深 圳 大 学 实 验 报 告

课程名称：Python 程序设计

实验项目名称：实验 4：字符串与函数式编程

学院：计算机与软件学院

专业：软件工程

指导教师：潘浩源

报告人：郑彦薇 学号：2020151022

实验时间：2022/5/1~2022/5/3

实验报告提交时间：2022/5/3

教务部制

一、实验目的

用 python 语言编写解决问题的代码并运行。

利用 python 中字符串和函数式编程的相应功能实现对问题的解决。

二、实验方法步骤

1、读题，对每个问题提出解决问题的思路

2、按照得到的思路，利用 python 语言编写解决问题的代码

3、运行代码，调试程序，直至程序可以正确输入输出

三、实验过程及内容

（一）解题思路和方法

1. 删除字符：

1) 对于该题，我有两个理解：一是在 s 中若能找到 t，t 在 s 中必须连续；二是在 s 中找到的 t 无需连续。

2) 对于第一个理解，解答过程很简单，只需借助字符串中的 find()函数即可。当 t 在 s 中可找到时，返回 t 第一次出现的位置，这个位置的最小值为 0；当 t 在 s 中不可找到时，返回-1；用一个临时变量 temp 记录 find()函数的返回值，在根据 temp 的值返回 True 或 False 即可。

3) 对于第二个理解，借助列表元素的对应查找方法。首先将输入得到的两个字符串转换为列表，并得到两个列表的长度；将列表中的元素一一对应比较，若当前位置匹配，则两个列表对应下标（即位置）同时往后移一位；若当前位置不匹配，则 s 对应列表的下标（即位置）往后移一位。若 t 对应列表已经比较到最后一个元素，则说明在 s 中可以找到 t，返回 True；若 s 对应列表已经查找到最后一位，仍然未能找到所有 t 中的元素，则说明 s 中不能找到 t，返回 False。

代码及细节解释如下：

```
1   def find1(s, t):
2       temp = s.find(t)
3       if temp == -1:
4           return False
5       return True
6   def find2(s, t):
7       s1 = list(s)
8       t1 = list(t)
9       l1 = len(s1)
10      l2 = len(t1)
11      j = 0
12      for i in range(l1):
13          if t[j] == s[i]:
14              j += 1
15          if j == l2:
16              return True
17      return False
```

对于第一种理解，直接应用 find()函数判断

比较当前对应位置，元素相同则比较下一位，不同则只是 s 往后移位

若 t 已经比较到最后一位，则说明 t 在 s 中，返回 True

```
s = input()
t = input()
if find2(s, t):
#if find1(s, t):
    print("True")
else:
    print("False")
```

2. 杨辉三角：

1) 创建一个二维列表用于存放杨辉三角中每一行的数据。根据杨辉三角第 N 行有 N 个元素，对二维列表进行扩充。
2) 将所有元素初始化为 1，每一行的第二个元素到倒数第二个元素则为上一行的相邻两个元素的和。
3) 根据最后一行中间位置的数（即杨辉三角中的最大数）的长度确定输出的元素的宽度，并计算每一行的输出的长度。
4) 使用 format()函数实现格式化输出，按照上述所求得得宽度，将每一行得最终输出存放在 res 列表中，应用 join()函数将 res 列表元素进行连接并居中输出。

代码及细节解释如下：

```
def print_yanghui_triangle(N):
    triangle = [[1]]
    for i in range(2, N+1):
        triangle.append([1] * i)
        for j in range(1, i-1):
            triangle[i-1][j] = triangle[i-2][j-1] + triangle[i-2][j]
    width = len(str(triangle[N-1][len(triangle[N-1])//2])) + 1
    column = len(triangle[N-1]) * width
    for s in triangle:
        res = []
        for c in s:
            res.append('{0:^{1}}'.format(str(c), width))
        print('{0:^{1}}'.format(''.join(res), column))

if __name__ == '__main__':
    N = int(input())
    print_yanghui_triangle(N)
```

创建二维列表

根据每一行的元素个数进行初始化

获取杨辉三角中非 1 的数

根据最大数设置元素的输出宽度以及行的输出长度

应用 format 函数进行格式化输出

3. 字符串推导式：

1) 第一个问题要输出列表字符串的首字符的大写：只需应用 upper()方法返回列表中每个字符串的第一个元素的大写字符给结果列表，然后对结果列表进行输出即可。
2) 第二个问题未给出具体要求，这里给出字符串操作规则为输出长度小于 6 的字符串：只需应用 len()函数对列表中字符串进行筛选，将符合条件的字符串返回给结果列表，然后对结果列表进行输出即可。
3) 第三个问题要求输出所有前缀为“TA”的名字：该题借助 split()方法对列表中的每个字符串按指定分隔符“_”进行分割，得到的新列表中第一个元素为“TA”，则对第二个元素返回给结果列表，然后对结果列表进行输出。
4) 第四个问题要求输出列表字符串以及字符串的长度并在输出列表中字符串及其对应长度为元组形式。该问题只需求列表中每个字符串的长度并将该长度与其对应字符

串放在一个括号里即可，将每一组结果返回给结果列表，然后对结果列表进行输出。

5) 第五个问题要求以字典形式输出列表中字符串以及它们的长度。只需将每个字符串作为键，字符串长度作为值，然后将每一个键值对返回给结果字典，对该结果字典进行输出即可。

代码及细节解释如下：

```
#(1)
List = ['apple', 'orange', 'pear']
res = [str.upper()[0] for str in List]
print(res)

#(2)
List = ['apple', 'orange', 'pear']
res = [str for str in List if len(str) < 6]
print(res)

#(3)
List = ["TA_path", "student_poohbear", "TA_michael", "TA_guido", "student_htiek"]
res = [str.split('_')[1] for str in List if str.split('_')[0] == "TA"]
print(res)

#(4)
List = ['apple', 'orange', 'pear']
res = [(str, len(str)) for str in List]
print(res)

#(5)
List = ['apple', 'orange', 'pear']
res = {str: len(str) for str in List}
print(res)
```

获得每一组结果

获得每一对键值对

4. 可读性等级：

1) 首先以空格分割文本，得到单词列表，根据单词列表的长度获得文本中的单词个数。

2) 根据句子的结尾为“.”，“？”和“!”，统计文本中这三种符号的总个数，该数则为文本中的句子个数。

3) 接下来是统计字母个数，遍历文本中的每个符号，若该符号是字母，即是 A~Z 或 a~z 中的任意一个，字母个数就加一。

4) 根据 Coleman Liau 公式，根据上述三个值计算最后 index 的值，进行判断，最后输出相应的结果即可。

代码及细节解释如下：

```python
def readability(str):
    word_num = len(str.split(' '))
    sent_num = str.count('.') + str.count('!') + str.count('?')
    letter_num = 0
    for i in str:
        if (i <= 'Z' and i >= 'A') or (i <= 'z' and i >= 'a'):
            letter_num += 1
    L = letter_num/word_num*100
    S = sent_num/word_num*100
    index = round(0.0588 * L - 0.296 * S -15.8)
    if index < 1:
        print("Before Grade 1")
    elif index>= 16:
        print("Grade 16+")
    else:
        print(f'Grade {index}')

if __name__ == '__main__':
    str = input()
    readability(str)
```

获得文本中的字母数、单词数和句子数

根据公式计算文本的可读性等级

5. 函数的参数传递：

1) 语句 sum(*(1, 2, 3))合法，三个参数都通过指针实现了可变，并作为 a，b，c 值传入函数，得到正确结果。
2) 语句 sum(1,*(2,3))合法，第一个参数直接传参为 1，后面两个值通过指针实现可变，并作为 b，c 值传入函数，得到正确结果。
3) 语句 sum(*(1,),b=2, 3)非法，在编程中输入该语句时会报错，关键字参数必须跟随在位置参数后面，如果使用关键字赋值，那所有参数都必须赋值，这里只赋值一个，因此该语句非法。
4) 语句 sum(*(1,),b=2, c=3)合法，该语句修改了上述语句的错误，对关键字都进行了赋值，可以得到正确结果。
5) 语句 sum(*(1, 2),c=3)合法，1，2 通过指针实现可变，作为可变数量参数传入函数，而 c=3 则是使用关键字赋值，所有参数都进行了赋值，可以得到正确结果。
6) 语句 sum(a=1, *(2, 3))非法，在编程中输入该语句运行会报错，参数 a 被多次赋值。因为在 a=1 时已经使用关键字参数对 a 进行了赋值，*(2,3)又通过指针实现可变，对参数再次进行了赋值，从而导致了错误。
7) 语句 sum(b=1, *(2, 3))非法，错误与上述语句错误相同。
8) 语句 sum(c=1, *(2, 3))合法，因为通过关键字参数进行赋值的是函数参数中的最后一位，而通过指针实现可变，对参数进行赋值只赋值了前两个参数，不会冲突，因此该语句可以得到正确结果。

6. Lambda：

1) 运行该语句得到输出为 1.
   该语句是传入的 val 值对 2 进行取余，故 5 对 2 取余得到结果 1.
   测试：输入 6，输出 0；输入 7，输出 1；输入 8，输出 0.
2) 运行该语句得到输出为 11.
   该语句是对传入的两个参数 x 和 y 进行按位异或，并输出结果（为十进制）。
   测试：输入（3，5），输出 6；输入（1，2）输出 3.
3) 运行该语句得到输出为 yth。

该语句首先是将字符串前面的空格丢弃，再将大写转换为小写，最后输出从下标为1开始从前往后连续三个字符。

测试：输入“happy”，输出“app”；输入“　　to be or not to be”，输出“o d”。

代码如下：

```
print((lambda val: val % 2)(5))
print((lambda x, y: x ^ y)(3, 8))
print((lambda s: s.strip().lower()[1:4])(' PyTHon'))
```

7. Map：

1) 第一个问题是将列表中的字符转换为整型即可。对于列表中的元素，使用 lambda 实现元素转换为整型，再使用 map 将该功能应用于整个列表，最终得到一个对应的整型列表，将该列表进行输出即可。
2) 第二个问题是获得列表中每个字符串的长度。该问题同样首先使用 lambda 得到一个函数对象为字符串的长度，再使用 map 将该功能应用于整个列表，最终得到一个对应的长度列表，将该列表进行输出即可。
3) 第三个问题是将列表中的字符串进行倒序输出。首先使用 lambda 操作对象，利用切片功能获得倒序的字符串，再使用 map 将该功能应用于整个列表，将最终得到的反转列表进行输出即可。
4) 第四个问题是按元组形式输出 2~6（不包含 6）的所有数本身及其平方和三次方。同样使用 lambda 获得函数对象为一个包含三个数的元组，这三个数分别为数本身、数的平方、数的三次方。再使用 map 将该功能应用于 2~6（不包含 6）的所有数。将得到的结果进行输出即可。
5) 第五个问题是将 zip 的每一对结果中的两个数进行相乘。仍然是使用 lambda 获得函数对象为元组中两个元素相乘的结果，再使用 map 函数将其应用于列表中的每一个元组，并将最终结果列表进行输出。

代码及细节解释如下：

```
#(1)
print(list(map(lambda x: int(x), ['12', '-2', '0'])))

#(2)
print(list(map(lambda x: len(x), ['hello', 'world'])))

#(3)
print(list(map(lambda x: x[::-1], ['hello', 'world'])))

#(4)
print(list(map(lambda x: (x, x**2, x**3), range(2, 6))))

#(5)
print(list(map(lambda x: x[0] * x[1], zip(range(2, 5), range(3, 9, 2)))))
```

8. Filter：

1) 第一个问题是筛选出列表中大于 0 的数。使用 lambda 获得函数对象，设置筛选条件为该对象的值大于‘0’，并输入操作列表，将最终筛选所得结果列表进行输出即可。
2) 第二个问题是筛选出列表中以字母 w 开头的字符串。使用 lambda 获得函数对象，设置筛选条件为字符串的首位即下标为 0 的字母为‘w’，并输入操作列表，将筛选所

得的结果列表进行输出即可。

3) 第三个问题可以看成是筛选出列表中前缀为“tech”的单词。通过切片功能获取字符串的前四位，将筛选条件设置为切片所得字符串等于“tech”即可。

4) 第四个问题为找到0~20（不包括20）中为3或5的倍数的数。只需将筛选条件设置为函数对象对3或5取余结果为0，然后将获得的结果列表进行输出即可。

代码及细节解释如下：

```
#(1)
print(list(filter(lambda x: x > '0', ['12', '-1', '0'])))

#(2)
print(list(filter(lambda x: x[0] == 'w', ['hello', 'world'])))

#(3)
print(list(filter(lambda x: x[0:4:1] == 'tech', ['technology', 'method', 'technique'])))

#(4)
print(list(filter(lambda x: x % 3 == 0 or x % 5 == 0, range(0, 20))))
```

9. Reduce 1：

1) 第一个问题是输出列表中的最大值。将reduce中的函数设置为max，并输入操作列表，将最终结果进行输出即可。

2) 第二个问题是将列表中的所有字符串进行拼接。首先使用lambda获得一个函数对象，这个函数对象为两个参数的和，在reduce中写入该功能，输入的列表进行操作，并将结果进行输出即可。

3) 第三个问题是将列表中的所有数进行相乘，首先使用lambda获得一个函数对象，为两个参数的积，在reduce中写入该功能，对输入的列表进行操作，并将结果进行输出即可。

代码及细节解释如下：

```
from functools import reduce

#(1)
print(reduce(max, [23, 49, 6, 32]))

#(2)
print(reduce(lambda x, y: x+y, ['foo', 'bar', 'baz', 'quz']))

#(3)
print(reduce(lambda x, y: x*y, [2, 4, 6]))
```

在 python3 中需要 import

10. Reduce 2：

1) 调用gcd函数计算相邻两个数的最小公约数，并用两个数的乘积除以最小公约数，用此方法可以得到两数的最小公倍数。

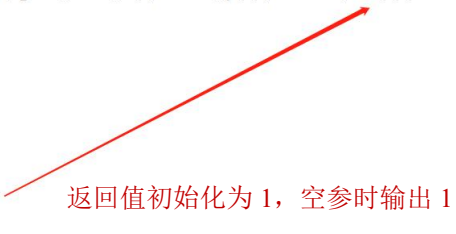
2) 使用reduce方法处理整个传入函数的多个数，得到这些数的最小公倍数。

3) 将结果转化为整型并输出即可。

代码及细节解释如下：

```python
from math import gcd
from functools import reduce
def lcm(*nums):
    print(int(reduce(lambda x, y: x*y/gcd(int(x), int(y)), nums, 1)))

if __name__ == '__main__':
    lcm(3, 5)
    lcm(41, 106, 12)
    lcm(1, 2, 6, 24, 120, 720)
    lcm(3)
    lcm()
```



11. Iterator：

1) 该题首先是定义了 it 为 0~100 范围内的迭代器。运行 66 in it 后迭代器位置在 66 处。
2) 运行 print(next(it))得到输出为 67，因为进行了 66 in it 的运行之后，迭代器在 66 位置，因此运行输出下一位可得到输出为 67.
3) 运行 print(33 in it)得到输出 False，因为上述语句运行后，迭代器在 67 且迭代器不能向前移动，故迭代器一直移动到最后一位，仍未找到 33，因此得到输出位 False。
4) 运行 print(next(it))引发 StopIteration，因为上述语句运行后，再运行该语句迭代器终止，引发 StopIteration 错误。

代码如下：

```python
it = iter(range(100))
print(66 in it)
print(next(it))
print(33 in it)
print(next(it))
```

12. Generator：

1) 编写生成器：初始化三角数为 0，加数为 1，在每一趟循环中，三角数的值都为上一个三角数的值加上加数，每一个加数都为上一个加数的值加 1，创建一个无限的序列，序列元素为三角数。
2) 编写函数：传入参数为界限，遍历无限序列中的数，如果数小于界限，则输出该数，若三角数超过界限，则结束循环。

代码及细节解释如下：

```python
def generate_triangles():
    sum, num = 0, 1
    while True:
        sum, num = sum + num, num + 1
        yield sum
def generate_triangle_under(N):
    for i in generate_triangles():
        if i > N:
            break
        print(i, end=' ')
if __name__ == '__main__':
    N = int(input())
    generate_triangle_under(N)
```

更新三角数和加数，获得一个三角数的无限序列

若三角数超过界限，退出循环

将输入转换为整数

13. Decorator:

1) 对传入的函数进行修改，在执行函数本身的功能之前，使用 time 函数记录开始的时间，在执行函数本身功能之后，再次使用 time 函数记录结束的时间，将这两个值相减乘以 1000，则可以得到函数执行是所用时间（以 ms 计）。

2) 上述添加了计时功能的函数即为对原先函数进行修改的结果，输出时间后返回修改后的函数。实现不改变函数参数内部语句的情况下添加计时功能。

代码及细节解释如下：

```python
from time import time
def timed(fn):
    def time_fn():
        start = time()
        fn()
        end = time()
        count = (end - start)*1000
        print(f'time is {count} ms')
    return time_fn
@timed
def test_plus():
    result = []
    for i in range(100000):
        result = result + [i]
@timed
def test_append():
    result = []
    for i in range(100000):
        result.append(i)

test_plus()
test_append()
```

修改函数，加入计时功能

返回修改后的函数

在函数声明时应用装饰器

函数本身代码未修改

（二）遇到的问题和收获

在解决第 10 个问题 lcm 函数的编写时，编写完成后尝试在主函数中对 lcm 操作的任意数量个整数进行输入，但发现无论是转换为列表还是元组或是使用 lambda 都无法正常

运行，最终直接给出参数才能得到正确结果。

在解决可读性等级这个问题时，对句子数量的统计，一开始尝试对文本按照句子结尾符号进行分割，然后统计分割后字符串数从而实现对句子数量的统计，但实际操作后发现这样会重复计数，得不到正确结果。最后通过统计三种句子结束符的个数实现对句子数量的统计。

（三）代码运行结果展示

1. 删除字符：

对于第一种理解，即要求 t 在 s 中连续：

```
whenever
never
True
```

```
shenzhen
sz
False
```

对于第二种理解，即对 t 在 s 中不要求连续：

```
whenever
never
True
```

```
shenzhen
sz
True
```

```
shenzhen
zheng
False
```

2. 杨辉三角：

输出前 5 行：

```
5
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

3. 字符串推导式：

5 个问题同时运行，得到对应的 5 个结果：

```
['A', 'O', 'P']
['apple', 'pear']
['path', 'michael', 'guido']
[('apple', 5), ('orange', 6), ('pear', 4)]
{'apple': 5, 'orange': 6, 'pear': 4}
```

4. 可读性等级：

等级小于 1：

```
One fish. Two fish. Red fish. Blue fish.
Before Grade 1
```

等级大于或等于 16:

```
A large class of computational problems involve the determination of properties of
 graphs, digraphs, integers, arrays of integers, finite families of finite sets,
 boolean formulas and elements of other countable domains.
Grade 16+
```

其他情况:

```
It was a bright cold day in April, and the clocks were striking thirteen. Winston
 Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped
 quickly through the glass doors of Victory Mansions, though not quickly enough to
 prevent a swirl of gritty dust from entering along with him.
Grade 10

There are more things in Heaven and Earth, Horatio, than are dreamt of in your
 philosophy.
Grade 9
```

5. 函数的参数传递：

合法语句的运行结果如下:

```
1  def sum(a, b, c):
2      print("a=%d, b=%d, c=%d" % (a,b , c))
3      print(a+b+c)
4
5  if __name__ == '__main__':
6      sum(*(1, 2, 3))
7      sum(1, *(2, 3))
8      sum(*(1,),b=2, c=3)
9      sum(*(1, 2), c=3)
10     sum(c=1, *(2, 3))
11
```

test4_1 × test4_5 ×

```
C:\Users\4334\PycharmProjects\pythonProject_Test4\venv\Scripts\python.exe
a=1, b=2, c=3
6
a=1, b=2, c=3
6
a=1, b=2, c=3
6
a=1, b=2, c=3
6
a=2, b=3, c=1
6
```

6. **Lambda:**

对代码参数进行更改，得到测试例子运行结果如下：

1) (lambda val: val % 2)(5):

将 5 更改为 6、7、8

```
print((lambda val: val % 2)(6))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythonP
0
```

```
print((lambda val: val % 2)(7))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythor
1
```

```
print((lambda val: val % 2)(8))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythonPr
0
```

2) (lambda x, y: x ^ y)(3, 8)

将(3, 8)更改为(3, 5)、(1, 2)

```
print((lambda x, y: x ^ y)(3, 5))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythonProj
6
```

```
print((lambda x, y: x ^ y)(1, 2))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythonPr
3
```

3) (lambda s: s.strip().lower()[1:4])(' PyTHon')

将' PyTHon'更改为'happy'、' to be or not to be'

```
print((lambda s: s.strip().lower()[1:4])('happy'))
```

```
test4_1 × test4_6 ×
C:\Users\4334\PycharmProjects\pythonProject_Test4\venv\Sc
app
```

```
3  print((lambda s: s.strip().lower()[1:4])('    to be or not to be'))
```

test4_1 × test4_6 ×

```
C:\Users\4334\PycharmProjects\pythonProject_Test4\venv\Scripts\python.exe
o b
```

7. Map：

5 个问题同时运行，得到对应的 5 个结果：

```
[12, -2, 0]
[5, 5]
['olleh', 'dlrow']
[(2, 4, 8), (3, 9, 27), (4, 16, 64), (5, 25, 125)]
[6, 15, 28]
```

8. Filter：

同时运行 4 个语句，得到对应的 4 个结果：

```
['12']
['world']
['technology', 'technique']
[0, 3, 5, 6, 9, 10, 12, 15, 18]
```

9. Reduce 1：

同时运行 3 个语句，得到对应的 3 个结果：

```
49
foobarbazquz
48
```

10. Reduce 2：

运行给出的 5 个例子，得到输出为：

```
15
26076
720
3
1
```

11. Iterator：

对题目给出的语句进行运行：

```
    print(next(it))
StopIteration
True
67
False
```

12. Generator:

输入 20:

```
20
1 3 6 10 15
```

输入 21:

```
21
1 3 6 10 15 21
```

13. Decorator:

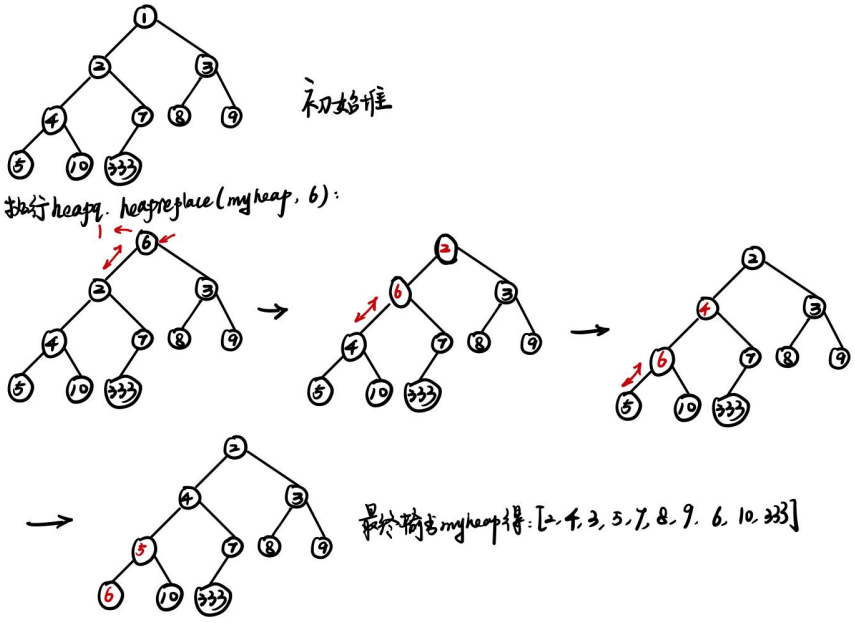
```
time is 15235.788106918335 ms
time is 6.001472473144531 ms
```

四、实验总结

1、该实验主要是字符串的功能以及函数式编程中的功能实现对问题的解决。字符串自带的 split()、len()等函数给处理文本带来极大的便利，format()函数可以更好的实现某些特定的输出格式。

2、Lambda()、map()等函数实现只需通过一句 python 语句便可以解决问题，生成器装饰器等的定义也使得对函数的应用或修改更加方便。

附：实验 3.12 堆的变化【补充】:



指导教师批阅意见：   **成绩评定：** **指导教师签字：** **年　　月　　日**
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。