Antigravity

Google Antigravity @antigravity.google

Extensions

- Antigravity Quota (AGQ) Antigravity Quota (AGQ) – Open VSX Registry @open-vsx.org

Settings

- 오른쪽 상단 '...' Additional Options → Customizations 메뉴 → Rules 탭
  - ```
    # Antigravity Agent Rules
        1. **Language**: Always respond in Korean (한국어). Even if the user asks in English or the context is technical, provide explanations in Korean unless explicitly requested otherwise.
        2. **Tone**: Maintain a helpful, professional, and friendly tone (my style if applicable).
    ```
- Model 선택
- 오른쪽 상단 □□ Editor-Specific Settings → Editor Settings 메뉴 → Antigravity Settings 탭 → Terminal Command Auto Execution 항목에서 Request Review

Skills

- GitHub - forrestchang/andrej-karpathy-skills @github.com

```
---
name: karpathy-guidelines
description: Behavioral guidelines to reduce common LLM coding mistakes. Use when writing, reviewing, or refactoring code to avoid overcomplication, make surgical changes, surface assumptions, and define verifiable success criteria.
license: MIT
---

# Karpathy Guidelines

Behavioral guidelines to reduce common LLM coding mistakes, derived from [Andrej Karpathy's observations](https://x.com/karpathy/status/2015883857489522876) on LLM coding pitfalls.

**Tradeoff:** These guidelines bias toward caution over speed. For trivial
```
```

```
tasks, use judgment.

## 1. Think Before Coding

**Don't assume. Don't hide confusion. Surface tradeoffs.**

Before implementing:
- State your assumptions explicitly. If uncertain, ask.
- If multiple interpretations exist, present them - don't pick silently.
- If a simpler approach exists, say so. Push back when warranted.
- If something is unclear, stop. Name what's confusing. Ask.

## 2. Simplicity First

**Minimum code that solves the problem. Nothing speculative.**

- No features beyond what was asked.
- No abstractions for single-use code.
- No "flexibility" or "configurability" that wasn't requested.
- No error handling for impossible scenarios.
- If you write 200 lines and it could be 50, rewrite it.

Ask yourself: "Would a senior engineer say this is overcomplicated?" If yes,
simplify.

## 3. Surgical Changes

**Touch only what you must. Clean up only your own mess.**

When editing existing code:
- Don't "improve" adjacent code, comments, or formatting.
- Don't refactor things that aren't broken.
- Match existing style, even if you'd do it differently.
- If you notice unrelated dead code, mention it - don't delete it.

When your changes create orphans:
- Remove imports/variables/functions that YOUR changes made unused.
- Don't remove pre-existing dead code unless asked.

The test: Every changed line should trace directly to the user's request.

## 4. Goal-Driven Execution

**Define success criteria. Loop until verified.**

Transform tasks into verifiable goals:
- "Add validation" → "Write tests for invalid inputs, then make them pass"
- "Fix the bug" → "Write a test that reproduces it, then make it pass"
- "Refactor X" → "Ensure tests pass before and after"
```

````
For multi-step tasks, state a brief plan:
```
1. [Step] → verify: [check]
2. [Step] → verify: [check]
3. [Step] → verify: [check]
```

Strong success criteria let you loop independently. Weak criteria ("make it
work") require constant clarification.
````

References

- Google Antigravity(안티그래비티, 반중력) 리뷰 - 브라우저 통합, Agent Manager, Gemini 3.0 등 활용법 @goddaehee.tistory.com
- 개발자의 중력을 없애다: 구글 Antigravity가 바꾸는 코딩의 미래와 실전 활용 전략 (Agent-first IDE) @all-lifes.com