场景实战题
开发指南

Ver.2

蓝桥 · 教学研究中心
Teaching and Research Center

文档版本	更新时间	作者
Ver. 1	2021-06	张航
Ver. 2	2022-03	张航

什么是 OJ（A+B 问题）

在线评测平台（英语：Online Judging System，简称：OJ），一般用于刷题训练，参与和组织比赛。

问题描述

输入两个正整数 A,B，请你计算 A+B，并输出。

输入描述

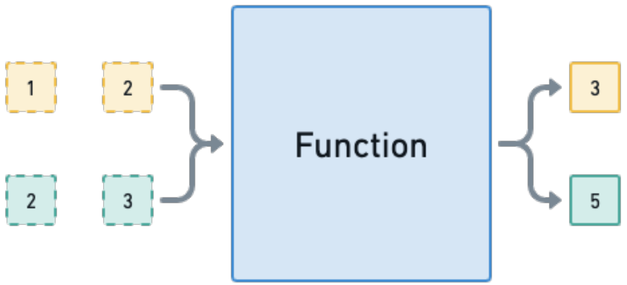
输入一行，为用空格分隔的两个正整数 a,b $(1 \leqslant a,b \leqslant 100)$。

输出描述

输出一行，为 a+b 的结果。（后台测试会忽略行末空格及换行）

输入输出样例

输入：1 2
输出：3

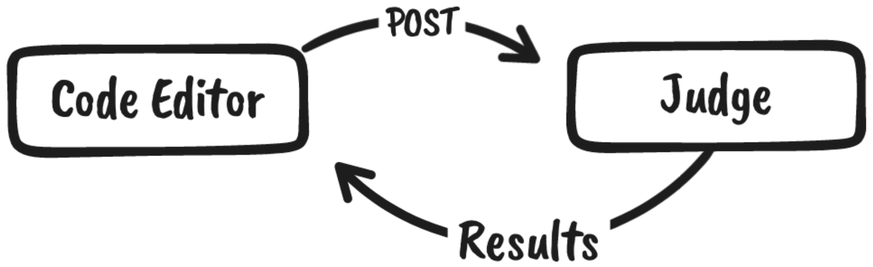



```python
import os

s = input().split(" ")
a = int(s[0])
b = int(s[1])
print(a + b)
```

OJ 的检测

OJ 后台测试用例配置支持多组。一般为纯文本或 .txt 附件。



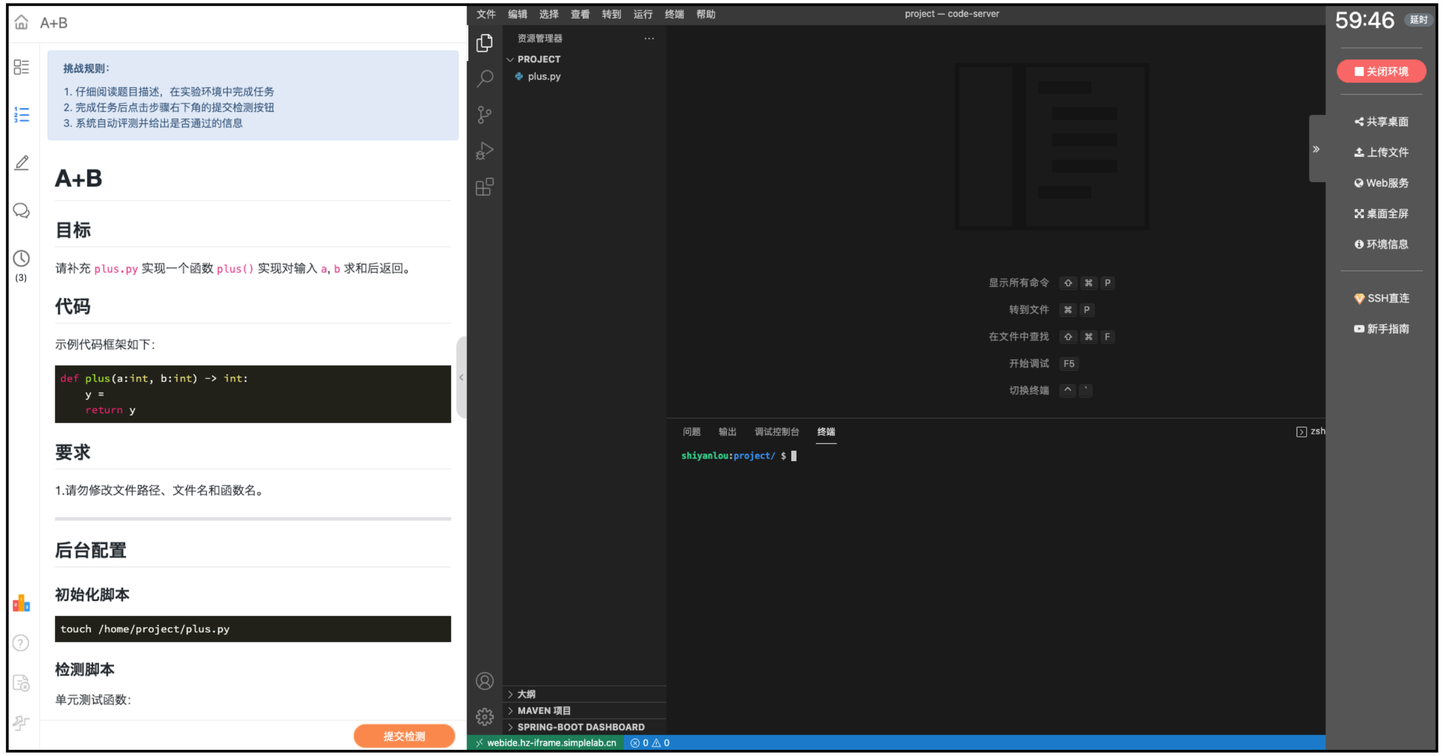
- 题型说明：基于编程语言的标准输入输出，实现相应的功能。
- 支持语言：同一道编程题，OJ 不限制编程语言。
- 检测方式：严格和死板的输入输出测试用例。
- 制作难度：题面的难度可高可低，检测配置难度低。
- 明显弊端：不适用于场景实操类题目，例如：前端、数据库等。

什么是挑战

挑战是蓝桥云课支持的一种考查章节类型，依托于在线实验环境实现，可针对多场景实操能力考查。

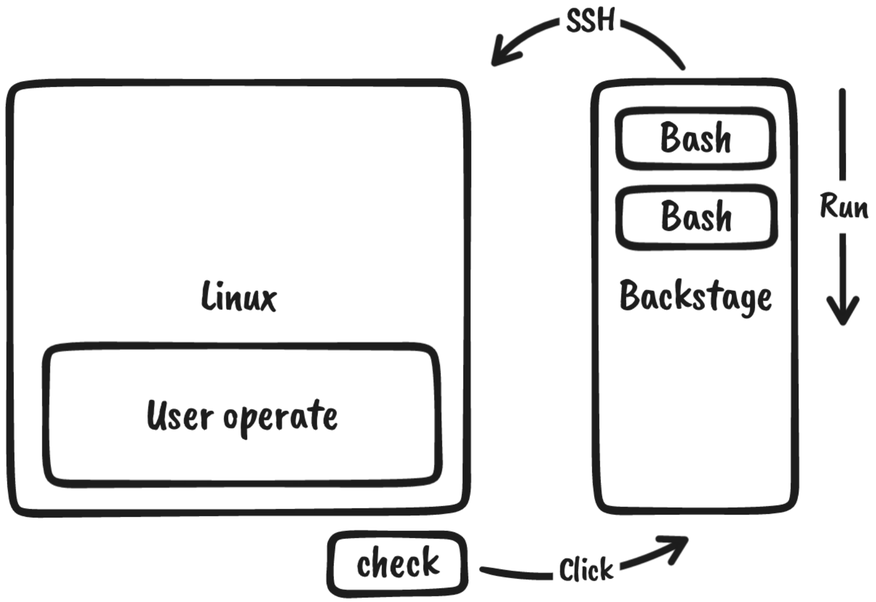
首先，挑战可以完全兼容 OJ，但有不同：

1. 题目的描述需要更加明确，尤其是要求和规范，例如需要用户在指定路径文件下编写代码。OJ 只提供了一个代码编辑器，挑战提供了一个完整的 Linux 环境。
2. 检测配置会有不同，具体涉及到挑战的检测机制。



挑战的检测机制

挑战的检测机制和 OJ 有很大不同。



挑战检测过程大致如下：

1. 用户依据题目描述在环境完成相应操作。
2. 点击提交检测后，触发后台检测脚本依次执行。
3. 脚本执行过程是通过 SSH 连接到用户环境，以默认用户权限在环境中执行。检测脚本为 Bash 命令。
4. 执行报错即抛出相应的错误，全部执行完成即通过。

A+B 挑战版

如何使用挑战方式呈现 A+B 问题？

题面

问题描述

输入两个正整数 A,B，请你计算 A+B，并输出。

输入描述

输入一行，为用空格分隔的两个正整数 a,b (1≤a,b≤100)。

输出描述

输出一行，为 a+b 的结果。（后台测试会忽略行末空格及换行）

输入输出样例

输入：1 2
输出：3

挑战要求

请补充 plus.py 使其支持通过 Python 标准输入、输出达到挑战目标。

后台配置初始化 + 检测脚本

检测脚本详解

```
echo 1 2|python /home/project/plus.py|grep 3
echo 7 8|python /home/project/plus.py|grep 15
```

检测脚本编写原则：

1. Bash 书写，使用绝对路径。
2. 默认情况下不打开「展示详细错误信息选项」，stderr 可能会暴露检测方法。
3. 检测脚本尽量拆分足够细致，给予用户更多的提示信息。
4. 注意挑战检测脚本之间的关联性，可能会导致重复运行时检测未按预期完成。

进阶玩法：通过自定义的单元测试脚本抛出代码异常提示

A+B 挑战优化版

如何使用更贴近挑战风格的方式优化 A+B 问题？

题面

目标

请补充 plus.py 实现一个函数 plus() 实现对输入 a, b 求和后返回。

代码

示例代码框架如下：

```
def plus(a:int, b:int) -> int:
    y =
    return y
```

要求

1. 请勿修改文件路径、文件名和函数名。

单元测试

```
# 普通单元测试 plus_test.py

import sys

sys.path.append('/home/project')

from plus import plus

assert plus(1, 1) == 2
assert plus(2, 3) == 5
```

```
# unittest 单元测试 plus_test.py

import sys

sys.path.append('/home/project')

import unittest
from plus import plus


ut = unittest.TestCase()
ut.assertEqual(plus(1, 1), 2)
ut.assertEqual(plus(2, 3), 5)
```

检测脚本

```
cd /tmp
wget https://labfile.oss.aliyuncs.com/courses/3947/plus_test.py && (python plus_test.py; rm plus_test.py)
```

检测脚本编写原则：

1. 使用自己熟悉的语言编写单元测试。
2. 将单元测试脚本打包上传至「上传文件」。
3. 使用 Bash 编写检测流程（下载单元测试到 /tmp → 执行单元测试 → 移除单元测试。）

挑战的特点

相比于 OJ，挑战具备的特点如下：

- 题型说明：挑战可以兼容 OJ 的全部题型，还可以覆盖场景实战。
- 支持语言：挑战支持的编程语言更加丰富，场景更加多样。但同一道题大部分只支持单一语言。
- 检测方式：纯 Bash 或者 Bash+单元测试。
- 制作难度：题面的难度可高可低，准确的检测脚本编写难度较高。

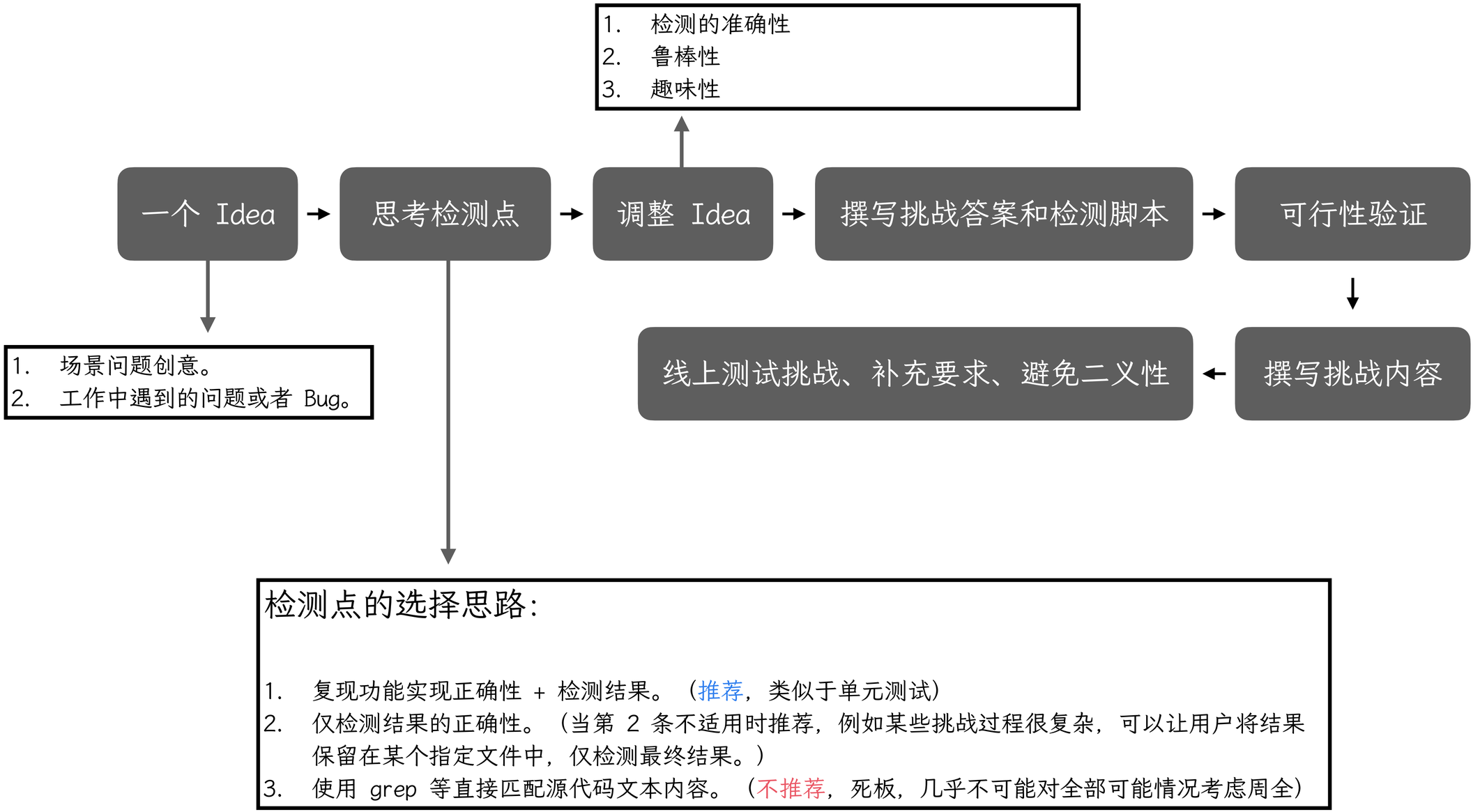
格式要求

- 题面使用 Markdown 书写，基础的格式要求和实验一致。
- 必须存在的模块有「挑战介绍」，「知识点」，「挑战要求」。
- 按需选择的模块有「挑战内容」，「挑战准备」，「测试用例」，「示例代码」，「挑战提示」等。

注意事项

- 准确、清晰、简洁地表达挑战内容和要求。
- 挑战检测的准确和灵活，尺度的把控。

挑战设计流程


1. 检测的准确性
2. 鲁棒性
3. 趣味性
一个 Idea
思考检测点
调整 Idea
撰写挑战答案和检测脚本
可行性验证
撰写挑战内容
线上测试挑战、补充要求、避免二义性
1. 场景问题创意。
2. 工作中遇到的问题或者 Bug。
检测点的选择思路：
1. 复现功能实现正确性 + 检测结果。（推荐，类似于单元测试）
2. 仅检测结果的正确性。（当第 2 条不适用时推荐，例如某些挑战过程很复杂，可以让用户将结果保留在某个指定文件中，仅检测最终结果。）
3. 使用 grep 等直接匹配源代码文本内容。（不推荐，死板，几乎不可能对全部可能情况考虑周全）

前端自动化测试

前端自动化测试分为三部分：效果（UI 界面）测试、交互测试、单元测试，详见介绍文档。


```
export function hello() {
  return 'Hello, World!';
}
```

```
import { hello } from './hello-world';

describe('Hello World', () => {
  test('Say Hi!', () => {
    expect(hello()).toEqual('Hello,
World!');
  });
});
```

原生方法或者使用 Jest

Java 单元测试

Java 单元测试推荐使用 Junit 5，可以参考：

1. exercism/java

```java
class Greeter {

    String getGreeting() {
        return "Hello, World!";
    }

}
```

```java
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class GreeterTest {

    @Test
    public void testThatGreeterReturnsTheCorrectGreeting() {
        assertEquals("Hello, World!", new Greeter().getGreeting());
    }

}
```

Python 单元测试

Python 单元测试推荐使用 unittest，可以参考：

1. donnemartin/interactive-coding-challenges
2. exercism/python

```python
def hello():
    return 'Hello, World!'
```

```python
import unittest

from hello_world import (
    hello,
)

class HelloWorldTest(unittest.TestCase):
    def test_say_hi(self):
        self.assertEqual(hello(), "Hello, World!")

if __name__ == "__main__":
    unittest.main()
```

Go 单元测试

Go 单元测试推荐直接构造，可以参考：

1. exercism/go

```go
package greeting

func HelloWorld() string {
    return "Hello, World!"
}
```

```go
package greeting

import "testing"

func TestHelloWorld(t *testing.T) {
    expected := "Hello, World!"
    if observed := HelloWorld(); observed != expected {
        t.Fatalf("HelloWorld() = %v, want %v", observed, expected)
    }
}
```

Bash 单元测试

Bash 单元测试推荐使用 Bats-core，可以参考：

1. exercism/bash

```
#!/usr/bin/env bash

echo "Hello, World!"
```

```
#!/usr/bin/env bats
load bats-extra

# local version: 1.1.0.0

@test "Say Hi!" {
  run bash hello_world.sh

  # the program's exit status should be success (0)
  assert_success

  # program's output should be the expected text
  assert_output "Hello, World!"
}
```

其他场景检测

1. 数据库：可以使用擅长的编程语言连接到数据库访问数据判断。还可以利用数据库日志进行检测。
2. API/HTTP 请求：利用 curl 或者擅长编程语言构建 HTTP 请求。

准确检测的原则

使用混淆矩阵来表示用户答案的正确性和检测判定的正确性。

	判定正确	判定错误
用户正确	真阳性 (TP)	假阴性 (FN)
用户错误	假阳性 (FP)	真阴性 (TN)

努力降低 FP 和 FN。
宁可 FP，也不要 FN。

实验化的分步骤试题

相同点

1. 基本设计逻辑和要求与独立挑战完全一致；
2. 检测脚本的配置字段与独立挑战基本一致；
3. 适合用于复杂的场景题目，例如完整的大数据处理过程。

不同点

1. 使用 Checker 而非后台配置检测脚本。
2. 步骤中建议给出用户前台单元测试（与后台不同）。
3. 避免前后步骤的过度依赖。

挑战检查清单

完成的挑战可以参考检查单进行逐项核对 →

基础检查

- 清晰简洁的题目标题、步骤标题。
- 文档语句通顺，无语病。
- 文档格式规范。
- 文档配图清晰，比例规范。
- 文档无错别字。
- 文档重点提示信息明确、清晰。可使用加粗等样式处理。
- 文档内容和其他资源（源代码等）的一致性（内容、截图匹配等）。
- 源代码已格式化且代码注释详细、准确。
- 源代码英文关键字的单词拼写准确。

题意检查

- 题目考点明确，题意明确，细节描述到位。
- 题目的限制条件思考全面，限制条件明确。
- 题目的输入输出规范，输入输出规范明确。
- 题目中不存在可以这样、可以那样的描述，不要让用户多想或者少想。

判题检查

- 题目的检测不能过于死板。

任务发布

任务说明

大家可以结合自己的工作场景和经验，撰写 1 套楼赛的出题大纲（10 题目），其中：

1. 题目难度一般为 2 简单（90% 通过），6 中等（50% 以上），2 困难（50% 以下）。
2. 题目之间可以相互联系，同一主题，或者层层递进。也可以完全独立。
3. 列出题目名称和考查点，题目包含至少 1 个考查点，不宜多余 2 个。
4. 创建私有课，完成 1 道题目的试写。（也可以先完成 1 题，再写楼赛大纲）

奖励说明

1. 试写合格后，我们可以签订 1 份楼赛出题协议，提供 1 期楼赛题目。报酬 1500 元 + 500 首次奖励。
2. 楼赛上线后，我们可以给予出题专家认证，后续参与楼赛、蓝桥认证出题。
3. 楼赛题目约 1500 元/套，蓝桥认证题目报酬约 300~500 元/题。


蓝桥云课