实验三 列表与元组

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

一、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。

二、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

三 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。

```
import math
import random

# 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)
# 打印结果
print("列表对象创建结果:")
print(list1)
print(list2)
print(list3)
print(list4)
print("元组对象创建结果:")
print(tuple1)
```

```
print(tuple2)
print(tuple3)
print("字符串对象创建结果:")
print(s1)
print(s2)
print(s3)
print(s4)
print(s5)
print(s6)
print("数字序列对象创建结果:")
print(r1)
print(r2)
print(r3)
```

```
列表对象创建结果:
[]
[]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
元组对象创建结果:
()
(1,)
(1, 2)
字符串对象创建结果:
```

```
1
[1, 2, 3]
(1, 2)
[1, 2, 3]
数字序列对象创建结果:
range(0, 5)
range(3, 8)
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。

```
# 如果需要比较耗时,因为运行输出数字太多,耗时信息可能难以找到,所以最好关闭斐波那契数列的打印语句,只打印运行耗时
# 下一个代码块为只输出运行时间
import time

n = 10  # 太长了,这里只打印10项


def printTime(func):  # 函数装饰器,统计并打印时间
    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")  # "\033[32m"为带颜色输出
```

```
        return result

    return wrapper


@printTime
def method1(n):
    """
    打印斐波那契数列前n项
    """
    if n < 0:
        raise Exception("项数非负")
    print("[", end="")

    def fun(a, b, n):  # 递归函数
        """从a和b开始向后递推n项"""
        if n == 0:  # 递归出口
            return
        print(a, end=', ')
        fun(b, a + b, n - 1)  # f(n) = f(n-1) + f(n-2)

    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)
```

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
method1 运行耗时 0.14400000145542435 ms
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
method2 运行耗时 0.03380000271135941 ms

```
import time

# 去除斐波那契数列的打印语句,只打印运行耗时版本
```

```python
n = 2900  # 最大递归深度在2900~3000之间, 超过最大递归深度报错


def printTime(func):  # 函数装饰器,统计并打印时间
    def wrapper(*args, **kwargs):
        start = time.perf_counter()  #开始时间戳
        result = func(*args, **kwargs)  # 调用目标函数
        end = time.perf_counter()  #结束时间戳
        runTime = (end - start) * 1000  # 单位为s,转为ms
        print(f"\033[32m {func.__name__} 运行耗时 {runTime}  ms \033[0m")
        return result

    return wrapper


@printTime
def method1(n):
    """打印斐波那契数列前n项"""
    if n < 0:
        raise Exception("项数非负")

    def fun(a, b, n):
        """从a和b开始向后递推n项"""
        if n == 0:
            return
        fun(b, a + b, n - 1)

    fun(1, 1, n)  # 使用函数栈进行递推


method1(n)


@printTime
def method2(n):
    if n < 0:
        raise Exception("项数非负")
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b  # 递推


method2(n)
```

method1 运行耗时 0.8682000006956514 ms

method2 运行耗时 0.21669999841833487 ms

实验结果:
 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

通过不同的项数n,测得method1的耗时大于method2的耗时, 因为method1是通过函数栈实现的, 函数栈的出栈入栈,以及传参花费了大量时间
并且method1是递归实现,会受到函数栈最大递归深度的限制,综合来看,使用递推的方式会比递归更快更优

另外, 如果只需要求第n项, 可以使用矩阵快速幂优化dp的方式,在O(logn)的复杂度下快速求解

3. 有列表l=[54, 36, 75, 28, 50],请根据要求完成以下操作
   （1）在列表尾部插入元素42
   （2）在28前面插入元素66
   （3）删除并输出元素28
   （4）将列表按降序排列
   （5）清空整个列表。（教材93页第1题）

```
l = [54, 36, 75, 28, 50]
print("初始列表:", l)
print("（1）在列表尾部插入元素42")
l.append(42)
print("执行结果:", l)
print("（2）在28前面插入元素66")
l.insert(l.index(28), 66)
print("执行结果:", l)
print("（3）删除并输出元素28")
l.remove(28)
print("执行结果:", l)
print("（4）将列表按降序排列")
l.sort(reverse=True)
print("执行结果:", l)
print("（5）清空整个列表。")
l.clear()
print("执行结果:", l)
```

```
初始列表: [54, 36, 75, 28, 50]
（1）在列表尾部插入元素42
执行结果: [54, 36, 75, 28, 50, 42]
（2）在28前面插入元素66
执行结果: [54, 36, 75, 66, 28, 50, 42]
（3）删除并输出元素28
执行结果: [54, 36, 75, 66, 50, 42]
（4）将列表按降序排列
执行结果: [75, 66, 54, 50, 42, 36]
（5）清空整个列表。
执行结果: []
```

4. 创建长度为20的列表，其元素为1000~5000以内得随机整数。编写程序找出列表中不能被10以内素数整除的元素。（教材93页第5题）

```
def isPrime(x: int) -> bool:
    # 若 a*b=x, 且 a<=b, 则有a<=sqrt(x),b>=sqrt(x), 所以只需要遍历到sqrt(x)左右
    s = int(math.sqrt(x)) + 1
    for i in range(2, s):
        if x % i == 0: return False
    return x >= 2
```

```python
def canDiv(x: int, prime: list) -> bool:
    for p in prime:
        # if p > x: break # impossible: x∈[1000,5000], p<10
        if x % p == 0: return True
    return False


prime = [i for i in range(11) if isPrime(i)]  # 计算10以内的素数,因为数据量太小,所以没有使用素数筛
l = [random.randint(1000, 5001) for _ in range(20)]  # 生成20个随机数
print("生成的随机数:", l)

ans = [num for num in l if not canDiv(num, prime)]  # 遍历列表找出不能被10以内的素数整除的元素
# for num in l:
#     if not canDiv(num, prime):
#         ans.append(num)
print(f"不能被10以内素数整除的元素有:{ans}")
```

```
生成的随机数: [4075, 1491, 3328, 2987, 2521, 1090, 4136, 3262, 4820, 2651, 4169, 4252, 3114, 4795, 4830, 1586, 1522, 1767, 4729, 4425]
不能被10以内素数整除的元素有:[2987, 2521, 2651, 4169, 4729]
```

5.请参考例4 - 9，用嵌套的列表存储运动会报名表（见表4 - 4），并编程完成如下操作。（教材93页第6题）

(1) 求报名项目超过两项(含)的学生人数

(2) 输出女生的报名情况

(3) 输出所有报名3000m的学生姓名和性别

```python
def show(title: list[str], people: list[str | int]) -> None:
    """
    打印信息
    @:param title 标题栏
    @:param people 要打印的项
    """
    for t in title:
        print(f"{t:^8}", end='')
    print()
    for p in people:
        for i in p:
            s = i
            if i == 0:
                s = 'x'
            elif i == 1:
                s = '√'
            print(f"{s:^8}", end='')
        print()


title = ['姓名', '性别', '100m', '3000m', '跳远', '跳高']
sheet = [
```

```
    ['王平', '男', 1, 1, 0, 0],
    ['李丽', '女', 0, 1, 0, 1],
    ['陈小梅', '女', 0, 0, 1, 0],
    ['孙洪涛', '男', 0, 1, 1, 1],
    ['方亮', '男', 1, 0, 1, 0],
]
print("初始报名表:")
show(title, sheet)
print("(1)  求报名项目超过两项(含)的学生人数")
cnt = 0
for p in sheet:
    if p.count(1) >= 2:
        cnt += 1
print(f"报名项目超过两项(含)的学生人数为: {cnt}人")

print("-------------------------------------------------")
print("(2) 输出女生的报名情况")
idx = title.index('性别')
female = [p for p in sheet if p[idx] == '女']  # 筛选性别为'女'的人
print("女生的报名情况:")
show(title, female)

print("-------------------------------------------------")
print("(3)  输出所有报名3000m的学生姓名和性别")
idx = title.index('3000m')
students = [p for p in sheet if p[idx] == 1]
show(title, students)
```

```
初始报名表:
   姓名      性别      100m    3000m      跳远      跳高
  王  平      男       √       √       x       x
  李  丽      女       x       √       x       √
  陈小梅       女        x       x       √       x
  孙洪涛       男        x       √       √       √
  方  亮      男       √       x       √       x
(1)  求报名项目超过两项(含)的学生人数
报名项目超过两项(含)的学生人数为: 4人
-------------------------------------------------
(2) 输出女生的报名情况
女生的报名情况:
   姓名      性别      100m    3000m      跳远      跳高
  李  丽      女       x       √       x       √
  陈小梅       女        x       x       √       x
-------------------------------------------------
(3)  输出所有报名3000m的学生姓名和性别
   姓名      性别      100m    3000m      跳远      跳高
  王  平      男       √       √       x       x
  李  丽      女       x       √       x       √
  孙洪涛       男        x       √       √       √
```

四 实验总结

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。