**你要做什么（一眼看懂版）**

* 写一个**服务器** server.py 和一个**客户端** client.py，通信全部用 **JSON**（UTF-8 编解码）通过 TCP socket 发送/接收；消息类型固定（HI/BYE/ANSWER；READY/QUESTION/RESULT/LEADERBOARD/FINISHED）。
* 两个程序都用 --config <path> 启动并按格式读取配置（服务器配置含端口、题型顺序、时间限制、文案占位符等；客户端配置含用户名、模式 you/auto/ai 以及可选 Ollama 设置）。
* 服务器流程：收够 players 个 HI 后广播 READY，按 question\_types 顺序出题，每题限时 question\_seconds 秒，或全部答复到达则立刻结算并发 LEADERBOARD，所有题后发 FINISHED（处理并发/断线与计分、并**排除断线玩家**出榜单/最终排名）。
* 客户端流程：用户输入 CONNECT host:port 去连；收到 READY/QUESTION/RESULT/LEADERBOARD/FINISHED 要**打印指定字段**；模式 you/auto/ai 只影响**答案的生成**；每题超时必须放弃这个答案等待下一题（推荐 signal.alarm 之类）。
* **题型**与示例：Mathematics、Roman Numerals、Usable IP Addresses of a Subnet、Network & Broadcast；服务端要随机生成 short\_question，客户端 auto 必须 100% 正确；答案按**字符串**精确匹配（不做“聪明匹配”）。
* **LEADERBOARD/FINISHED** 排名与并列、单复数词形必须按规范构造并打印；并列按**字典序**打破且并列名次共享名次号。
* **错误处理**：两端对 --config 缺失、文件不存在等要按指定文案打印到 stderr 并 exit 1；服务器绑定失败要报错；客户端在 ai 模式缺少 Ollama 配置要报错。
* **允许导入**模块清单（socket、selectors、requests、signal、threading、dataclasses、pathlib…）见规格尾部，测试可以用更多 Python 模块（不受导入白名单限制）。
* **测试与打分**：功能覆盖 + 覆盖面（可用 ncat 测服务端，或你的服务端配你的客户端）；自动测试占 12 分，手改 3 分（测试 2 分 + 代码风格 1 分）。

**我们的实施路线（一步一步做）**

**Step 0 — 建项目骨架与样例配置**

在仓库根目录创建：

client.py

server.py

questions.py

utils.py # 放通用：json 编解码、收发、超时工具等

configs/

server.example.json

client.you.example.json

client.auto.example.json

client.ai.example.json

tests/ # 之后放 .in/.expected 或 Python 测试

拷贝下面两份最小配置作为起点（你可直接改数值）：

**configs/server.example.json**

{

"port": 7777,

"players": 2,

"question\_types": [

"Mathematics",

"Roman Numerals",

"Usable IP Addresses of a Subnet",

"Network and Broadcast Address of a Subnet"

],

"question\_formats": {

"Mathematics": "What is {}?",

"Roman Numerals": "What is the decimal value of the roman numeral {}?",

"Usable IP Addresses of a Subnet": "How many usable addresses are there in the subnet {}?",

"Network and Broadcast Address of a Subnet": "What are the network and broadcast addresses of the subnet {}?"

},

"question\_seconds": 8,

"question\_interval\_seconds": 3,

"ready\_info": "Game starts in {question\_interval\_seconds} seconds!",

"question\_word": "Question",

"correct\_answer": "Woohoo! Great job! You got it!",

"incorrect\_answer": "Maybe next time :(",

"points\_noun\_singular": "point",

"points\_noun\_plural": "points",

"final\_standings\_heading": "Final standings:",

"one\_winner": "The winner is: {}",

"multiple\_winners": "The winners are: {}"

}

（这些键名与占位替换规则都来自规格的“配置”段落；ready\_info 要用 str.format(\*\*config) 展开成具体秒数）。

**configs/client.you.example.json**

{ "username": "cheetah", "client\_mode": "you" }

**configs/client.ai.example.json**（ai 模式才需要 ollama\_config）

{

"username": "cheetah",

"client\_mode": "ai",

"ollama\_config": { "ollama\_host": "localhost", "ollama\_port": 11434, "ollama\_model": "mistral:latest" }

}

（ai 模式缺它要报错退出）。

**Step 1 — 放上“能启动且报对错”的最小主程序**

**server.py（最小版骨架）**

* 解析 --config → 读 JSON；缺参/不存在/端口 bind 失败时按文案报错并 exit 1。
* 打开 socket 监听 0.0.0.0:port，接收连接；收到 **HI**（含 username，且必须是**字母数字**，否则立即关闭所有连接并退出）后视为“加入”；人数到达 players 后群发 **READY**，info=把 ready\_info 用 str.format(\*\*config) 替换后得到的串。
* 暂不出题，先能收 HI 与发 READY 就行。

**client.py（最小版骨架）**

* 解析 --config；缺参/不存在按文案报错并 exit 1；ai 且无 ollama\_config 也要报错。
* 启动后等待用户输入：CONNECT host:port；连不上打印 Connection failed 并**退出**；连上立刻发 **HI**（带 username）。
* 能接收 READY 并把 info 打印出来，每行末尾记得换行。

先把这一步跑通，你就已经通过了大量“启动/错误处理/基本协议”的点。

**Step 2 — 抽出通用工具（utils.py）**

* send\_json(sock, obj) / recv\_json(sock)：json.dumps(...).encode('utf-8') 发送、json.loads(...) 解码（**通信内容只会是规范 JSON，不会缺字段**，UTF-8 不会失败）。
* format\_placeholders(template\_str, config\_dict)：用于 ready\_info 等占位符替换（"{a} + {b}".format(\*\*values) 思路）。
* 客户端超时工具：封装 signal.alarm(seconds)/signal.setitimer 以在 you/ai 等待答案时能“放弃等待”。

**Step 3 — 出题与解题（questions.py）**

* 提供**随机**生成 short\_question 的函数（服务端用）：
  + generate\_mathematics\_question()：正整数、最多 5 个数、4 个运算，+/- 比 \*// **更常见**；- 不是一元负号（不产生负数操作数）。
  + generate\_roman\_numerals\_question()：先 randint(1,3999) 再转罗马数字返回。
  + generate\_subnet\_usable\_question()：给 a.b.c.d/p；答案是可用地址数（如 /24 → 254）。
  + generate\_subnet\_net\_broadcast\_question()：给 a.b.c.d/p；答案是 net and broadcast 两个值，**客户端 ANSWER 要用 "x and y" 这种格式**。
* 客户端 auto 模式的**解题器**：
  + 数学题：建议“shunting-yard → RPN → 栈求值”（不是硬性要求）。
  + 罗马数字：标准规则映射。
  + 子网：用位运算或 ipaddress（未在白名单；建议你自己算：可用数= max(2^(32-p)-2, 0)；网络/广播按掩码清零/全一生成）。
  + 输出一律**字符串**，与正确答案**逐字相等**才算对。

**Step 4 — 完整消息流（最少可玩）**

* 服务器按题型循环构造 **QUESTION**：  
  trivia\_question = f"{question\_word} {i} ({type}):\n{format\_question}"；同时附上 short\_question 和 time\_limit（来自配置）。
* 客户端收到 QUESTION：打印 trivia\_question；随后**根据模式**生成答案并发 **ANSWER**；收 **RESULT** 打印 feedback。
* 服务器收到 ANSWER：字符串比较，答对给该玩家 +1 分，向该客户端发 **RESULT**（correct/feedback 用配置中的文案）。
* 每题结束：服务器向所有客户端发 **LEADERBOARD**（state 字符串按规则排序/并列号/单复数），最后一题后发 **FINISHED**（带标题、名次列表与“winner(s)”串）。

并发/断线处理：断线或 BYE 后此玩家分数不再变化，且**不出现在**榜单与最终名次中（但“在局”视为之后都答错）。

**Step 5 — ai 模式（Ollama）**

* 规范里对端点有措辞不一致：一处写“使用 /api/chat（无 streaming/tool/structured）”，紧接着又说“只用 /api/generate”。为稳妥起见，**实现 /api/generate** 的简单 POST，并保留可选回退到 /api/chat 的逻辑（若 404 则退回），且**不要修改模型返回文本**，原样发给服务器（测试会对比）。
* 你的本地可用课程提供的 ollama.py 脚本模拟 API（评测也会用相似脚本）。

**Step 6 — 基本测试与自检**

* **单端测试**：
  + 用 ncat 伪造客户端测服务器消息是否完全符合规范（建议做若干 .in/.expected）。
  + 你的客户端连你的服务器，校对每条消息的**打印内容**是否完全匹配期望文本（换行！）。
* **评分点清单**（交前逐项打勾）：
  + 必备错误文案 & 退出码 1（两端）。
  + READY/QUESTION/RESULT/LEADERBOARD/FINISHED 的**打印字段**正确。
  + 排名构造与并列/单复数/字典序 OK（含多赢家）。
  + auto 题解 100% 正确；ai 不篡改 LLM 回答；超时能放弃。
  + 只用**允许导入**（测试脚本不限）。

为了确保你能够拿到满分，我将根据作业要求帮你梳理出**要做的事情**，并且提供每个任务的具体步骤和要点。

**1. 配置文件与参数验证**

* **配置文件**：实现 --config 参数，允许用户传入配置文件路径，若文件缺失或格式不正确，则程序报错并退出。
  + **错误提示**：配置文件不存在、字段缺失或格式不对时，服务器和客户端都需要**给出清晰的错误信息**并退出（exit 1）。
  + **配置文件内容**：确保你的配置文件结构正确，且能满足作业中要求的格式（比如端口、玩家数量、问题类型、时间限制等）。

**2. 服务器功能**

* **监听端口**：服务器需要**监听指定端口**并接受客户端连接。
  + **端口绑定失败**：如果服务器无法绑定端口，应该**报错并退出**。
* **处理连接**：
  + 每当一个客户端连接后，接收并处理 **HI** 消息（含用户名）。
  + **用户名验证**：客户端的用户名必须是字母或数字，若用户名无效，服务器应**断开所有连接并退出**。
* **玩家匹配与 READY 消息**：
  + 收到 **HI** 消息后，**存储玩家信息**，直到满足玩家数量（players）。
  + 一旦玩家数量达到指定数量，发送 **READY** 消息，并**广播到所有客户端**，格式应包括配置文件中指定的 ready\_info。
* **出题逻辑**：
  + 按照配置中的题目类型（如数学、罗马数字等）生成问题。
  + 每道题在 **question\_seconds** 时间内必须给出答案，超时则跳过该题。
* **处理玩家答案**：
  + 接收每个玩家的答案，进行**正确性验证**。
  + 根据答案**更新分数**，并向客户端返回 **RESULT** 消息。
* **结算与排名**：
  + 每道题结束后，服务器会**更新排名**并发 **LEADERBOARD** 消息。
  + 最后一题结束后，发 **FINISHED** 消息，包含最终排名，打印 **winner(s)**。

**3. 客户端功能**

* **连接与验证**：
  + 客户端在启动后需要**解析配置文件**，并获取用户名与客户端模式（you/auto/ai）。
  + 客户端需要**连接到服务器**，并发送 **HI** 消息，包含用户名。
* **接收 READY 消息**：
  + 当收到服务器发送的 **READY** 消息时，客户端需要**打印提示信息**（info）。
* **回答问题**：
  + 在 **QUESTION** 消息到来时，客户端根据不同模式（you/auto/ai）提供答案：
    - **you** 模式：用户输入答案。
    - **auto** 模式：客户端自动生成答案，必须**保证正确**。
    - **ai** 模式：客户端通过配置的外部 API（如 Ollama）自动生成答案。
* **接收 RESULT 消息**：
  + 客户端收到 **RESULT** 消息后，应**打印结果**（是否正确，提示信息）。
* **处理超时**：
  + 如果客户端在限定时间内未能回答问题，应**跳过该问题**并等待下一题。
* **接收 LEADERBOARD 和 FINISHED 消息**：
  + 客户端接收 **LEADERBOARD** 后**打印排名**。
  + 当收到 **FINISHED** 消息时，客户端应**打印最终排名**。

**4. 题目生成**

* **题目类型**：根据配置生成不同类型的问题，如：
  + 数学问题：生成加减乘除的计算题。
  + 罗马数字：给定一个数字，要求转换为罗马数字。
  + 子网问题：计算子网的可用地址数，或者给定 IP 和子网掩码，计算网络地址和广播地址。
* **题目格式**：
  + 对于每种题型，确保输出问题时遵循指定格式，并提供正确答案。

**5. 错误处理与异常管理**

* **配置文件错误**：如果缺少必需的配置文件或配置内容不符合要求，程序应**报错并退出**。
* **网络错误**：如果服务器无法绑定端口，或客户端无法连接到服务器，应**打印错误信息并退出**。
* **无效答案**：如果客户端提交的答案格式不符合要求，应该**忽略该答案**并继续等待下一题。

**6. 测试**

* **自动化测试**：通过模拟客户端和服务器的交互，确保问题能够正确生成、答案能够正确判定，排名能够准确更新。
* **功能覆盖**：
  + 每个功能点（如连接、出题、答题、排名、错误处理等）都要确保能够通过测试验证其正确性。

**7. 提交前的最终检查**

* **功能完整性**：确保所有核心功能（玩家匹配、问题生成、答案验证、排名、结算）都已经实现，并且按要求正常运行。
* **错误处理**：确保所有的错误都能**优雅地处理**，并给出明确的错误提示。
* **代码风格与规范**：
  + 确保代码清晰，注释到位。
  + 按照作业要求提交配置文件、客户端、服务器以及其他必要的文档。

**总结**

为了拿到满分，必须确保程序能够：

1. 正确地读取配置文件，按配置运行；
2. 实现服务器与客户端之间的通信，能够正确处理玩家连接、出题、答题、评分和排名；
3. 处理各种错误情况，确保程序能够稳定运行；
4. 生成问题并验证答案，确保答案的正确性；
5. 最后通过正确的排名与最终结算显示游戏结果。