第四次小测题目已经发布到：
[2025 小测练习 1 - 题单 - 洛谷 | 计算机科学教育新生态]

题解：

A. 晶圆产线批次管理

方法思路：按照题目要求设计对应的属性和方法实现即可。注意，初始化给出的 chip 和 tray 数量可能是 17 和 29 的数倍，所以只对 chip（tray）进行一次-17（-29）会导致部分测试点 WA。

如图一所示，self.tray += self.chip // 17 完成的操作是：将 self.chip 整除 17 后得到的数量加到 self.tray 上，在该行代码中，整除（//）的运算优先级比 += 高。另外，构造方法（__init__()）中，self.tray += self.chip // 17 和 self.chip %= 17 这两行代码的顺序不能改变，self.lot += self.tray // 29 和 self.tray %= 29 同理。

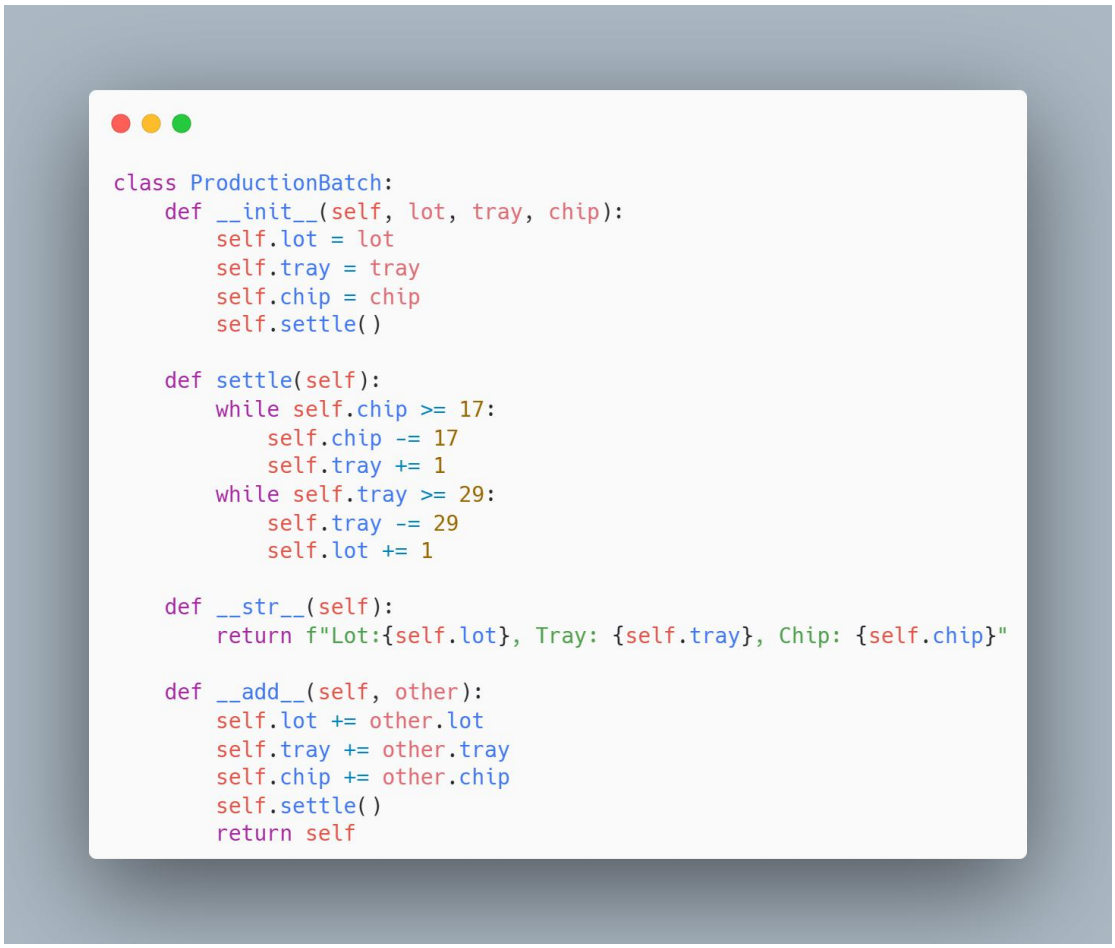
重载__str__()方法时，注意按照题目要求格式返回即可。

重载__add__()方法时，建议使用新变量记录经过加法计算后的值，然后在方法内部实例化一个新的 ProductionBatch 对象，返回该对象。当然，在本题中，修改 self.lot，self.tray，self.chip 然后 return self 也能通过测试点。

```python
class ProductionBatch:
    def __init__(self, lot, tray, chip):
        self.lot = lot
        self.tray = tray
        self.chip = chip
        self.tray += self.chip // 17
        self.chip %= 17
        self.lot += self.tray // 29
        self.tray %= 29

    def __str__(self):
        return f"Lot:{self.lot}, Tray: {self.tray}, Chip: {self.chip}"

    def __add__(self, other):
        new_lot = self.lot + other.lot
        new_tray = self.tray + other.tray
        new_chip = self.chip + other.chip
        new_batch = ProductionBatch(new_lot, new_tray, new_chip)
        return new_batch
```

图一

如图二所示，对于修改 self.lot，self.tray，self.chip 的__add__()方法实现，需要在加法后进行题目要求的“由于防静电包装限制的自动封装处理”，在图二实现中，为了方便代码复用，可以将“封装处理”操作写成一个 settle()方法，然后在构造方法和重载加法中分别调用。

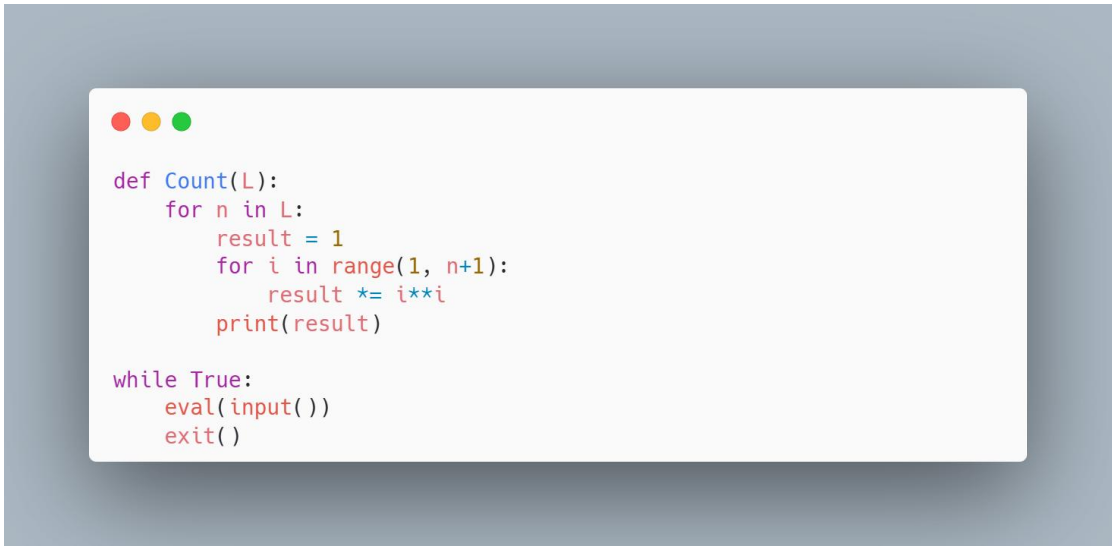


```python
class ProductionBatch:
    def __init__(self, lot, tray, chip):
        self.lot = lot
        self.tray = tray
        self.chip = chip
        self.settle()

    def settle(self):
        while self.chip >= 17:
            self.chip -= 17
            self.tray += 1
        while self.tray >= 29:
            self.tray -= 29
            self.lot += 1

    def __str__(self):
        return f"Lot:{self.lot}, Tray: {self.tray}, Chip: {self.chip}"

    def __add__(self, other):
        self.lot += other.lot
        self.tray += other.tray
        self.chip += other.chip
        self.settle()
        return self
```

图二

B. 金字塔阶乘

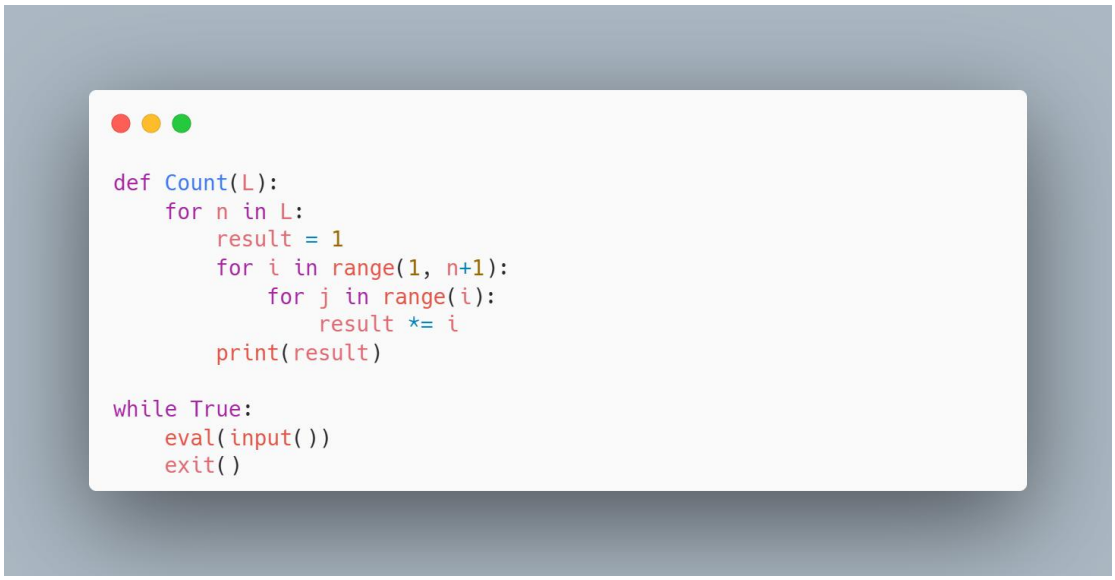
方法思路：如图三根据题目所给公式，只需要每次对列表 L 中的一个数字 n，计算 n 的金字塔阶乘即可，注意计算时 n 的金字塔阶乘时，需要在第一个 for 循环内侧，第二个 for 循环外侧初始化 result = 1 来保存计算结果。对于 i，i 的 i 次方的计算应写成 i**i 而不是“i^i”。另外，本题不会因为计算结果过大而导致 RE 或 WA。



```
def Count(L):
    for n in L:
        result = 1
        for i in range(1, n+1):
            result *= i**i
        print(result)

while True:
    eval(input())
    exit()
```

图三

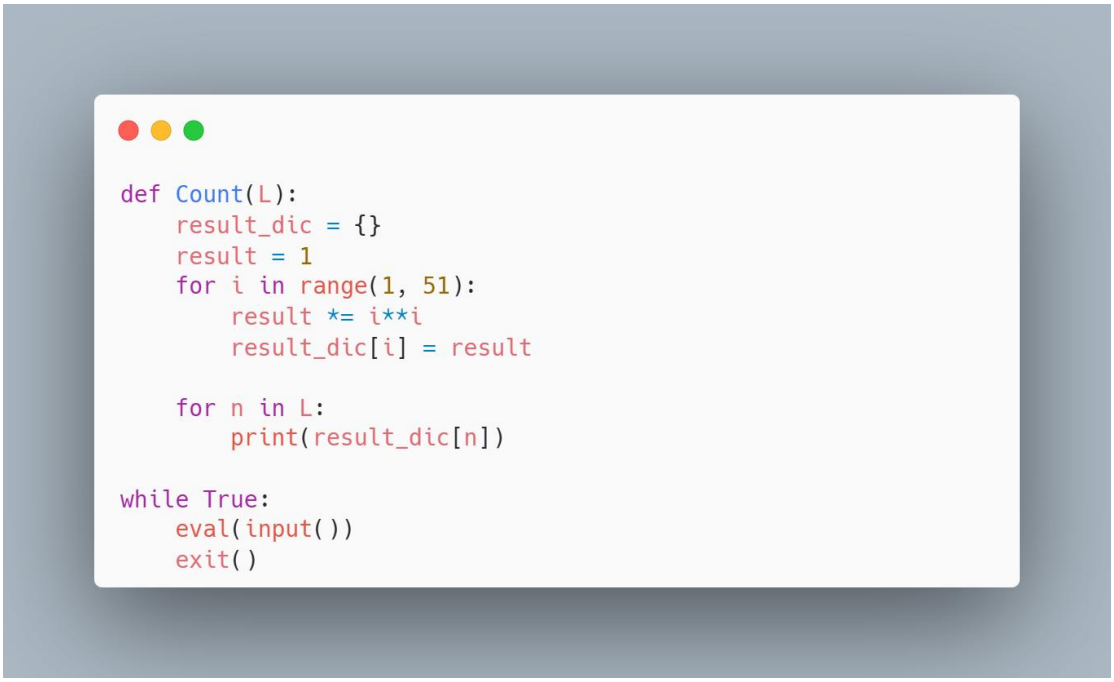
如图四所示，如果忘记 i 的 i 次方怎么计算时，可以再用一个 for 循环来通过乘法实现乘方计算。



```
def Count(L):
    for n in L:
        result = 1
        for i in range(1, n+1):
            for j in range(i):
                result *= i
        print(result)

while True:
    eval(input())
    exit()
```

图四

如图五所示，由于 N 的范围和 L 的范围数量级差距很大，所以会导致重复计算问题，这是非常消耗时间的行为。因此可以预定义一个字典，将 1 到 50 的金字塔阶乘结果保存到字典中，随后通过第二个 for 循环将 L 中的数字对应的金字塔阶乘结果输出即可。



```
def Count(L):
    result_dic = {}
    result = 1
    for i in range(1, 51):
        result *= i**i
        result_dic[i] = result

    for n in L:
        print(result_dic[n])

while True:
    eval(input())
    exit()
```

图五