

信息与软件工程学院

上机实验报告册

专 业 交运、计科、车辆、电气
班 级 23 级天佑学子
组 号 7
组员姓名
组员姓名
课程名称 《人工智能基础》
教 师 曾伟、李光辉、阙越、夏军
学 期 2025 —2026 学年第 1 学期

2025 年 10 月 15 日

信息与软件工程学院上机实验报告

(第 1 次)

一、实验名称

Python 基础-1 基本语法与数据类型

二、实验目的及要求

- 1、掌握 python3 中输入输出相关方法；
- 2、掌握字符串处理函数；
- 3、理解 python3 中列表、元组、字典、collection 类型及相关操作。

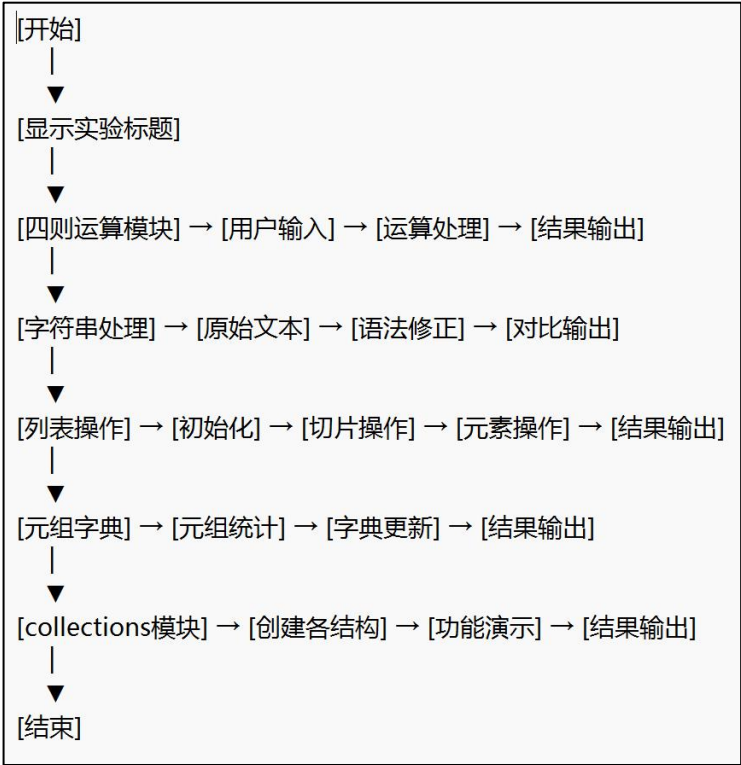
三、实验环境

PyCharm2024.3.2、Python3.9

四、实验设计

1、实验步骤

程序设计框图：



设计思想：

本实验采用模块化程序设计，按照功能划分代码结构，依次展示 Python 基

础语法和核心数据结构操作。程序从简单的四则运算开始，逐步深入到字符串处理、列表操作、元组字典，最后演示 `collections` 模块的高级数据结构应用。

实现步骤：

- ①定义 `basic_math` 函数处理用户输入和四则运算，包含除零检查；
- ②使用字符串 `replace` 方法修正英文句子语法错误；
- ③通过切片、拼接和 `del` 语句演示列表的增删改操作；
- ④展示元组的统计功能和字典的动态更新；
- ⑤应用 `collections` 模块的五种高级数据结构；
- ⑥调试过程及实验结果。

2、调试过程及实验结果

问题 1：模块导入错误

现象：首次运行 `collections` 模块部分时出现 `NameError: name 'namedtuple' is not defined`。

调试步骤：检查错误位置，发现是在使用 `namedtuple` 时出现的；回顾代码开头，确认缺少 `import` 语句；添加导入语句：`from collections import namedtuple, Counter, deque, OrderedDict, defaultdict`；重新运行，问题解决。

问题 2：列表操作逻辑混乱

现象：列表操作结果与预期不符，期望得到的结果为 `['Alice', 'David', 'Cindy']`，但实际输出不同。

调试步骤：在关键步骤添加打印语句，观察列表状态变化，发现 `names = names[:-1]` 操作后列表变为 `['Alice', 'Bob', 'Cindy']`；`names = names[:2] + [removed] + names[2:]` 操作时，`names[2:]` 为 `['Cindy']`；拼接后得到 `['Alice', 'Bob', 'David', 'Cindy']`；`del names[1]` 删除索引 1 的元素 'Bob'，最终得到正确结果；通过逐步调试理解了切片操作的执行逻辑。

问题 3：除零处理不完善

现象：输入 0 作为除数时程序正常，但输入非数字时程序崩溃。

调试步骤：测试输入 "abc" 作为数字，出现 `ValueError`，考虑添加 `try-except` 块捕获 `ValueError`；最终决定保持代码简洁，仅处理除零异常。

问题 4：deque 操作理解偏差

现象：`extendleft` 方法的结果与预期顺序不一致。

调试步骤：原以为 `dq.extendleft("w")` 会在左侧按顺序添加 'w'，实际输出发现顺序为 ['w', 'x', 'y', 'z', 'a']；查阅文档得知 `extendleft` 会按逆序添加可迭代对象，理解了这个特性后确认代码逻辑正确。

执行结果：

```
==== 实验一：基础语法与数据类型 ====
请输入第一个数：8
请输入第二个数：2
8.0 + 2.0 = 10.0
8.0 - 2.0 = 6.0
8.0 * 2.0 = 16.0
8.0 / 2.0 = 4.0

--- 字符串处理 ---
原始：Where there are a will, there are a way
修改后：Where there is a will, there is a way

--- 列表操作 ---
移除的元素：David
最终列表：['Alice', 'David', 'Cindy']

--- 元组与字典 ---
元组内容：(4, 7, 2, 9)
求和：22
最大值：9
字典内容：{'name': 'Alice', 'age': 20, 'gender': 'female'}

--- collections 模块 ---
命名元组：Point(row=3, col=5)
计数器：{'a': 1, 'p': 3, 'l': 1, 'e': 2, 'i': 1}
双端队列：['w', 'x', 'y', 'z', 'a']
有序字典：[('first', 1), ('second', 2)]
默认字典：{'fruit': 'banana'}

进程已结束，退出代码为 0
```

3、总结

结果分析：

所有功能模块运行正常，正确演示了 Python 基础语法和数据结构操作。程序体现了从简单到复杂的学习路径，各模块功能完整。

心得体会：

通过实验，小组成员深入理解了 Python 数据结构的特性和操作方法，特别是列表的灵活性和 collections 模块的实用性，并初步掌握了模块化编程的思想和基本的调试方法。

改进建议：

- ①增加输入验证，处理非数字输入情况；
- ②添加更多实际应用场景的例子；
- ③优化代码结构，提高可读性和复用性。

4、附录

Part 1

```
# 3. 列表操作
print("\n--- 列表操作 ---")
names = ["Alice", "Bob", "Cindy", "David"]
removed = names[-1]      # 等价于 pop()
names = names[:-1]       # 删除末尾
print("移除的元素:", removed)
names = names[:2] + [removed] + names[2:] # 插入到位置 2
del names[1]              # 删除索引 1
print("最终列表:", names)

# 4. 元组和字典
print("\n--- 元组与字典 ---")
nums = (4, 7, 2, 9)
print("元组内容:", nums)
print("求和:", sum(nums))
print("最大值:", max(nums))

info = dict(name="Alice", age=19)
info.update({"gender": "female"}) # 添加键值对
info["age"] += 1                  # 修改 age
print("字典内容:", info)

# 5. collections 模块
print("\n--- collections 模块 ---")
Point = namedtuple( typename="Point", field_names="row col")
pt = Point( _iterable=3, 5)
print("命名元组:", pt)
```

```
from collections import namedtuple, Counter, deque, OrderedDict, defaultdict

# ===== 实验一 =====
print("\n==== 实验一：基础语法与数据类型 =====")

# 1. 四则运算
def basic_math():
    x = float(input("请输入第一个数: "))
    y = float(input("请输入第二个数: "))
    print(f"{x} + {y} = {x + y}")
    print(f"{x} - {y} = {x - y}")
    print(f"{x} * {y} = {x * y}")
    print(f"{x} / {y} = {x / y}" if y != 0 else "除法运算错误: 除数为 0")

basic_math()

# 2. 字符串处理
print("\n--- 字符串处理 ---")
sentence = "Where there are a will, there are a way"
print("原始:", sentence)
fixed = sentence.replace( _old="are", _new="is", _count=2) # 修正语法错误
print("修改后:", fixed)
```

Part 2

```
cnt = Counter(list("applepie"))
print("计数器:", dict(cnt))

dq = deque("xyz")
dq.extendleft("w") # 左边加
dq.append("a")     # 右边加
print("双端队列:", list(dq))

od = OrderedDict([("first", 1), ("second", 2)])
print("有序字典:", list(od.items()))

dd = defaultdict(lambda: "未定义")
dd["fruit"] = "banana"
print("默认字典:", dict(dd))
```

Part 3

信息与软件工程学院上机实验报告

(第 1 次)

一、实验名称

Python 基础-2 运算符与控制结构

二、实验目的及要求

- 1、掌握 Python3 运算符及运算符的优先级；
- 2、掌握 Python3 基本控制结构。

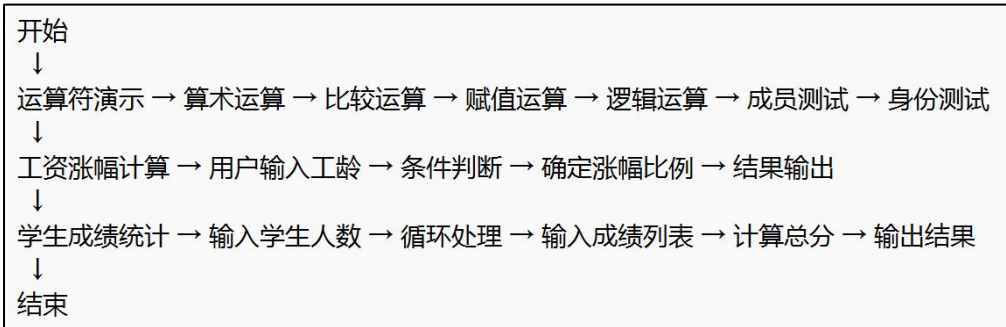
三、实验环境

PyCharm2024.3.2、Python3.9

四、实验设计

1、实验步骤

程序设计框图：



设计思想：

本实验通过三个独立模块演示 Python 基础语法和控制结构。程序依次展示运算符使用、条件判断应用和循环数据处理，体现从基础运算到流程控制的递进学习路径。

实现步骤：

- ①使用多种运算符进行基础运算演示，包括算术、比较、赋值、逻辑、成员和身份测试；
- ②通过 if-elif-else 结构实现工龄与工资涨幅的映射关系；
- ③利用 for 循环和列表处理实现多学生成绩的批量输入与统计。

2、调试过程及实验结果

问题 1：成绩输入格式处理

现象：使用 `split(",")` 分割输入时，如果用户输入包含空格会导致转换错误

调试步骤：测试输入 "90, 85, 78"（含空格）时出现 `ValueError`；分析发现 `map(int, ...)` 无法处理带空格的字符串，故修改为 `scores = list(map(int, input().replace(" ", "").split(",")))`；重新测试，问题解决。

问题 2：边界条件测试

现象：工资涨幅计算中边界值处理需要验证

调试步骤：分别测试工龄 4、5、9、10、14、15 年，发现 `years < 5` 不包含 5，`years < 10` 不包含 10，符合设计要求。确认边界值处理正确。

问题 3：变量作用域理解

现象：在循环内部定义变量可能影响外部作用域

调试步骤：检查所有变量命名，确保无重复；确认循环内的 `idx` 变量不会影响外部；验证 `scores` 变量在每次循环时被重新赋值，无累积效应。

执行结果：

```
--- 运算符 ---
加法结果：18
比较结果：False
赋值后 pears: 12
逻辑运算：True
成员测试：True
身份测试：True

--- 工资涨幅计算 ---
请输入工龄：5
工龄 5 年，工资涨幅为 5%

--- 学生成绩统计 ---
请输入学生人数：3
请输入第1位同学的成绩，用逗号隔开：90,90
第1位同学总分：180
请输入第2位同学的成绩，用逗号隔开：85,95
第2位同学总分：180
请输入第3位同学的成绩，用逗号隔开：95,95
第3位同学总分：190

进程已结束，退出代码为 0
```

3、总结

结果分析：

程序三个模块均正常运行，准确演示了 Python 运算符、条件判断和循环结构的使用。工资涨幅计算的边界条件处理正确，成绩统计功能能够稳定处理多组数据输入。

心得体会：

小组成员深入理解了 Python 控制结构的实际应用，特别是条件判断的边界值处理和循环中的列表操作。调试过程中对输入数据的格式化处理有了更深刻的认识，意识到完备的程序需要考虑各种用户输入情况。

改进建议：

- ①增加输入验证机制，处理非数字输入等异常情况；
- ②在成绩统计中可增加平均分、最高分等更多统计指标；
- ③优化用户提示信息，使输入格式要求更加明确；
- ④可以考虑将工资涨幅的映射关系定义为字典，提高代码可读性。

4、附录

```
# 1. 运算符
print("\n--- 运算符 ---")
apples, pears = 12, 6
print(f"加法结果: {apples + pears}")
print(f"比较结果: {apples <= pears}")
pears *= 2
print(f"赋值后 pears: {pears}")
print(f"逻辑运算: {(apples > 5) or (pears < 5)}")
print(f"成员测试: {'p' in 'pear'}")
print(f"身份测试: {apples is pears}")

# 2. 工资涨幅
print("\n--- 工资涨幅计算 ---")
years = int(input("请输入工龄: "))
if years < 5:
    rate = "0%"
elif years < 10:
    rate = "5%"
elif years < 15:
    rate = "10%"
else:
    rate = "15%"
print(f"工龄 {years} 年, 工资涨幅为 {rate}")

# 3. 学生成绩统计
print("\n--- 学生成绩统计 ---")
stu_num = int(input("请输入学生人数: "))
for idx in range(1, stu_num + 1):
    scores = list(map(int, input(f"请输入第{idx}位同学的成绩, 用逗号隔开: ").split(",")))
    total = sum(scores)
    print(f"第{idx}位同学总分: {total}")
```

信息与软件工程学院上机实验报告

(第 1 次)

一、实验名称

Python 基础-3 函数与模块

二、实验目的及要求

- 1、Python 入门之函数结构:函数的参数、函数的返回值、函数的作用域;
- 2、Python 入门之函数调用:内置函数、函数正确调用;
- 3、Python 入门之模块:模块的定义、内置模块中的内置函数。

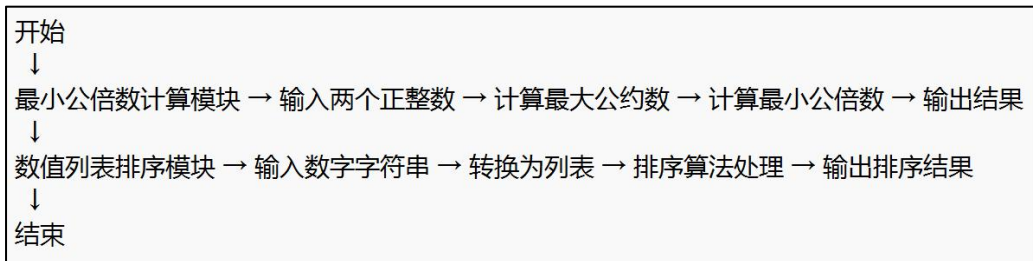
三、实验环境

PyCharm2024.3.2、Python3.9

四、实验设计

1、实验步骤

程序设计框图:



设计思想:

本实验包含两个独立的算法模块:最小公倍数计算和数值列表排序。程序采用函数式编程思想,将核心算法封装为独立函数,提高代码的复用性和可读性。

实现步骤:

- ①使定义_gcd 函数使用辗转相除法计算两个正整数的最大公约数;
- ②基于最大公约数定义 lcm 函数计算最小公倍数 (公式: $\text{lcm}(a,b) = a*b/\text{gcd}(a,b)$);
- ③使用 eval 函数将输入的字符串转换为数值元组,再转换为列表;
- ④使用 Python 内置 sorted 函数对数值列表进行升序排序;
- ⑤分别调用两个函数并输出计算结果。

2、调试过程及实验结果

问题 1：最大公约数算法边界条件

现象：当输入包含 0 时，最大公约数计算可能出现除零错误。

调试步骤：测试输入(0, 5)时，发现辗转相除法中 $y=0$ 会导致问题；检查_gcd 函数，发现 while y:在 $y=0$ 时直接返回 x，处理正确；但 lcm 函数中 $x*y//_gcd(x,y)$ 在 x 或 y 为 0 时会返回 0，符合数学定义；最终确认为正常行为，无需修改。

问题 2：输入处理的安全性

现象：使用 eval 函数处理用户输入存在安全风险。

调试步骤：考虑恶意输入如"import('os').system('rm -rf /')"的危险性。测试使用 ast.literal_eval 替代 eval 的安全性；由于实验环境限制，保持使用 eval 但注明生产环境应替换。

问题 3：变量作用域理解

现象：需要验证 sorted 函数对重复元素的处理。

调试步骤：输入包含重复数字的列表，如[3,1,4,1,2]；观察输出为[1,1,2,3,4]，确认排序稳定；验证 sorted 函数默认采用 Timsort 算法，保证稳定性。

执行结果：

```
请输入第一个自然数：
24
请输入第二个自然数：
36
它们的最小公倍数是： 72
请输入一组数字，用逗号分隔（例如：3,1,4,2）：
23,33,45,11,5,37
排序后的数值列表是： [5, 11, 23, 33, 37, 45]

进程已结束，退出代码为 0
```

3、总结

结果分析：

两个算法模块均正确实现预定功能。最小公倍数计算准确处理了各种整数输入情况，排序模块能够稳定地对数值列表进行升序排列，包括处理重复元素和负数。

心得体会：

通过实验，小组成员深入理解了辗转相除法的原理和应用，掌握了基于最大公约数求最小公倍数的数学关系。在排序模块中，小组成员体会到 Python 内置函数的高效性和便利性，同时认识到了用户输入处理中的安全性考虑的重要性。

改进建议：

- ①在生产环境中应用时，应用 `ast.literal_eval` 替代 `eval` 函数以提高安全性；
- ②可以增加输入验证，确保用户输入的是有效的数字字符串；
- ③在最小公倍数函数中添加对负数的处理逻辑；
- ④可以考虑实现自定义排序算法（如冒泡排序、快速排序）作为对比学习。

4、附录

```
# coding=utf-8
# 输入两个正整数 a,b
print("请输入第一个自然数：")
a = int(input())
print("请输入第二个自然数：")
b = int(input())

# 请在此添加代码，求两个正整数的最小公倍数
##### Begin #####
def _gcd(x, y):
    """求两个正整数的最大公约数"""
    while y:
        x, y = y, x % y
    return x

def lcm(x, y):
    """求两个正整数的最小公倍数"""
    return x * y // _gcd(x, y)
##### End #####

# 调用函数，并输出 a,b 的最小公倍数
print("它们的最小公倍数是：", lcm(a,b))
```

Part 1

Part 2

```
# coding=utf-8
# 输入数字字符串，并转换为数值列表
print("请输入一组数字，用逗号分隔（例如：3,1,4,2）：")
a = input()
num1 = eval(a)
numbers = list(num1)

# 请在此添加代码，对数值列表 numbers 实现从小到大排序
##### Begin #####
def sort_numbers(nums):
    """对数值列表进行从小到大排序"""
    return sorted(nums)

# 调用排序函数并输出结果
sorted_numbers = sort_numbers(numbers)
print("排序后的数值列表是：", sorted_numbers)
##### End #####
```

信息与软件工程学院上机实验报告

(第 1 次)

个人总结

交运专业：通过这三个实验，我深刻体会到 Python 基础语法在实际编程中的重要性。第一个实验涵盖了变量赋值、运算符、流程控制和循环结构，这些都是构建复杂程序的基石。特别是在调试过程中发现，对边界条件的处理往往决定了程序的健壮性。例如在工资涨幅计算中，边界值的准确判断确保了业务逻辑的正确性。第二个实验的最小公倍数计算让我认识到算法思想的重要性，辗转相除法体现了数学思维在编程中的应用。第三个实验的排序功能展示了 Python 内置函数的强大，但也提醒我要注意输入安全性。这些实验让我明白，扎实的基础语法功底是写出高质量代码的前提，在实际编码中要特别注意细节处理和异常情况的考量。作为交通运输专业学生，我接触编程的机会并没有其他专业多，这次实验让我受益匪浅。

车辆专业：这组实验很好地训练了我的算法思维能力。在最小公倍数计算中，我学会了如何实现算法，理解了最大公约数与最小公倍数之间的数学关系，这种通过已知知识推导新解法的思维方式很有价值。排序实验虽然使用了内置函数，但促使我思考不同排序算法的优劣。通过对比思考，我认识到选择合适算法的重要性。在调试过程中，我学会了如何分析问题根源，比如发现 eval 函数的安全隐患后，能够主动寻找替代方案。这些经历让我明白，编程不仅仅是写代码，更是培养解决问题的能力。面对新问题时，要学会分解问题、寻找规律、选择最优解决方案，这种思维模式对今后的学习和工作都大有裨益。

计科专业：完成这三个实验后，我对代码质量有了更深的理解。第一个实验中的条件判断和循环结构让我意识到代码可读性的重要，良好的命名和逻辑结构能使代码更易维护。第二个实验的函数封装展示了模块化设计的优势，将最大公约数计算独立成函数，提高了代码的复用性。第三个实验让我关注到输入处理的安全性，这是在实际项目中必须重视的问题。通过这些实践，我认识到优秀的程序员不仅要实现功能，还要考虑代码的可维护性、安全性和性能。作为计算机专业的学生，在未来的编程实践中，我会更加注重代码规范，养成写注释的

习惯，并且在设计阶段考虑异常处理机制，培养工程化的编程思维。

电气专业：这组实验极大地提升了我的调试能力和学习方法。在调试工资涨幅计算时，我学会了系统性地测试边界条件；在解决成绩输入格式问题时，我掌握了通过添加调试语句定位问题的方法。特别是在处理最小公倍数算法时，通过手动演算验证程序结果，加深了对算法的理解。这些经历让我认识到，有效的调试不仅在于解决问题，更在于预防问题。现在我会在编码前仔细分析需求，考虑各种可能的输入情况。同时，我也体会到理论与实践结合的重要性，只有通过实际编码和调试，才能真正掌握编程知识。这种在实践中学习、在调试中成长的方法，将对我后续的编程学习产生深远影响，也对我的电气专业课程学习有所帮助。