Tutorial 07: Loop Agents - Iterative Refinement

Difficulty: advanced
Reading Time: 1.5 hours
Tags: advanced, loop-agents, iterative, refinement, quality-control
Description: Create self-improving agents using LoopAgent for iterative refinement, quality control, and progressive enhancement of outputs.

Tutorial 07: Loop Agents - Iterative Refinement with Critic/Refiner Patterns

Overview

Learn how to build self-improving agent systems using `LoopAgent`! This tutorial teaches you iterative refinement patterns - perfect for when quality matters more than speed. Build agents that critique their own work and keep improving until it's excellent.

🎯 **Working Implementation Available**: A complete, tested essay refinement system is available at `tutorial_implementation/tutorial07/` (https://github.com/raphaelmansuy/adk_training/tree/main/tutorial_implementation/tutorial07). The implementation includes comprehensive tests, documentation, and a user-friendly setup process.

🚀 **Quick Start**: Want to see it in action immediately? Jump to the Complete Working Code (#complete-code-reference) section below!

Prerequisites

- **Completed Tutorials 01-06** - Understanding of agents, workflows, and multi-agent systems
- **Installed ADK** - `pip install google-adk`
- **API key configured** - From Tutorial 01

Core Concepts

LoopAgent

The `LoopAgent` executes sub-agents **iteratively** (in a loop) for refinement and quality improvement. Unlike Sequential (runs once) or Parallel (runs concurrently), Loop runs the same agents **multiple times** until quality is sufficient or a limit is reached.

Key Characteristics:

- Executes sub-agents repeatedly in a loop
- Deterministic (not LLM-powered - just loop logic)
- MUST have termination conditions (prevent infinite loops!)
- Perfect for iterative refinement and self-improvement

The Critic → Refiner Pattern

The most common and powerful loop pattern:

```
1. Critic: Evaluates current quality
2. Refiner: Improves based on critique
3. REPEAT until quality sufficient or max iterations
```

Why This Works:

- Critic provides objective feedback
- Refiner focuses on applying improvements
- Separation of concerns (evaluate vs improve)

- Iterative approach improves quality over time

Termination Strategies

YOU must prevent infinite loops! Three strategies:

Strategy 1: Max Iterations (Safety Net)

```
loop = LoopAgent(
    sub_agents=[critic, refiner],
    max_iterations=5  # Stops after 5 iterations MAX
)
```

Always have this as a safety limit!

Strategy 2: Exit Tool (Smart Termination)

```
def exit_loop(tool_context: ToolContext):
    """Signal that refinement is complete."""
    tool_context.actions.end_of_agent = True
    return {"text": "Loop exited successfully. The agent has determined the ta

refiner = Agent(
    tools=[exit_loop],
    instruction="If critic says 'APPROVED', call exit_loop"
)
```

Allows early exit when quality is good!

Strategy 3: Combination (Best Practice)

Use BOTH - exit tool for early termination, max_iterations as safety:

```
loop = LoopAgent(
    sub_agents=[critic_with_approval, refiner_with_exit_tool],
    max_iterations=5  # Safety limit
)
# Loop exits when:
# - exit_loop called (quality good!) OR
# - 5 iterations reached (safety limit)
```

When to Use Loop Agents

Use `LoopAgent` when:

- ✔️ Quality improvement through iteration
- ✔️ Self-correcting systems (write → review → fix → repeat)
- ✔️ Retry logic with validation
- ✔️ Gradual refinement (each iteration improves)

Don't use when:

- ❌ Single pass is sufficient
- ❌ No clear improvement metric
- ❌ Speed is more important than quality

Use Case

We're building an **Essay Refinement System** that:

1. **Initial Writer** - Creates first draft (runs once)
2. **Refinement Loop** - Repeats until essay is excellent:
3. *Critic*: Evaluates essay quality, gives specific feedback
4. *Refiner*: Applies improvements OR signals completion

This demonstrates the classic critic → refiner loop pattern!

Step 1: Create Project Structure

```
mkdir essay_refiner
cd essay_refiner
touch __init__.py agent.py .env
```

Copy your `.env` file from previous tutorials.

Step 2: Set Up Package Import

essay_refiner/init**.py**

```
from . import agent
```

Step 3: Build the Loop-Based Refiner

essay_refiner/agent.py

```python
from __future__ import annotations

from google.adk.agents import Agent, LoopAgent, SequentialAgent
from google.adk.tools.tool_context import ToolContext

# ===== Exit Tool for Loop Termination =====
def exit_loop(tool_context: ToolContext):
    """
    Signal that the essay refinement is complete.
    Called by the refiner when critic approves the essay.
    """
    print(f"  [Exit Loop] Called by {tool_context.agent_name} - Essay approved
    tool_context.actions.end_of_agent = True  # Signal to stop looping
    # Return a minimal valid content part so the backend always produces a val
    return {"text": "Loop exited successfully. The agent has determined the ta

# ============================================================
# PHASE 1: Initial Writer (Runs ONCE before loop)
# ============================================================
initial_writer = Agent(
    name="InitialWriter",
    model="gemini-2.0-flash",
    description="Writes the first draft of an essay",
    instruction=(
        "You are a creative writer. Write a first draft essay on the topic "
        "requested by the user.\n"
        "\n"
        "Write 3-4 paragraphs:\n"
        "- Opening paragraph with thesis\n"
        "- 1-2 body paragraphs with supporting points\n"
        "- Concluding paragraph\n"
        "\n"
        "Don't worry about perfection - this is just the first draft.\n"
        "\n"
        "Output ONLY the essay text, no meta-commentary."
    ),
    output_key="current_essay"  # Saves to state
)

# ============================================================
# PHASE 2: Refinement Loop (Runs REPEATEDLY)
# ============================================================

# ===== Loop Agent 1: Critic =====
critic = Agent(
    name="Critic",
```

```
    model="gemini-2.0-flash",
    description="Evaluates essay quality and provides feedback",
    instruction=(
        "You are an experienced essay critic and teacher. Review the essay bel
        "and evaluate its quality.\n"
        "\n"
        "**Essay to Review:**\n"
        "{current_essay}\n"
        "\n"
        "**Evaluation Criteria:**\n"
        "- Clear thesis and organization\n"
        "- Strong supporting arguments\n"
        "- Good grammar and style\n"
        "- Engaging and coherent writing\n"
        "\n"
        "**Your Task:**\n"
        "IF the essay meets ALL criteria well (doesn't need to be perfect, jus
        "  Output EXACTLY this phrase: 'APPROVED - Essay is complete.'\n"
        "\n"
        "ELSE if essay needs improvement:\n"
        "  Provide 2-3 specific, actionable improvements. Be constructive and
        "  Example: 'The thesis is vague - make it more specific about X.'\n"
        "\n"
        "Output ONLY the approval phrase OR the specific feedback."
    ),
    output_key="critique"  # Saves feedback to state
)

# ===== Loop Agent 2: Refiner =====
refiner = Agent(
    name="Refiner",
    model="gemini-2.0-flash",
    tools=[exit_loop],  # Provide exit tool!
    description="Improves essay based on critique or signals completion",
    instruction=(
        "You are an essay editor. Read the critique below and take appropriate
        "\n"
        "**Current Essay:**\n"
        "{current_essay}\n"
        "\n"
        "**Critique:**\n"
        "{critique}\n"
        "\n"
        "**Your Task:**\n"
        "IF the critique says 'APPROVED - Essay is complete.':\n"
        "  Call the 'exit_loop' function immediately. Do NOT output any text.\
        "  This means your response should ONLY be the function call, nothing
```

```
        "\n"
        "ELSE (the critique contains improvement suggestions):\n"
        "  Apply the suggested improvements to create a better version of the
        "  Output ONLY the improved essay text, no explanations or meta-commen
        "  Do NOT call any functions when improving the essay.\n"
        "\n"
        "IMPORTANT: You must EITHER call exit_loop OR output improved essay te
        "Never do both in the same response."
    ),
    output_key="current_essay"  # Overwrites essay with improved version!
)

# ===== Create Refinement Loop =====
refinement_loop = LoopAgent(
    name="RefinementLoop",
    sub_agents=[
        critic,   # Step 1: Evaluate
        refiner   # Step 2: Improve OR exit
    ],
    max_iterations=5  # Safety limit - stops after 5 loops max
)

# ============================================================
# COMPLETE SYSTEM: Initial Draft + Refinement Loop
# ============================================================
essay_refinement_system = SequentialAgent(
    name="EssayRefinementSystem",
    sub_agents=[
        initial_writer,    # Phase 1: Write first draft (once)
        refinement_loop    # Phase 2: Refine iteratively (loop)
    ],
    description="Complete essay writing and refinement system"
)

# MUST be named root_agent for ADK
root_agent = essay_refinement_system
```

Code Breakdown

Execution Flow:

```
User: "Write an essay about climate change"
    ↓
Initial Writer (runs ONCE):
    → Writes first draft → state['current_essay'] = "Climate change is..."
    ↓
Refinement Loop (iterates):
    ┌──────────────────────────────────────┐
    | Iteration 1:                         |
    |   Critic: Evaluates draft            |
    |   → state['critique'] = "Thesis    |
    |      vague, add examples..."         |
    |   Refiner: Reads critique            |
    |   → Applies improvements             |
    |   → state['current_essay'] =         |
    |      improved version                |
    ├──────────────────────────────────────┤
    | Iteration 2:                         |
    |   Critic: Evaluates improved essay   |
    |   → state['critique'] = "Good,     |
    |      but conclusion weak..."         |
    |   Refiner: Improves conclusion       |
    |   → state['current_essay'] =         |
    |      even better version             |
    ├──────────────────────────────────────┤
    | Iteration 3:                         |
    |   Critic: Evaluates again            |
    |   → state['critique'] = "APPROVED  |
    |      - Essay is complete."           |
    |   Refiner: Sees APPROVED             |
    |   → Calls exit_loop() ✔              |
    └──────────────────────────────────────┘
    ↓
Loop exits (early termination - only 3 iterations!)
    ↓
Final Output: Refined essay from state['current_essay']
```

Key Patterns:

1. **State Overwriting**: `refiner` uses same `output_key` as `initial_writer`
2. Each iteration overwrites the essay with improved version
3. Critic always evaluates the LATEST version
4. **Exit Tool Pattern**:
5. Critic outputs special phrase "APPROVED..."

6. Refiner detects phrase and calls `exit_loop()`
7. `tool_context.actions.end_of_agent = True` signals stop
8. **Safety Net**: `max_iterations=5` prevents infinite loop if approval never comes

Step 4: Run the Essay Refiner

Navigate to parent directory and launch:

```
cd ..  # Go to parent of essay_refiner/
adk web
```

Open `http://localhost:8000` and select "essay_refiner".

Try These Prompts

Basic Essay:

```
Write an essay about the importance of education
```

Technical Topic:

```
Write an essay explaining how artificial intelligence works
```

Argumentative:

```
Write an essay arguing for renewable energy adoption
```

Creative:

```
Write an essay about the future of space exploration
```

Understanding Loop Execution

Open the **Events tab** to watch the iterative refinement:

1. **Event**: InitialWriter starts
2. **Event**: InitialWriter completes → first draft created
3. **Event**: RefinementLoop starts
4. **Iteration 1**:
5. Event: Critic starts
6. Event: Critic completes → feedback generated
7. Event: Refiner starts
8. Event: Refiner completes → essay improved
9. **Iteration 2**:
10. Event: Critic starts (evaluates improved version)
11. Event: Critic completes
12. Event: Refiner starts
13. Event: Refiner completes
14. **Iteration 3**:
15. Event: Critic starts
16. Event: Critic completes → outputs "APPROVED"
17. Event: Refiner starts
18. Event: Refiner calls exit_loop ✔️
19. **Event**: RefinementLoop completes (early exit!)

Notice: Loop can exit early (3 iterations) OR hit safety limit (5 iterations)

Expected Behavior

Example: "Write an essay about climate change"

```
User: Write an essay about climate change

[Initial Writer creates first draft]

First Draft:
Climate change is a problem. It affects the environment. We should do somethin
about it. Many scientists agree.

[Refinement Loop - Iteration 1]
Critic: "Thesis is too vague. Add specific examples of climate change impacts.
Strengthen the conclusion with concrete actions."

Refiner: [Improves essay]

Improved Draft (v2):
Climate change represents one of the most pressing challenges facing humanity,
with rising global temperatures causing unprecedented weather patterns and eco
disruption...

[Refinement Loop - Iteration 2]
Critic: "Much better! Consider adding transition sentences between paragraphs.
The conclusion could be stronger."

Refiner: [Improves transitions and conclusion]

Improved Draft (v3):
Climate change represents one of the most pressing challenges facing humanity.
[Better transitions added]
Therefore, immediate action combining policy changes, technological innovation
and individual responsibility is essential...

[Refinement Loop - Iteration 3]
Critic: "APPROVED - Essay is complete."

Refiner: [Calls exit_loop]

Final Output: [Displays refined version 3]
```

Result: 3 iterations produced a high-quality essay!

How It Works (Behind the Scenes)

Loop Mechanics:

1. **LoopAgent.sub_agents** = [critic, refiner]
2. **First iteration**:
3. Run critic → saves to state['critique']
4. Run refiner → reads critique, saves to state['current_essay']
5. **Next iteration**:
6. Run critic AGAIN → evaluates NEW state['current_essay']
7. Run refiner AGAIN → either improves OR calls exit_loop
8. **Continue** until:
9. `exit_loop()` called → `tool_context.actions.end_of_agent = True`
10. OR `max_iterations` reached

Tool Context Actions:

```
def exit_loop(tool_context: ToolContext):
    tool_context.actions.end_of_agent = True  # THIS stops the loop!
    return {"text": "Loop exited successfully. The agent has determined the ta
```

When end_of_agent=True, ADK stops the LoopAgent immediately.

State Overwriting Pattern:

- Initial writer: `output_key="current_essay"` → Creates
- Refiner: `output_key="current_essay"` → Overwrites each iteration
- Critic: Reads `{current_essay}` → Always gets latest version

Key Takeaways

✔️ **LoopAgent enables iterative refinement** - Quality improves over iterations

✔️ **Critic → Refiner is the pattern** - Separate evaluation from improvement

✔️ **MUST have termination** - exit_loop tool + max_iterations

✔️ **State overwriting** - Same output_key creates versioning

✔️ **Early exit saves time** - Don't waste iterations if quality is good

✔️ **Safety net essential** - max_iterations prevents infinite loops

✔️ **Tool escalation** - `tool_context.actions.end_of_agent = True` stops loop

Best Practices

DO:

- Always set `max_iterations` as safety net
- Use exit tool for intelligent early termination
- Keep loop simple (2-3 agents max)
- Test termination conditions thoroughly
- Use descriptive approval phrases (not just "yes")
- Monitor iteration count in Events tab

DON'T:

- Forget max_iterations (infinite loop risk!)
- Put too many agents in loop (complexity!)
- Assume loop will always exit early
- Use loops when single-pass is sufficient
- Make termination condition too strict (might never exit early)

Common Issues

Problem: "Loop runs all 5 iterations even though quality is good"

- **Solution**: Check refiner is correctly detecting approval phrase
- **Solution**: Verify exit_loop tool is properly called
- **Solution**: Check for typos in approval phrase matching

Problem: "Loop exits immediately on first iteration"

- **Solution**: Critic might be too lenient (always approving)
- **Solution**: Check critique instruction is properly evaluating

Problem: "Loop never exits early, always hits max"

- **Solution**: Critic might be too harsh (never approving)
- **Solution**: Refiner might not be calling exit_loop correctly
- **Solution**: Lower max_iterations to test faster

Problem: "State not updating between iterations"

- **Solution**: Check refiner has `output_key="current_essay"`
- **Solution**: Verify same key name used consistently

What We Built

You now have a self-improving essay system that:

- Writes initial drafts quickly
- Iteratively refines through critique and improvement
- Exits early when quality is sufficient
- Has safety limits to prevent infinite loops

And you understand how to build iterative refinement systems!

Real-World Applications

Loop Agents Are Perfect For:

- **Content Refinement**: Essays, articles, code, documentation
- **Quality Assurance**: Test generation, bug fixing, validation
- **Creative Iteration**: Image generation, music composition, design
- **Self-Correction**: Math problems, logical reasoning, planning
- **Retry Logic**: API calls with validation, data processing with error checking
- **Consensus Building**: Multi-reviewer approval systems

Next Steps

🚀 **Tutorial 08: State & Memory** - Learn session management and long-term memory

📖 **Further Reading**:

- Loop Agents Documentation (https://google.github.io/adk-docs/agents/workflow-agents/loop-agents/)
- Tool Context API (https://google.github.io/adk-docs/tools/function-tools/)
- Workflow Agents Overview (https://google.github.io/adk-docs/agents/workflow-agents/)

Exercises (Try On Your Own!)

1. **Different quality metrics** - Add grammar score, readability score
2. **Multiple critics** - Parallel critics for different aspects
3. **Adaptive max_iterations** - Adjust based on topic complexity
4. **Revision history** - Save each iteration to see progression
5. **User approval** - Add human-in-the-loop approval tool

Complete Code Reference

Working Implementation: See `tutorial_implementation/tutorial07/` (https://github.com/raphaelmansuy/adk_training/tree/main/tutorial_implementation/tutorial07) for a complete, tested version with comprehensive documentation.

🚀 **Ready to run the code?** The implementation is fully functional with 22 passing tests. Just follow the Quick Start instructions below!

Key Files:

- `essay_refiner/agent.py` (https://github.com/raphaelmansuy/adk_training/blob/main/tutorial_implementation/tutorial07/essay_refiner/agent.py) - Complete LoopAgent orchestration with critic-refiner pattern
- `tests/test_agent.py` (https://github.com/raphaelmansuy/adk_training/blob/main/tutorial_implementation/tutorial07/tests/test_agent.py) - 22 comprehensive tests covering all functionality

- `README.md` (https://github.com/raphaelmansuy/adk_training/blob/main/tutorial_implementation/tutorial07/README.md) - Detailed implementation guide and architecture overview
- `Makefile` (https://github.com/raphaelmansuy/adk_training/blob/main/tutorial_implementation/tutorial07/Makefile) - Development commands for testing and deployment

Quick Start with Working Code:

```
cd tutorial_implementation/tutorial07/
make setup  # Install dependencies
make test   # Run all tests (22 passing)
make dev    # Start development server
```

Congratulations! You've mastered iterative refinement with loop agents! 🎯[FLOW]📝

Generated on 2025-10-19 17:56:25 from 07_loop_agents.md

Source: Google ADK Training Hub