What are Tools?

Tools are like **helping hands** for an AI agent. They allow the agent to **do something** instead of just talking.

Imagine your agent is like a smart assistant. Sometimes, it needs to:

- Look something up on Google
- Do a calculation
- Find a file
- Call another expert

It uses **tools** to do these things.

1. Hosted Tools (Pre-made tools)

These are tools already created by OpenAI. You just turn them on — no need to code anything.

Example:

User: "What's the latest news about the weather in Karachi?"

The agent uses the **web search tool** (hosted by OpenAI) to search the internet and find the answer.

You don't need to write code — OpenAI already provides this tool.

2. Function Calling (Your own Python functions)

You can write your **own tool** using Python. The agent can then use your function like a tool.

Example:

You write a function:

```
def get_calories(food: str) -> str:
    "Returns calories in a given food item."
```

Then the user says: "How many calories are in an apple?"

The agent sees that your tool can answer this and calls your function with `"apple"`.

It will respond with something like: "An apple has around 95 calories."

The agent understands your function automatically, including:

- What input it needs (food: str)
- What it does (from the description/docstring)

You don’t need to manually explain your function — the SDK reads the information from your code.

3. Agents as Tools (One agent uses another)

Sometimes, you want **one agent to ask another agent** for help, without giving full control.

Example:

You have:

- A **main agent**: Helps with general questions
- A **math agent**: Only solves math problems

User: "What is 23 multiplied by 45?"

The **main agent** sends this task to the **math agent** (as a tool), gets the result, and replies:

"23 multiplied by 45 is 1035."

You didn’t hand over the full conversation. One agent just helped the other quietly, like a teammate whispering advice.

This is useful in **multi-agent systems** where each agent is an expert in one thing.

Extra Feature: Automatic Argument and Docstring Parsing

When you write a Python function like:

```
def greet(name: str) -> str:
    "Greets the user with their name."
```

The Agent SDK will:

- Read that `name` is a string input
- Understand what the function does from the description

So the agent can decide: “Oh, the user gave a name — I can call this function!”

You don’t need to manually describe the function; the SDK figures it out.

Handling Errors (So your agent doesn't break)

Sometimes users give wrong or incomplete input.

Example:

You have a function that gets weather by city.

User: "What's the weather?"

But they didn’t give a city.

Without error handling, your function might crash.

With good error handling, your function says:
“Please tell me the name of the city.”

The agent then responds to the user:
“Could you let me know which city you’re asking about?”

This makes your agent friendly and professional — not confusing or broken.

Final Real-Life Example

Let’s say you build a **study assistant agent**.

- It uses a **function tool** to create a daily study plan
- It uses a **hosted tool** to search online for good study resources
- It uses another **agent as a tool** to explain hard topics (like a science tutor)

Together, your agent becomes very smart — because it has tools to act, not just speak.