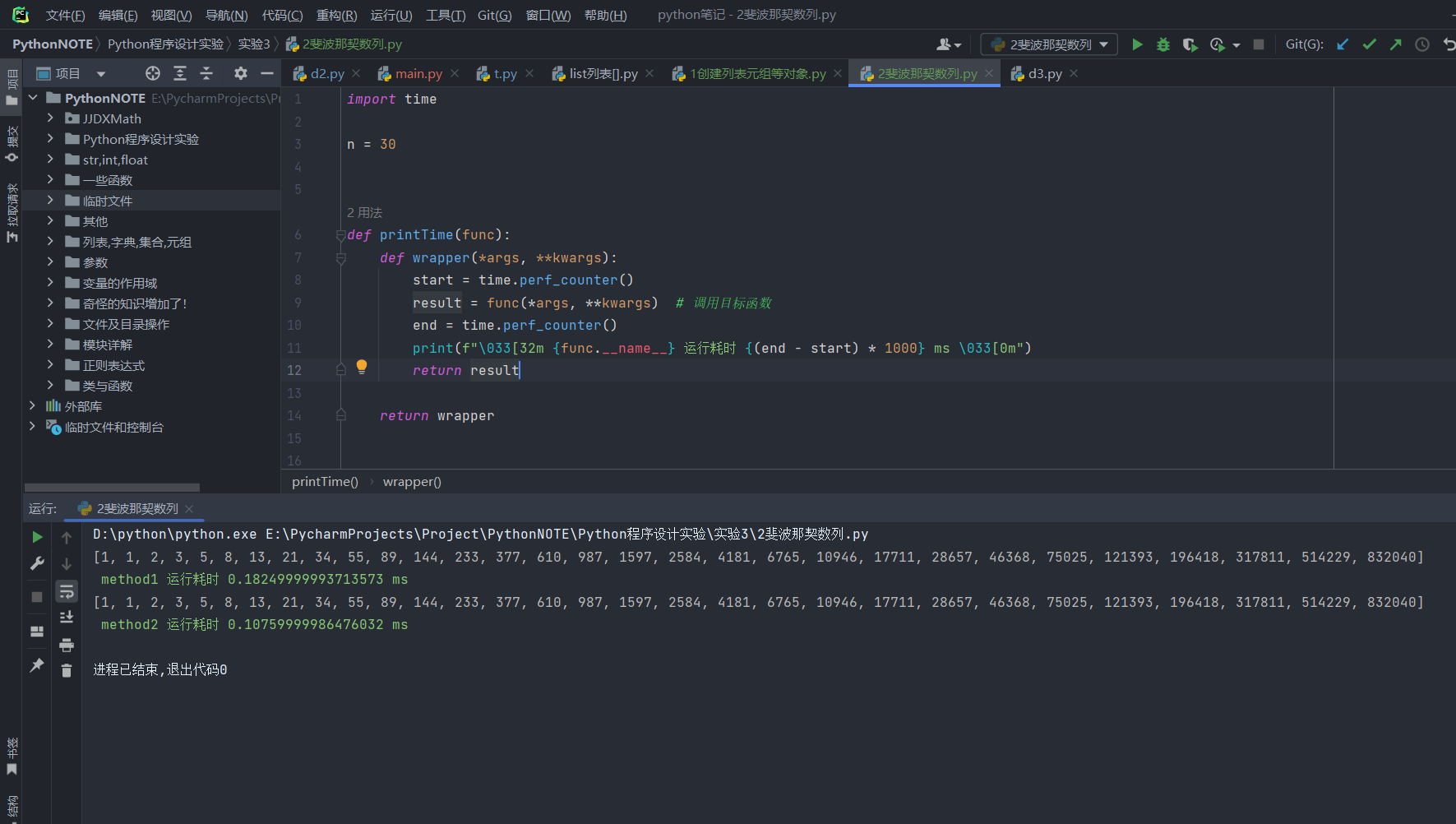
1. 实验结果

1.练习列表、元组、字符串和数字序列对象的创建方法。



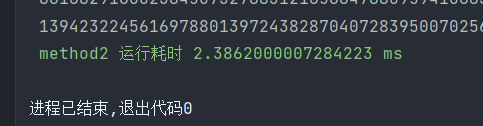
2.利用多种方法实现斐波那契数列的输出，并比较不同实现方法的性能

当n为30时, 方法1使用了0.182ms,方法二使用了0.107ms,差距并不明显



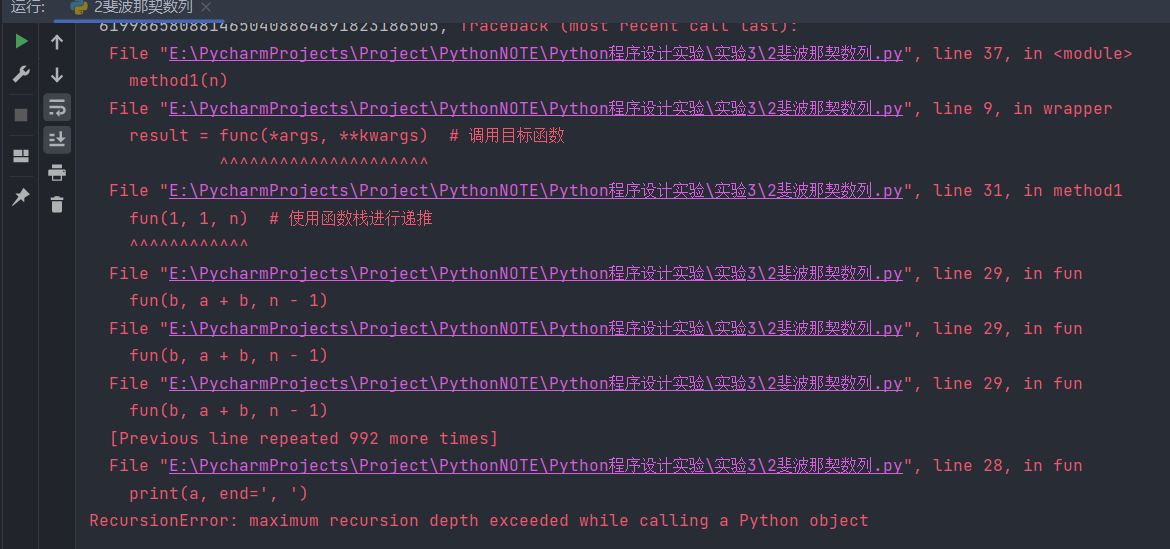
当n增大到500:





方法二比方法一快了接近5ms

当n增大到1000:



程序报错, 因为方法一超出了最大递归深度

所以, 递推会比递归更快,

1. 实验体会

对于练习列表、元组、字符串和数字序列对象的创建方法，我深刻认识到了这些数据结构在编程中的重要性。通过实际操作，我更加熟练地掌握了它们的创建和使用方法，能够根据不同的需求选择合适的数据结构。

而在实现斐波那契数列的输出过程中，我尝试了多种方法。这让我明白，在解决问题时，往往有多种途径可以选择。通过比较不同实现方法的性能，我了解到不同方法在时间复杂度和空间复杂度上的差异。这使我在今后的编程中能够更加明智地选择算法，以提高程序的效率。

此外，实验过程也让我更加熟悉了 Python 的编程风格和特点。它的简洁性和可读性让编程变得更加轻松和愉快。

总之，通过这次实验，我不仅提高了自己的编程技能，还培养了分析问题、解决问题的能力。我相信这些经验和知识将对我今后的学习和工作产生积极的影响。