**Mini “Vibe Coding” Challenge — Ping-Pong in Python**

**Tool A: GitHub Copilot**

* **Project Structure:** Multi-file project with clear separation: constants.py, paddle.py, ball.py, game.py, and main.py .
* **Features:** Meets all requirements—window, paddles, ball movement, collisions, scoring. Both paddles are player-controlled.
* **Strengths:** Modular and easy to extend (AI, power-ups, menus). Constants in one file make tuning easy. Runs smoothly at 60 FPS.
* **Weaknesses:** Test files don’t match the codebase generated. Ball reset logic is predictable (always in the same direction).

**Tool B: Kiro**

* **Project Structure:** One single file (ping\_pong.py). All logic—paddles, ball, scoring—is inline.
* **Features:** Same basics covered: ball bounces, paddle collisions, score updates, on-screen display.
* **Strengths:** Very quick to run (python ping\_pong.py). Easy for beginners since everything is in one place.
* **Weaknesses:** Harder to maintain if adding new features; no modularity. Like Copilot’s version, ball reset is basic and collisions are simplistic.

**Debugging & error handling**

* **Copilot:** Because classes are small, I can isolate issues quickly (e.g., breakpoints in Ball.move or the collision block). The quit flow is correct in main.py. I’d add small guardrails if running on headless machines, but for a lab it’s fine.
* **Kiro:** Easy to pepper print()s anywhere. Event loop/quit handling is correct, and the 60 FPS tick keeps input smooth. As the file grows, it’ll be harder to keep things tidy without refactoring to modules.

**Flexibility & customization**

* **Copilot:** Better baseline for growth. I can drop in a computer paddle (update\_ai(ball)), angle the ball based on paddle contact point, or add “speed-up on hit” without making a single file unwieldy. Constants being centralized help a lot when tuning gameplay.
* **Kiro:** Fine for quick tweaks (change speeds/sizes, font, controls). For bigger features, I’d eventually split it into modules that look… basically like Copilot’s layout.

**Ball reset logic could be spicier in both.** Right now, the ball resets to center and either uses a fixed speed (Copilot) or flips X (Kiro). For more variety, I’d randomize the initial X direction and pick a small range for Y so rallies don’t feel same. But I would need to break the code a lot and then go about with this.

**Comparison**

| **Aspect** | **Copilot** | **Kiro** |
| --- | --- | --- |
| **Organization** | Modular, professional, easy to expand | Simple, single file |
| **Ease of Run** | Needs imports/modules, but clean | Run immediately |
| **Extensibility** | Better suited for future upgrades | Would get messy for large features |
| **Learning Curve** | Matches standard Pygame structure | Easier for a quick start |
| **Issues** | Mismatched tests, predictable ball reset | Less maintainable, also predictable reset |

**My takeaway:**

* If I just want a fast playable demo, Kiro’s single-file code is perfect.
* If I plan to extend or polish the game, Copilot’s modular design is the better foundation.