Building an AI Agent with Python

Prepared by Fuad Habib

June 2025

1. INTRODUCTION

Mambo niaje? Hope you're ready for another exciting Python workshop! As you've already figured, in this workshop we will be building an AI agent with Python. And no this is not a bluff, you'll make an actual useful agent that can handle any task you provide it with. Some next level Tony Stark stuff (alright perhaps I am exaggerating a bit).

In this workshop you will build the foundations of how AI agents work, these fundamentals are the same ones used by major Companies such as OpenAI, Cursor and Anthropic with their AI agents.

The mystery behind them will be revealed, and hopefully by the end of this workshop something will light in you that will lead you to build amazing things. Excited? Let's get started.

1.1 PREREQUISITES

To make the most of this workshop, you should have:

- Familiarity with Python syntax. You don't have to be a Python guru, but you should at least be able to make simple functions (such as adding two numbers) with Python. You can check this Python in 30 minutes tutorial if you need to brush up a bit.
- A computer with Python installed. I mean this is a Python workshop of course.
- An internet connection.
- A desire to learn and experiment. If you're reading this, you probably already are.

2. SET UP YOUR DEV ENVIRONMENT

Every cook needs their kitchen prepared, and for a smooth learning experience we need to get setting up the environment out of the way.

2.1 Make sure Python is installed

Open your terminal/command prompt and type `python --version` to confirm if it is installed. Your Python version needs to be at least version 3.8 and above.

2.2 Create a Virtual Environment

It is always good practice to isolate your python projects into their own virtual environments. This prevents conflicts between package and library versions required by different python projects.

To create a virtual environment:

I. Navigate to the project directory. In other words just go to the folder where you want to have your project.
II. Open the terminal or command prompt and enter the following command to create a virtual environment:
    A. For Linux/Mac OS: `python3 -m venv venv`
    B. For Windows: `python -m venv venv`

    This will create a folder called `venv` with the virtual environment files.

III. To activate the virtual environment:
    A. For Linux/Mac: `source venv/bin/activate`
    B. For Windows command prompt: `venv\Scripts\activate`

Once the virtual environment is activated you'll see a `(venv)` on your terminal showing you that the virtual environment is active.

2.3 Installing Necessary Libraries

With the `venv` set we can proceed to install the necessary dependencies:

2.3.1 The `groq` python client

We will be Groq as our AI provider, mainly because they provide good AIs for a cheap price. The AI response speed is also a huge plus. If you are in the workshop, the API keys will be provided to you. Otherwise you can signup and create an account here: https://console.groq.com. To install groq simply run this command on your terminal:

```
pip install groq
```

2.3.2 A GUI Library: CustomTkinter

Although not necessary, I think it would be great if we had a nice GUI (a graphical user interface) instead of using the terminal. To install this simply run, and you guessed it:

```
pip install customtkinter
```

2.4 A Note About API Keys

API keys are sensitive credentials. If an API key is exposed, a bad actor could use it to make requests and guess who ends up paying the bill? **You!** More so, exposing keys makes you vulnerable to various attacks.

You can hard code them for this session, but the recommended way of managing keys is via environmental variables. Learn how to set and use them.

3. Interacting with Groq

With our dev environment set, let's see how we can communicate with LLMs via the Groq API. This will be the foundation of the Agent we will build.

3.1 A Quick Look At LLMs

At its core, Large Language Models are sophisticated text predictors. They are trained on large volumes of data and honestly nobody fully understands how they work. And we really don't need to for us to make use of them. All we know is that they mimic intelligence and we can use them for various tasks.

Groq provides us with a platform that allows us to access various LLMs. They also provide fast inference speeds. What this means is that when a request is sent to an AI via Groq, the response is generated very quickly. By the way do not confuse Groq (an LLM platform provider) with Grok (an AI developed by X/Twitter).

3.2 Sending Requests to Groq with Python

When communicating with an AI, we send messages in the form of roles. The messages we send usually has three roles:

- **System:** This is where we instruct the AI *to act as* something we want. For example: *You are an AI Assistant that knows how to code*.
- **User:** This represents the messages sent by humans like yourself (assuming you're not an alien)
- **Assistant:** This represents the messages sent by the AI.

These roles are sent via a `chat completion` interface, and since we have already installed the `groq` python library, let's see what this entails. Write the following lines of code:

```python
from groq import Groq

client = Groq(api_key="your-api-key-here")

def chat(message):
    chat_completion = client.chat.completions.create(
        messages = [{"role": "system", "content": "you are a helpful assistant"}, {"role": "user", "content": message}],
        model = "llama3-8b-8192",
        temperature=0.7,
        max_tokens=1024
    )

    return chat_completion

message = input("Your message: ")

response = chat(message)

print(response)
```

Explanation:

Line 1: You import the Groq client to be able to communicate with Groq

Line 3: You load the Groq client with your api key to a variable called `client`

Line 5: You define a function called chat, which accepts a `message` parameter

Line 6: Inside the chat function you create a chat completion by providing the messages. Note that you are setting the system role and also provide the actual message.

Line 8: We provide the AI model name, in this case its `llama3-8b-8192`

Line 9: We provide the `temperature`, this basically means how creative we want the AI to be. Higher temperature means more creativity, lower temperature means a more predictable response.

Line 10: The max number of tokens we want to receive from the AI.

Line 15: We get the human's message

Line 17 and 19: We get the response and print it. The response will be something like this:

```
ChatCompletion(id='chatcmpl-13ced6b9-3593-4e40-816e-593dc4534975',
choices=[Choice(finish_reason='stop', index=0, logprobs=None,
message=ChatCompletionMessage(content="Hello! It's nice to meet you. Is there
something I can help you with or would you like to chat? I'm here to assist you with
any questions or tasks you may have.", role='assistant', executed_tools=None,
function_call=None, reasoning=None, tool_calls=None))], created=1749754389,
model='llama3-8b-8192', object='chat.completion', system_fingerprint='fp_8b7c3a83f7',
usage=CompletionUsage(completion_tokens=40, prompt_tokens=21, total_tokens=61,
completion_time=0.07036377, prompt_time=0.005539158, queue_time=0.12874749300000002,
total_time=0.075902928), usage_breakdown=None, x_groq={'id':
'req_01jxjqt8jgf06ajk7xk6qx1ecw'})
```

A lot of unnecessary information for us, so lets clean it to get the actual message. Rewrite line 19 as follows:

```
print(response.choices[0].message.content)
```

Now what will be printed is something like this:

```
Hello! It's nice to meet you. Is there something I can help you with or would you
like to chat? I'm here to assist you with any questions or tasks you may have.
```

That's much better.

4. Designing the Agents Brain

Now that we can communicate with Groq, let's start giving it the ability to act as a proper Agent. We will do this by giving it *tools* to act as an agent. What are tools you ask? Read ahead.

4.1 LLM Tools (aka Function Calling)

If you ever heard of MCP (Model Context Protocol), function calling etc etc you might think of them as some crazy API standard. I wouldn't blame you, it took me a while to understand what they are and I only understood *after* I made my own agent.

So do you know what LLM tools are? Functions. And they are not like some fancy functions, they are just regular functions.

Let's say you want to create an AI agent that can calculate 2 numbers or let's say you want to make an AI Calculator MCP. The first step is to create the calculator function:

```
def calculate(x, y):

    return x + y
```

The next step is to tell the AI that, "Yo mr AI! There's this function I have for calculating numbers. Use it whenever a user asks to add two numbers."

That's it, congratulations, you have made an AI Agent and an MCP Server that adds two numbers. Pretty silly right?

It is silly and yet it is very powerful and I'll show you why.

4.2 Defining the Capabilities of Our Agent

The AI Agents will only work using the tools we give them. If we give them a tool to calculate then that agent will only know how to calculate. Note that I am using the word tool, but in reality they are just functions. *Functions are AI tools*.

We will build an Agent that can help us create folders, create files and read files. So basically we will need to create three functions:

a. A function to create folders
b. A function to create files
c. A function to read files

Giving the AI these three simple functions will allow us to do awesome things as you will see. So how does the AI actually use the tools? It basically follows these steps:

Step 1: A user sends message like please create a folder called 'Test'

Step 2: The message and the available functions will be sent to the AI. The AI will check the message and be like "Hmmm, this user wants me to create a folder. What tools do I have to do this? Wait I have this tool called `create_folder`, I can use that to create the folder". Then the AI will respond with a tool name (ie the function the program should run).

Step 3: The program will receive the message from the AI and say, “The AI wants me to run the function to create a folder, let me run that”, it will run the function and send the response of the function to the AI.

Step 4: The AI will receive the response of running the function and then format the message, so it will reply with something like “I have successfully created the folder ‘Test’ for you.” or if let’s say the program failed then the AI would respond with “I cannot create the folder ‘Test’”.

If the steps above still don’t seem clear to you then don’t worry, we will create an Agent that does this and you’ll understand how it works.

4.3 Crafting Tools for the AI Agent

So let’s make the tools, one for creating folders, one for creating files and one for reading files for our Agent. Let’s start with reading files.

4.3.1 Reading Tool

Tools are just, so what we want is to create a function to read files. Write the following lines of code:

```
def read_file_tool(file_name: str, path: str = "./") -> str:
    try:
        full_path = os.path.join(path, file_name)
        with open(full_path, 'r') as file:
            return file.read()
    except Exception as e:
        return f"Failed to read from file: {str(e)}"
```

This function has two parameters; `file_name` and `path`. It basically looks for the file in the provided directory and returns its contents. If there is any error it will return the error message.

Now in order for the AI to be able to use it, we first need to tell it that the function exists. We do this by creating a variable called `tools` and add our tools description there. Write the following lines of code:

```python
tools = [
    {
        "type": "function",
        "function": {
            "name": "read_file_tool",
            "description": "Use it to read a file. It will return the contents of the file or an error message",
            "parameters": {
                "type": "object",
                "properties": {
                    "file_name": {
                        "type": "string",
                        "description": "The name of the file to read",
                    },
                    "path": {
                        "type": "string",
                        "description": "The path to the file to read. Defaults to the current directory",
                    }
                },
                "required": ["file_name"],
            },
        }
    }
]
```

We're basically telling the AI what the function is and how to use it. And finally we should modify the chat function as follows:

```python
def chat(message):
    # Step 1: Send the message to the AI with the available tools
    conversation_history = [{"role": "system", "content": "you are a helpful assistant"}, {"role": "user", "content": message}]
    ai_response = client.chat.completions.create(
        messages = conversation_history,
        model = "llama-3.3-70b-versatile",
        tools=tools,
        tool_choice="auto",
        temperature=0.7,
        max_tokens=4096
    )

    # Step 2: Get the response message and add it to our conversation history

    response_message = ai_response.choices[0].message

    conversation_history.append(response_message)

    # Step 3: Check if the AI wants us to run any functions

    tool_calls = response_message.tool_calls

    available_functions = {
        "read_file_tool": read_file_tool,
    }

    if tool_calls:
        for tool_call in tool_calls:
            function_name = tool_call.function.name
            function_to_call = available_functions[function_name]
            function_args = json.loads(tool_call.function.arguments)
            function_response = function_to_call(**function_args)

            conversation_history.append(
                {
                    "role": "tool",
                    "content": str(function_response),
                    "tool_call_id": tool_call.id,
                }
            )

    # Step 4: Send the function response to AI for and get a final response

    final_ai_response = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=conversation_history,
        tools=tools,
        tool_choice="auto",
        max_completion_tokens=4096
    )

    return final_ai_response
```

Spend some time understanding how it works. You're basically sending a message (eg: read the file demo.txt) -> The AI will respond with what functions to use to fulfill your request -> The program then calls whatever function the AI wants it to run. Then finally the function response is sent to the AI which will reply to us with a more human friendly message.

To test this, create a file called `test.txt` in the same directory and add some random texts. Then run the program to see if it can read it:

```
💀 evilMonkey …/agents venv python tut.py
Your message: what is inside the file test.txt in the current dir?
I have read the file test.txt in the current directory. The file contains a poetic description of a calm and peaceful night scene, with the
 moon, stars, trees, and wind all contributing to a serene atmosphere. The poem explores the idea of the night as a time for secrets, myste
ries, and the interplay between light and darkness.
```

Nice! It can actually read the file. Here is the full code for reference:

```python
import json
import os

from groq import Groq

client = Groq(
    api_key=os.environ.get("GROQ_API_KEY"),
)

def read_file_tool(file_name: str, path: str = "./") -> str:
    try:
        full_path = os.path.join(path, file_name)
        with open(full_path, 'r') as file:
            return file.read()
    except Exception as e:
        return f"Failed to read from file: {str(e)}"

tools = [
    {
        "type": "function",
        "function": {
            "name": "read_file_tool",
            "description": "Use it to read a file. It will return the contents of the file or an error message",
            "parameters": {
                "type": "object",
                "properties": {
                    "file_name": {
                        "type": "string",
                        "description": "The name of the file to read",
                    },
                    "path": {
                        "type": "string",
                        "description": "The path to the file to read. Defaults to the current directory",
                    }
                },
                "required": ["file_name"],
            },
        }
    }
]

def chat(message):
    # Step 1: Send the message to the AI with the available tools
    conversation_history = [{"role": "system", "content": "you are a helpful assistant. You will always mention what tasks you have done."}, {"role": "user", "content": message}]
    ai_response = client.chat.completions.create(
        messages = conversation_history,
        model = "llama-3.3-70b-versatile",
        tools=tools,
        tool_choice="auto",
        temperature=0.1,
        max_tokens=4096
    )

    # Step 2: Get the response message and add it to our conversation history

    response_message = ai_response.choices[0].message

    conversation_history.append(response_message)

    # Step 3: Check if the AI wants us to run any functions

    tool_calls = response_message.tool_calls

    available_functions = {
        "read_file_tool": read_file_tool,
    }

    if tool_calls:
        for tool_call in tool_calls:
            function_name = tool_call.function.name
            function_to_call = available_functions[function_name]
            function_args = json.loads(tool_call.function.arguments)
            function_response = function_to_call(**function_args)

            conversation_history.append(
                {
                    "role": "tool",
                    "content": function_response,
                    "tool_call_id": tool_call.id,
                }
            )

    # Step 4: Send the function response to AI for and get a final response

    final_ai_response = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=conversation_history,
        tools=tools,
        tool_choice="auto",
        temperature=0.1,
        max_completion_tokens=4096
    )

    return final_ai_response

message = input("Your message: ")

response = chat(message)

print(response.choices[0].message.content)
```

Great. Now let’s add the remaining tools.

4.3.2 Creating Folder and File Tool

Add these two functions below the `read_file_tool` function:

```python
def create_folder_tool(folder_name: str):
    try:
        full_path = os.path.join("./", folder_name)
        os.makedirs(full_path, exist_ok=True)
        return "folder created successfully"
    except Exception:
        return "failed to create folder"

def create_file_tool(file_name: str, content: str):
    try:
        full_path = os.path.join("./", file_name)
        with open(full_path, 'a') as file:
            file.write(content)
        return f"Content written successfully to '{full_path}'"
    except Exception as e:
        return f"Failed to write to file: {str(e)}"
```

And then update the `tools` variable to be as follows:

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "read_file_tool",
            "description": "Use it to read a file. It will return the contents of the file or an error message",
            "parameters": {
                "type": "object",
                "properties": {
                    "file_name": {
                        "type": "string",
                        "description": "The name of the file to read",
                    },
                    "path": {
                        "type": "string",
                        "description": "The path to the file to read. Defaults to the current directory",
                    }
                },
                "required": ["file_name"],
            }
        }
    },
        {
        "type": "function",
        "function": {
            "name": "create_folder_tool",
            "description": "Use it to create folders. It will return a success or failure message",
            "parameters": {
                "type": "object",
                "properties": {
                    "folder_name": {
                        "type": "string",
                        "description": "The folder to create",
                    }
                },
                "required": ["folder_name"],
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "create_file_tool",
            "description": "Use it to create files. It will return a success or failure message",
            "parameters": {
                "type": "object",
                "properties": {
                    "file_name": {
                        "type": "string",
                        "description": "The folder to create",
                    },
                    "content": {
                        "type": "string",
                        "description": "The content to write to the file",
                    },
                },
                "required": ["file_name", "content"],
            }
        }
    }
]
```

Finally update the `available_functions` variable to include the new tools:

```python
available_functions = {
    "read_file_tool": read_file_tool,
    "create_file_tool": create_file_tool,
    "create_folder_tool": create_folder_tool,
}
```

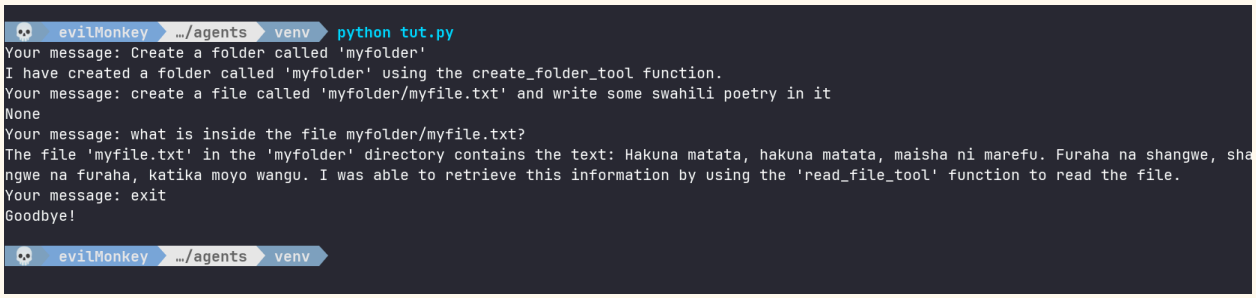
Now if you test the application, you'll see that you can create files and folders:

And there you have it, you have a full blown working AI Agent. Pat yourself on the back if you've reached this far.

5. Exercise

5.1 Wrap It In A Loop

You'll notice that you have to run the program each time to communicate with it. It works but it's not ideal. Therefore you are tasked to wrap the communication with the AI Agent with loop, modify the code so that you have something like this:



6. Conclusion

You now have the foundations of how to make AI Agents. Please do note that even though this is exciting, LLMs at their current state are not perfect. **THEY WILL MESS UP IF YOU GIVE THEM TOO MUCH POWER.** Therefore only give them non destructive tasks.

Thank you for reading this tutorial. For any questions, comments or suggestions feel free to reach out to me:

Github: https://github.com/AvicennaJr

Email: fuad@subcode.africa

Twitter: @AviTheDev