Lab 3 · Part 1 计算基础与 Python 入门 AI Level 0

字符串处理与函数入门

A 段七题练字符串、格式化与函数，其中一道专门暴露 print 和 return 的混淆，最后一道把 Lab 2 的海伦公式写成函数再搭出两个新函数；B 段把一行原始数据变成一张对齐的表，最后会遇到中文名对不齐的问题；C 段学在终端里跑 Python 和重定向。

截止：小作业 1 · 10 月 10 日 周六 23:59

本次目标

- 熟练使用 f-string 控制小数位与对齐
- 正确处理 `input()` 返回的字符串
- 写出带参数和返回值的函数
- **彻底分清 `print()` 和 `return`**

AI 政策提醒

本周处于 **Level 0**：不得使用 AI 生成或补全代码。

判分方式

平台调用你提交的函数，比对**返回值**。除非题目写明「输出」，否则一律 `return`。用 `print` 代替 `return` 的函数，平台拿到的是 `None`，即使算法完全正确也是零分。

这周还没学循环

循环是第 4 讲、列表是第 5 讲。这次的题都只需要顺序代码和函数。`.split()` 会用到一次，但只取固定的两段，不用遍历。

A 段 · 必做

3-1 温度转换

实现 `c_to_f(c)`，摄氏度转华氏度（`F = C * 9/5 + 32`），返回保留一位小数的**浮点数**。

```
c_to_f(0)     →  32.0
c_to_f(37)    →  98.6
c_to_f(-40)   →  -40.0
```

> **提示**
>
> 返回浮点数，不是字符串。`round(x, 1)` 返回的是数，`f"{x:.1f}"` 返回的是字符串—— 这道题要前者。

3-2 学号脱敏

实现 `mask_id(sid)`：保留学号的**前四位和后四位**，中间一律换成 `*`。

```
mask_id("526030910001")  →  "5260****0001"
mask_id("12345678")      →  "12345678"
```

只用切片和拼接，不需要循环。

> **第二个例子想让你想清楚一件事**
>
> 长度正好是 8 的时候，中间没有字符要盖。你的写法在这种情况下会怎样？ 长度不足 8 呢？把你的处理方式填进答题卡。

3-3 对齐的成绩表

实现 `format_row(name, score)`，返回一行格式化字符串：姓名左对齐占 10 格， 分数右对齐占 6 格、保留 1 位小数。

```
format_row("Alex", 87.5)    →  "Alex        87.5"
format_row("Blake", 92.0)   →  "Blake       92.0"
format_row("Chen", 65.25)   →  "Chen        65.2"
```

用 f-string 的 `:<10` 和 `:>6.1f`。

> **中文名不会对齐**
>
> f-string 的宽度按**字符个数**算，不按显示宽度。一个汉字占两个字符位的显示宽度， 但在 `:<10` 眼里只算一个。所以 `f"{'张三':<10}"` 后面会补 8 个空格， 显示出来比 `f"{'Alex':<10}"` 宽一大截，表格就歪了。
>
> 这道题只用 ASCII 姓名，不用处理这个问题。但你要知道它存在—— 真要对齐中文，得自己按显示宽度算，Python 标准库没有直接的办法。

3-4 找出并修正 print/return 的误用

下面这段代码想算出两个数的平均值再乘以 2，但会报 `TypeError`。 **指出原因，改对它**，并在答题卡上用一两句话解释。

```python
def average(a, b):
    print((a + b) / 2)

result = average(10, 20) * 2
print(result)
```

这道题是本次实验的重点

如果你觉得「这题也太简单了」，很好——说明你已经分清了。 但每年都有大量同学在后面的作业里犯同一个错，只是藏得更深。

3-5 安全的数字输入

实现 `read_int(prompt)`：显示提示、读入一行、转成整数并返回。

暂时不用处理输入非数字的情况（那要等 L11 的异常处理），但**自己试一次**： 输入 `abc`，看它抛的是哪一类错误。答题卡上问的是 `input()` 的返回类型， 和这件事是同一个根。

提示

这道题要求你 `print` 提示并 `return` 数值——说明「函数内不要 print」不是绝对的， 而是「函数的**结果**要 return」。提示信息属于交互，不是结果。

3-6 返回多个值

实现 `stats(a, b, c)`，返回三个数的**最小值**、**最大值**、**平均值**三个值。

```
stats(3, 9, 5)  →  (3, 9, 5.666666666666667)
```

调用方应该能这样接：

```
low, high, mean = stats(3, 9, 5)
```

平均值不要四舍五入，原样返回。

3-7 三角形的外接圆与内切圆（平台判分）

Lab 2 你用海伦公式算过三角形面积，那时候是一段顺序代码。这次**把它写成函数**，再用它搭出两个新函数。

```
def area(a, b, c): ...
def circles(a, b, c): ...    # 返回（外接圆半径 R, 内切圆半径 r）
```

公式：

```
S = 海伦公式算出的面积
s = (a + b + c) / 2          外接圆 R = a·b·c / (4S)
                             内切圆 r = S / s
```

```
area(3, 4, 5)      →  6.0
circles(3, 4, 5)   →  (2.5, 1.0)
circles(5, 5, 6)   →  (3.125, 1.5)
```

`circles` 里面**要调用** `area`，不要把海伦公式再抄一遍。 这道题考的就是这件事 —— 判题会检查。

> **3、4、5 那组自己能验**
>
> 直角三角形的外接圆直径就是斜边，所以 `R` 应该正好是 `5 / 2 = 2.5`。 算出来不是，回头看公式。

> **为什么「抄一遍」是错的**
>
> 抄一遍当然也能跑对。但哪天你发现海伦公式在很扁的三角形上误差大、要换个写法 （Lab 2 的 B-4 提过），抄了两遍就要改两处 —— 而且总有一处会被忘掉。 **一件事只写一遍**，是这门课后面反复要讲的事。

B 段 · 深入：把一行原始数据变成一张对齐的表

真实的数据很少是现成可用的。它通常是一行文本，你得先把它拆开、转成对的类型，再排成人能读的样子。这一段四问，每问建立在前一问上，**都不需要循环**。

原始数据长这样，一行一条记录：

```
张三,87.5
李四,65.25
欧阳修远,92
```

B-1 拆开一条

实现 `parse(record)`，把一条记录拆成姓名和分数，**分数要是浮点数**：

```
parse("张三,87.5")   →  ("张三", 87.5)
parse("欧阳修远,92")  →  ("欧阳修远", 92.0)
```

两个提示

`"a,b".split(",")` 会得到两段，用下标 `[0]` 和 `[1]` 取出来 —— 取法和第 3 讲讲的字符串下标一样。

注意第二个例子：输入里写的是 `92`，但要求返回 `92.0`。想想为什么用 `float()` 而不是 `int()`。

B-2 排成一行

用 A 段的 `format_row` 把 `parse` 的结果排成一行：姓名左对齐占 10 格， 分数右对齐占 6 格保留一位小数。

把三条记录各处理一次（写三次调用，不用循环），输出贴进答题卡。

B-3 中文名对不齐

看你 B-2 的输出。ASCII 名字那几行是齐的，中文名那几行**歪了**。

请回答：

1. `len("张三")` 是多少？`len("Alex")` 呢？
2. 但它们在屏幕上占的宽度一样吗？
3. 所以 `f"{名字:<10}"` 到底按什么补空格？

> **这是个真问题，不是玩具**
>
> f-string 的宽度按**字符个数**算，不按显示宽度。一个汉字在等宽字体里占两格，但在 `:<10` 眼里只算一个字符。所以中文名后面补的空格太多，表格就歪了。
>
> 这件事在处理任何中文表格输出时都会遇到。Python 标准库没有直接的办法，得自己按显示宽度算——第 12 讲讲 Unicode 时会给出正式的做法。

B-4 凑合的办法

在不引入新知识的前提下，想一个能让这三行**大致**对齐的办法，写出来并说明它的局限。

> **没有标准答案**
>
> 几个方向：按显示宽度自己补空格、给中文名单独用一个更窄的宽度、或者干脆不对齐改用别的分隔方式。**每一种都有缺点，说清楚缺点比选对更重要。**

C 段 · 基本功：在终端里跑 Python，以及把输出接走

前两周学了走动（`pwd` `ls` `cd`）和增删改（`mkdir` `touch` `cp` `mv` `rm`）。这周学两件事：**在终端里跑 Python 的两种方式**，和**把输出接到别处去**。

交互式与脚本

直接运行 `python` 进入**交互式**（提示符是 `>>>`），一行一行试，适合验证一个想法：

```
python
>>> 2 ** 100
>>> exit()          # 或者按 Ctrl+D
```

运行 `python 文件名.py` 是**执行脚本**，从头到尾跑一遍。写程序用这个。

按键	作用
`Ctrl+C`	打断正在跑的程序
`Ctrl+D`	退出交互式（Windows 上是 `Ctrl+Z` 再回车）
`↑` `↓`	翻之前敲过的命令

重定向：把输出接到文件里

```
python hello.py > out.txt      # 输出写进文件，覆盖
python hello.py >> out.txt     # 追加到文件末尾
python guess.py < input.txt    # 把文件内容当作键盘输入喂给程序
```

为什么这个有用

程序输出几百行时，滚屏根本看不过来。接到文件里再用 `cat` 或编辑器慢慢看。

`<` 那个更有用：程序要 `input()` 五个数，调试时不必每次都手工输入五遍。 写进文件，一行一个，用 `<` 送进程序。

C-1 走一遍

把命令和输出从终端复制出来，填进答题卡（不必截图）：

1. 进交互式，算一下 `2 ** 100`，然后退出
2. 把 B-2 的程序输出重定向到 `table.txt`，再用 `cat table.txt` 看一眼
3. 建一个 `input.txt`，里面写两行数字；用 `<` 喂给你 A 段 3-5 那个读数字的程序

C-2 想一想

1. `>` 和 `>>` 差在哪？你怎么验证？
2. 用了 `>` 之后屏幕上还看得到输出吗？为什么？

提交前自查

- [ ] 每个函数都是 `return` 而不是 `print`（3-5 的提示除外）
- [ ] 3-1 返回的是浮点数，不是字符串
- [ ] 3-7 的 `circles` 是调用 `area` 的，没有把海伦公式抄第二遍
- [ ] A 段答题卡交了（3-2 的边界、3-4 的解释、连读三个数那一问都在里面）
- [ ] B 段答题卡交了
- [ ] C 段答题卡交了