Summary of AI Agent

An AI tool that helps you do tasks on your computer. You can run it using the command line or terminal. It chats with you to understand what you need and creates a plan using the Gemini API key. It shows you the plan and asks if it's okay. If you agree, it asks for a command to run the task. If not, it asks what to change. If the command works, the task is done, and the tool stops.

Code Break Down

Importing modules

```
from google import genai
import subprocess
```

Importing modules which are required for the project.
genai connects to the Google Gemini Api key.
subprocess module in Python is a powerful standard library used for creating and managing additional processes by running system-level commands directly from a Python program. It essentially allows your Python code to interface with the operating system's shell and execute shell commands just as you would from a terminal

API Key

```
API_KEY = "API Key"
```

This variable stores the API key required for authenticating requests to the Gemini service.
API Key refers to our API Key.

Generating Task Plan

```
def generate_task_plan(task_description):
    try:
        client = genai.Client(api_key=API_KEY)
        response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=f"Create a step-by-step plan for this task: {task_description}"
        )
        return response.text.strip()
    except Exception as error:
        print(f"Error creating task plan: {error}")
        return None
```

Generates step by step by step plan for the user's Task.
client = genai.Client(api_key=API_KEY) creates a connection to the Gemini API.
response = client.models.generate_content(...) sends a request to the **gemini-2.0-flash** model with the user's task description.
task_description is the user's task which is to be performed.
Returning response which is generated by Api else returning error as can't generate output.

Refining the Task Plan

```
def refine_task_plan(task_description, changes):
    try:
        client = genai.Client(api_key=API_KEY)
        response = client.models.generate_content(
            model="gemini-2.0-flash",
             contents=f"The task plan needs improvement. Task: {task_description}, Requested Changes: {changes}"
        )
        return response.text.strip()
    except Exception as error:
        print(f"Error refining task plan: {error}")
        return None
```

Refines an existing task plan based on user feedback or requested changes.
response = client.models.generate_content(…) sends original task descripting along with the requested changes to API and returns it.

Executing Command

```
def execute_command(command):
    try:
        result = subprocess.run(command, shell=True, capture_output=True, text=True)
        if result.returncode == 0:
            print("Command ran successfully! ✅\nOutput:", result.stdout)
            return True
        else:
            print("Error running the command ❌:\n", result.stderr)
            return False
    except Exception as error:
        print(f"Failed to execute command: {error}")
        return False
```

Runs system-level shell commands (e.g., running a script, compiling code). This returns true if command runs successfully else returns false.
subprocess.run() executes the provided command in the system shell.
capture_output=True stores both the output (stdout) and error messages (stderr).
text=True ensures the output is returned as a readable string.

Main Function

```
def main():
    print("Welcome to the AI Task Assistant!")
```

This is entry point of the program
Step-1:User input for the task

```
 task_description = input(“What task do you need help with? “)
```

Talking user input from the user and assigning it in task_description.

Step-2:Create and display the task plan

```
print("\nGenerating task plan...")
task_plan = generate_task_plan(task_description)
if not task_plan:
    print("Could not create a plan. Exiting...")
    return
print("\nTask Plan:\n", task_plan)
```

Calls **generate_task_plan()** to create a step-by-step plan using the Gemini API. Exits the program if a task plan cannot be created.

Step-3:User approval

```
approval = input("Do you approve this plan? (yes/no): ").lower()
while approval != "yes":
    changes = input("What changes would you like? ")
    task_plan = refine_task_plan(task_description, changes)
    if not task_plan:
        print("Could not refine the plan. Exiting...")
        return
    print("\nUpdated Task Plan:\n", task_plan)
    approval = input("Do you approve this plan now? (yes/no): ").lower()
```

Asks the user to approve the task plan. If the user disapproves, prompts for changes and refines the plan using **refine_task_plan().**The loop continues until the user approves the plan.

Step-4:Command execution

```
command = input("Enter the command to run: ")
success = execute_command(command)
```

Prompts the user to enter the command associated with the task. Executes the command using **execute_command()**.

Step-5:If command fails, refine plan and retry

```
while not success:
    feedback = input("Task failed. Provide feedback to fix: ")
    task_plan = refine_task_plan(task_description, feedback)
    if not task_plan:
        print("Could not refine the plan. Exiting...")
        return
    print("\nRefined Task Plan:\n", task_plan)
    command = input("Enter the new command to retry: ")
    success = execute_command(command)
```

A Feedback Loop, If the command fails, asks the user for feedback on what went wrong. Refines the task plan using Gemini AI and retries execution with a new command.

Task Completion

```
    print("Task completed successfully! ✅")
```

Displays a success message when the task is completed.

Program Entry Point

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

Ensures the **main()** function runs when the script is executed directly.

Overall Workflow

Input Task: User describes the task.
Generate Plan: AI generates a step-by-step task plan.
Approval Loop: User approves or refines the task plan until satisfied.
Command Execution: Runs system commands based on the plan.
Retry on Failure: Refines the task plan and retries execution until successful.