Lab 2 · Part 1 计算基础与 Python 入门 AI Level 0

字符、进制与浮点精度

A 段练转义、ord/chr、进制和两道算式题；B 段把浮点数的误差查清楚，回答 Lab 1 留下的那个问题：两种数学上等价的写法，为什么结果不同。C 段是文件与目录命令。

截止：小作业 1 · 10 月 10 日 周六 23:59

本次目标

- 用转义字符控制输出的样子
- 熟练使用 `ord()` / `chr()` 和 `bin()` / `oct()` / `hex()`
- **亲手测出浮点数从第几位开始不准**，而不是只听说过"浮点数不准"这句话
- 掌握 `/` `//` `%` `**` 的类型规则

AI 政策提醒

本周处于 **Level 0**：不得使用 AI 生成或补全代码。

这周还没学循环

循环是第 4 讲、列表是第 5 讲。**这次实验的每道题都只需要一行接一行地写。** 如果你觉得某道题非要循环不可，那多半是想复杂了。

有两道题要你填一个函数体。`def` 那一行**平台的模板里已经给好了**， 你只要把里面的逻辑补上；函数本身第 3 讲才正式讲。

A 段 · 必做

2-1 转义字符画一只熊

用 `print()` 输出下面这只熊。注意里面有反斜杠和引号，**都要正确转义**：

```
        (()__(()
        /       \
       ( /    \  \
        \ o o    /
        (_()_)__/ \
       / _,==.____ \
      (   |--|      )
      /\_.|__|'-.__/\_
     / (        /     \
     \  \      (      /
      )  '._____)    /
   (((____.--(((____/
```

转义对了是一只可爱的熊，错了是一只畸形的熊。

> **两条出路**
>
> 一是每行一个 `print()`，把 `\` 写成 `\\`； 二是用三引号字符串一次打完（L2「多行字符串」那一节）。**两种都试一遍**， 自己体会一下差别在哪。

2-2 大小写转换（平台判分）

把一个大写字母转成小写。**只能用 `ord()` 和 `chr()`**，不许用 `.lower()`。 不是大写字母的原样返回。

```
to_lower('H')  →  'h'
to_lower('a')  →  'a'
to_lower('7')  →  '7'
```

函数体只有两三行

模板里 `def to_lower(ch):` 已经写好了，你补里面。 思路：先判断它是不是大写字母（用比较运算符，`'A' <= ch <= 'Z'`），是就 `ord()` 加 32 再 `chr()` 回去，不是就原样返回。

2-3 一个字符的一生

挑一个你名字里的**英文字母**（比如 `Z`），顺序写下去，不用函数也不用循环：

1. 用 `ord()` 求出它的编号
2. 用 `bin()` / `oct()` / `hex()` 把这个编号的三种进制打出来
3. 用 `chr()` 从编号变回字符，确认和原来一样
4. 把编号加上 32，再 `chr()` 一次，看看得到什么

第 4 步不是巧合

ASCII 表里大写字母和小写字母正好差 32。想一想：32 在二进制里是 `100000`， 也就是第 6 位。所以「变大小写」在位运算层面只是**翻转一个 bit**。 位运算第 4 讲讲。

2-4 三角形面积

已知三角形三边 `a`、`b`、`c`，海伦公式是：

```
s = (a + b + c) / 2
面积 = √(s(s−a)(s−b)(s−c))
```

开方用 `** 0.5`（`math.sqrt` 要 `import`，第 3 讲才讲）。

分别算这两组，**把两个结果原样打印出来，不要 round**：

	a	b	c
第一组	3	4	5
第二组	0.3	0.4	0.5

第二组的三边正好是第一组的十分之一，所以面积应该正好是第一组的百分之一。

对上了吗？答题卡上有一问就是问这个。这个现象 B 段会解释。

2-5 安全的相等判断（平台判分）

实现 `almost_equal(a, b, eps=1e-9)`：两个浮点数的差的绝对值小于 `eps` 就算相等。

一行就够。`abs()` 是内置函数，直接用。

```
almost_equal(0.1 + 0.2, 0.3)   →  True
almost_equal(1.0, 1.5)         →  False
```

这个函数 B 段要用。

2-6 类型预测

不运行，先把下面每个表达式的**结果和类型**写下来，然后运行验证。 答题卡上会挑其中一个来问，并让你写下猜错了哪几个：

```
8 / 4
8 // 4
8.0 // 4
8 % 3
-8 % 3
-8 // 3
2 ** 10
2 ** -2
2 ** 0.5
True + True
```

这道题必须先猜后跑

直接运行然后抄答案，等于没做。这些规则要形成条件反射—— 后面写循环和列表下标时，一个意外的 `float` 会让程序直接崩。

2-7 一元二次方程求根（平台判分）

实现 `roots(a, b, c)`，解 `a·x² + b·x + c = 0`，**大的根在前**。

可以假定 `a > 0` 且判别式 `b² - 4ac` 为正 —— 也就是恰好两个不相等的实根。（其余情况要用 `if` 判断，那是第 4 讲。）

```
roots(1, -3, 2)   →  (2.0, 1.0)
roots(2, 5, -3)   →  (0.5, -3.0)
```

开方仍然用 `** 0.5`。

返回两个值

直接 `return 大根, 小根` 就行，Python 会把它们打包成一个元组。这个写法第 3 讲讲，这里先用着。

> **判别式要不要先算出来存着**
>
> `(-b + (b*b - 4*a*c) ** 0.5) / (2*a)` 写两遍，判别式就算了两遍。 先 `d = (b*b - 4*a*c) ** 0.5`，下面两行都用它 —— **少算一遍是次要的，读起来清楚才是主要的。**

B 段 · 深入：浮点数到底在哪一位骗你

「浮点数不准」是句空话，直到你知道它**从第几位、在什么情况下**不准。 这一段四问，每问都只要几行顺序代码。

B-1 先看真相

```
print(0.1 + 0.2 == 0.3)
print(0.1 + 0.3 == 0.4)
```

一个是 `False`，一个是 `True`。**同样是两个小数相加，凭什么结果不一样?**

用 `f"{x:.20f}"` 把下面四个数各打二十位小数出来，对照着看：

```
print(f"{0.1:.20f}")
print(f"{0.3:.20f}")
print(f"{0.1 + 0.3:.20f}")
print(f"{0.4:.20f}")
```

在答题卡上回答：`0.1` 存进去是偏大还是偏小？想清楚 `0.3` 偏哪边，下一问要用。

B-2 找出三个例子

`0.1 + 0.2 == 0.3` 是 `False`。请**自己找出另外三个**「数学上明明相等，`==` 却是 `False`」的例子，每个都用 `.20f` 打出双方的真实值。

不要用我给过的那个。

B-3 大数会吃掉小数

试试这两行：

```
print(1e15 + 1 == 1e15)
print(1e16 + 1 == 1e16)
```

一个 `False` 一个 `True`。也就是说，**到了某个量级之后，加 1 这个操作完全没有效果。**

请回答：

1. 从 `1e` 几开始，`+ 1` 就失效了？
2. 在那个量级上，`+ 2` 还有效吗？`+ 4` 呢？把你试出来的写下来。
3. 用一句话解释这是为什么。（提示：浮点数的位数是固定的）

> **这不是玩具问题**
>
> 2016 年，一个统计程序把上亿条记录的金额累加起来，结果比逐条核对少了几万块。 原因就是这个：累加到后期，总额已经很大，每笔小额加上去被直接吃掉了。 **对策是先排序再从小到大加，或者干脆用整数存"分"。**

B-4 回到 Lab 1 那个问题

Lab 1 的 B-3 里，你把

```
x = 3.9 * x * (1 - x)
```

换成了数学上完全等价的

```
x = 3.9 * x - 3.9 * x * x
```

初值相同、公式等价，结果却不同。 当时让你猜，现在你能答了。

请回答：

1. 这两种写法从**第几轮**开始出现差别？（自己跑一次，两个值打二十位小数对照）
2. 为什么会不同？用 B-1 到 B-3 学到的东西解释。
3. 那么，**哪一种写法是"对"的？**

> **第 3 问没有标准答案**
>
> 两种都不是"对"的——它们算的都不是那个数学式子，只是错的方式不同。 这正是数值计算这门学科存在的理由：同一个公式有很多种写法， 它们在数学上等价，在浮点数下误差大小可以差很多倍。
>
> 回过头看 2-4 那道三角形题：0.3、0.4、0.5 那组算出来不是正好 0.06， 就是同一件事。海伦公式在三角形很"扁"的时候误差会大到离谱， 数值分析课上会给你一个专门改写过的版本。

C 段 · 基本功：文件与目录

上周学了 `pwd` `ls` `cd`，只能看和走。这周学**建、删、改、移**。

命令	作用	记法
`mkdir 名字`	建一个目录	**mk**ake **dir**ectory
`touch 文件名`	建一个空文件	文件已存在则只更新时间
`cp 源 目标`	复制	**cp**y
`mv 源 目标`	移动，**也用来改名**	**m**ove
`rm 文件名`	删除	**rm**ove
`cat 文件名`	把文件内容打印出来	con**cat**enate

> **`rm` 没有回收站**
>
> 命令行删掉的东西**不进回收站，找不回来**。删之前先 `ls` 确认一遍。 尤其不要在不确定自己在哪个目录的时候敲 `rm`——先 `pwd`。
>
> 永远不要敲 `rm -rf /`。这条命令会删掉整个系统。

看见平时看不见的文件

`ls` 默认不显示以 `.` 开头的文件，那是约定俗成的「隐藏文件」， 配置文件基本都长这样（`.gitignore`、`.bashrc`、macOS 的 `.DS_Store`）。

```
ls -a       # 全部显示，包括以 . 开头的
ls -l       # 一行一个，带大小、时间、权限
ls -al      # 两个一起用
```

`ls -a` 会多出两项：`.` 和 `..`。它们**不是普通文件**，而是每个目录里都有的两个入口：

	指向
`.`	当前目录本身
`..`	上一级目录

上周用过的 `cd ..` 就是它。`cp ../hello.py .` 里的那个 `.` 也是它。

> **权限那一列第 11 周再讲**
>
> `ls -l` 最左边那串 `drwxr-xr-x` 是权限位，现在不用看懂。只要知道开头的 `d` 表示这是个目录（**d**irectory）就够了。

绝对路径与相对路径

- **绝对路径**从根开始写全：`/home/你的名字/cs1602/hello.py`（Windows 是 `C:\cs1602\hello.py`）
- **相对路径**从当前位置算起：`hello.py`、`../hello.py`、`code/hello.py`

`.` 是当前目录，`..` 是上一级。`~` 是当前用户的主目录。

C-1 走一遍

在终端里依次做完，把**命令和输出**从终端复制出来，填进答题卡（不必截图）：

1. `cd` 到你的 cs1602 文件夹
2. `mkdir week2` 建一个目录，`ls` 确认它出现了
3. `cd week2`，`touch note.txt` 建一个空文件
4. `cp ../hello.py .` —— 把上周的 `hello.py` 复制到当前目录（注意最后那个 `.`）
5. `mv note.txt readme.txt` —— 改名，`ls` 确认
6. `cat hello.py` —— 看看内容

7. `ls -a` 看一眼，`.` 和 `..` 出来了吗？还有别的隐藏文件吗
8. `cd ..`，然后 `pwd` 确认回到了上一级

C-2 想一想

不用敲，直接答：

1. `cp a.txt b.txt` 执行之后，`a.txt` 还在吗？`mv a.txt b.txt` 之后呢？
2. 你在 `~/cs1602/week2` 目录下，想访问 `~/cs1602/hello.py`，相对路径怎么写？

提交

在评测平台的「小作业 1」里提交。2-2、2-5、2-7 是代码题，提交即判分； 其余的（A 段的观察题和 B、C 两段）都在三张**答题卡**上作答：选择题当场判， 文字回答写进评测报告交给助教。

> **记得 return 而不是 print**
>
> 平台调用你提交的函数，比对返回值。这周的题都是顺序代码，直接打印就行； 从下周学了函数开始，这条规则就要一直守着了。