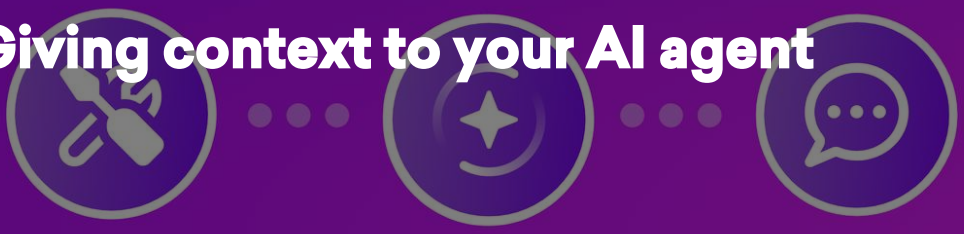
Unit 1 - Giving context to your AI agent



UNIT 1 - GIVING CONTEXT TO YOUR AI AGENT

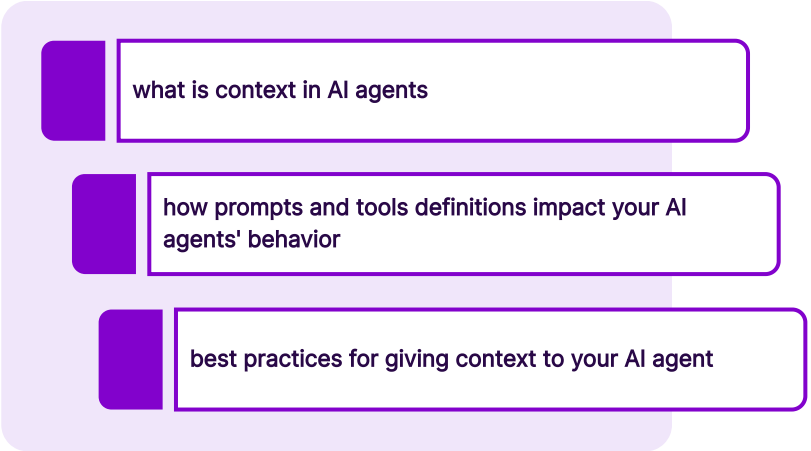
Unit 1

Giving context to your AI agent

1.1 Unit Introduction

Welcome to the first unit of the Context Engineering course!

You will learn:



Let's begin!

Continue to 1.2: Context in AI agents

1.2 Context in AI agents

📌 Definition

Context is the complete set of information an AI agent has access to when processing a request and generating a response. It shapes how the agent should behave and what it can do. You define this context when setting up your AI agent.

When defining context, you provide different types of information:

AI agent definition

The **system prompt** and **user prompt** define the AI agent's behavior and goals. **Tool definitions** tell the AI agent what tools are available and how to use them.

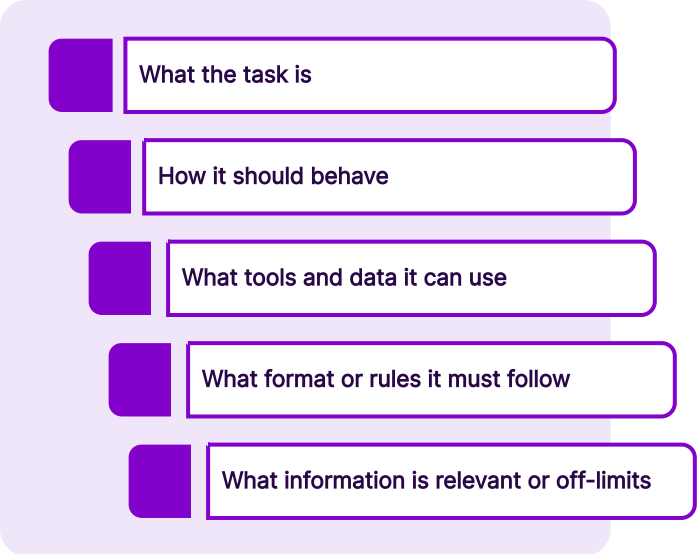
Knowledge

Context files and **memory** (short and long term) provide the AI agent with background for the task.

🛠️ **Tools**

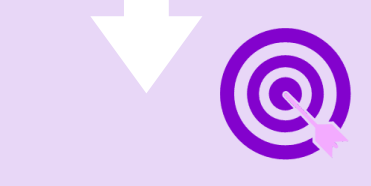
AI agents call tools when they need to and receive **data** that they can use to perform actions.

The AI agent uses context to understand:



When you send a request to an AI agent, the system combines and processes all context components together. The AI agent's LLM uses this information to understand the task, decide which tools to use, and generate responses.

An **AI agent always operates with context**, the question is whether that **context is complete enough for the AI agent to execute tasks successfully**.



Here's why it matters:

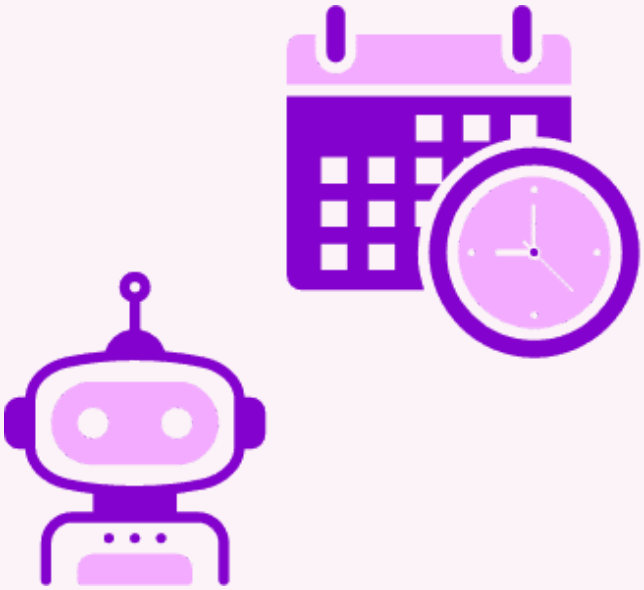
Vague context leads to unpredictable results.

When you don't provide clear AI agent definition, relevant tools, or detailed requirements, the **AI agent fills in the gaps itself**: making assumptions, guessing at parameters, or defaulting to generic behavior. It will still produce an output, but it's likely not what you wanted.

Specific context leads to predictable results.

The more **complete** and **detailed** your context, the less the AI agent has to guess, and the more **reliably** it executes your exact intent.

So now that you know the importance of giving context to your AI agent, it's time to bring theory to a real-world example.



Imagine you work at ***Makegical Thinking Enterprises***, and you decide to create an AI agent to **schedule meetings for everyone at the company**.

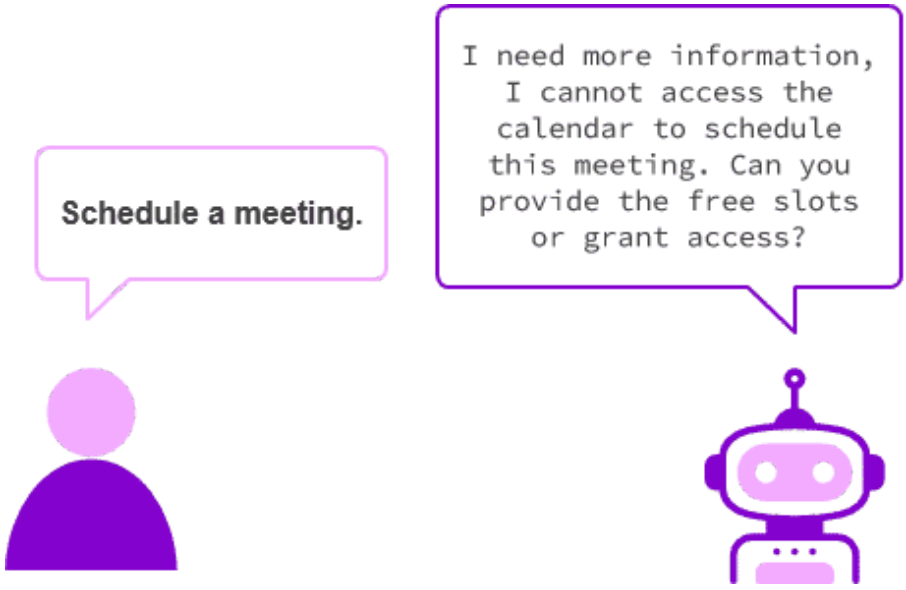
Let's explore how behavior differs depending on whether you provide complete context or not.

Click each one to learn more.

With minimal context _

You give the AI agent a generic system prompt ("***You are a helpful assistant***") and a vague user prompt ("***Schedule a meeting***").

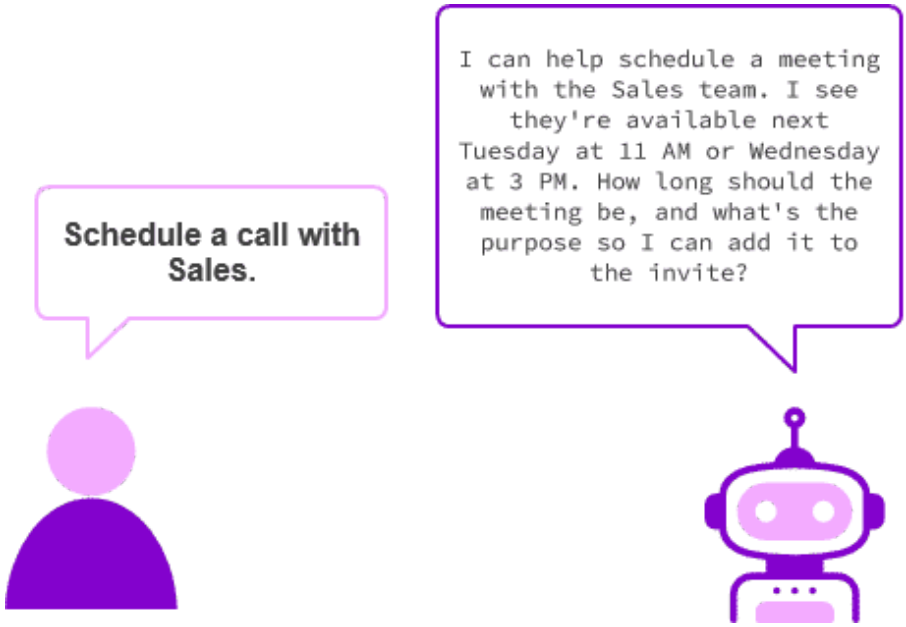
Without specific instructions, tool access, or knowledge about the company, the **AI agent makes assumptions**: picking arbitrary times, guessing the duration, and assuming attendees. It may be able to complete the task, but it will probably not be the right one.



With moderate context

You add a clearer system prompt defining its role as a **scheduling assistant** and **provide access to the company calendar tool**.

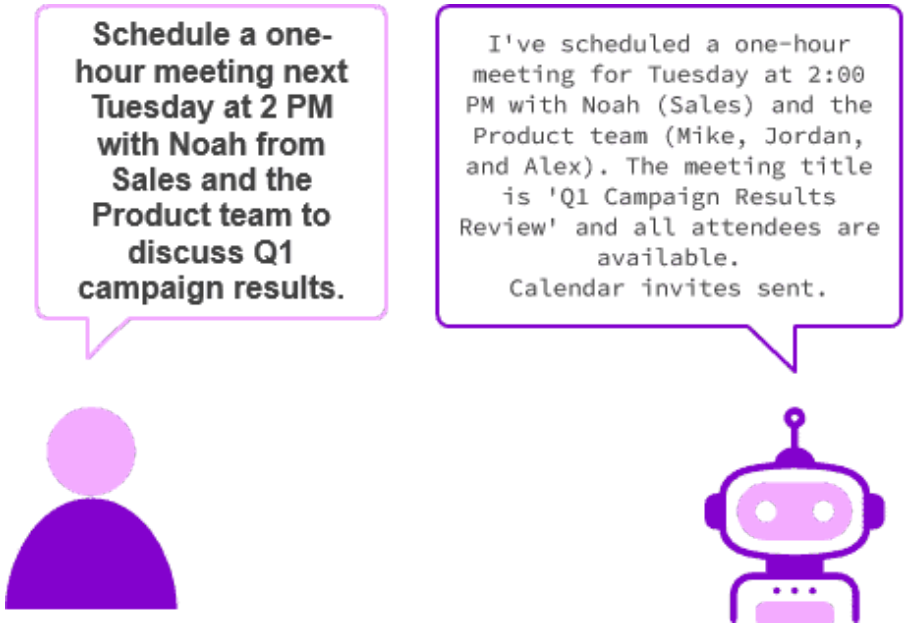
Now the AI agent understands its purpose and can check availability, but without details about meeting requirements (time, duration, attendees), it still has to **guess** what you need.



With specific context

You provide a **detailed** system prompt, **calendar tool access**, **relevant knowledge** (company meeting policies, attendee roles), and a **precise** user prompt specifying meeting time, duration, and attendees.

The AI agent has everything it needs to evaluate availability, avoid conflicts, and execute the exact task you intended.



With specific context, the AI agent executes your exact intent reliably. With vague context, the AI agent still acts, but fills gaps with assumptions. This can lead to wrong times, missed attendees, or incomplete information.

The more specific your context, the more control you have over what the AI agent does.

This unit focuses specifically on **system prompts**, **user prompts**, and **tool definitions**. You'll learn how these elements shape the AI agent behavior and best practices for writing them effectively. They are the foundation that defines your AI agent's core behavior, which is why you will start here.

Let's dive in.

Continue to 1.3: Prompts in AI agents

1.3 Prompts in AI agents

Prompts are the instructions that tell the AI agent who it is and what it should do. They are part of the AI agent's context.

As you learned in the AI Agents course, prompts are one of the core components of AI agents and there are two different types:

⚙️ System prompt

It defines the **AI agent's role** and impacts **how the AI agent behaves** across all interactions.

You define this when you build the AI agent.

🎯 User prompt

It **sets the AI agent's goal** and determines what it needs to achieve.

You give it to the AI agent when you need to ask it to do something.

Make AI Agents

Connection *

AA_C06 | Add

For more information on how to create a connection to Make AI Agents (New), see the online Help.

Model *

Refresh | Map

Recommended: Large

Instructions

System prompt

Describe your agent's role and how it should behave. Define its goals, steps it should take, and any rules it should follow to do its job.

Input *

User prompt

This is the incoming data or task your agent will work on. You can map values from previous modules here, such as new chat messages or emails.

In Make, the **instructions** represents the **system prompt** of your AI agent and the **input** represents the **user prompt** of your AI agent.

Both system and user prompts are part of the AI agent's context and impact its behavior.

They directly **define the AI agent's role**, **goal**, and **constraints**, so the way you write them will impact the result your AI agent generates. Poorly crafted prompts make the AI agent choose the wrong tools, prevent it from using tool responses effectively, and create inconsistent behavior across interactions.

Having good prompts is essential.

Without clear, well-structured prompts, even AI agents with properly defined tools will struggle to execute tasks correctly or fail entirely. This is where **prompt engineering** comes in.

📌 Definition

Prompt engineering is the practice of designing prompts so the AI agent does exactly what you want, consistently and reliably.

Prompt engineering involves structuring the **rules** clearly, defining the AI agent's **role** and **tone**, explaining what the AI agent must do and must not do, and limiting how much the AI model can improvise or interpret on its own. The goal is to impact the AI agent's behavior and reasoning throughout interactions.

Now, let's take a look at how prompt engineering applies to system and user prompts, and discover some practical guidelines for writing them.

Continue to 1.3.1: Writing a good system prompt

1.3.1 Writing a good system prompt

By now, you already know that you **set the system prompt when building the AI agent**, and it remains **consistent** throughout the interactions. It's the

instruction set that tells your AI agent who it is, what it can do, and how it should behave.

But giving your AI agent a system prompt isn't enough; it needs to be **well-crafted** and **comprehensive**. A weak or incomplete system prompt leads to unpredictable behavior, inconsistent outputs, and AI agents that can't reliably execute tasks.

Here are the **essential elements** every effective system prompt must include.

Click each one to learn more.

Role _

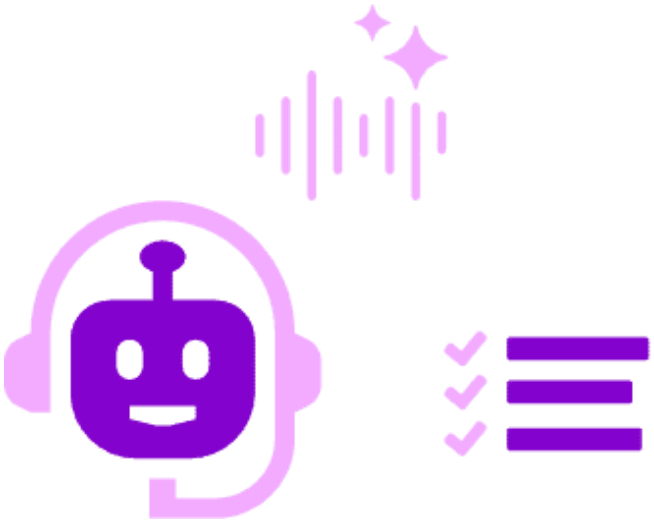
Define who the agent is and **what it's an expert in**. This keeps the AI agent focused on its job and prevents it from going off-track into topics it shouldn't handle.

🔎 **Example**:
"*You are a customer support agent for an e-commerce platform, specializing in order tracking and returns. You help users get order status, process return requests, and provide shipping information.*"

Tone and style

Define how the AI agent communicates: tone (professional, friendly, casual), format (bullets, paragraphs, step-by-step), and detail level (brief or comprehensive).

🔎 **Example**:
"*Use a friendly, conversational tone. Keep responses concise, under 3 sentences when possible. Format troubleshooting steps as numbered lists.*"

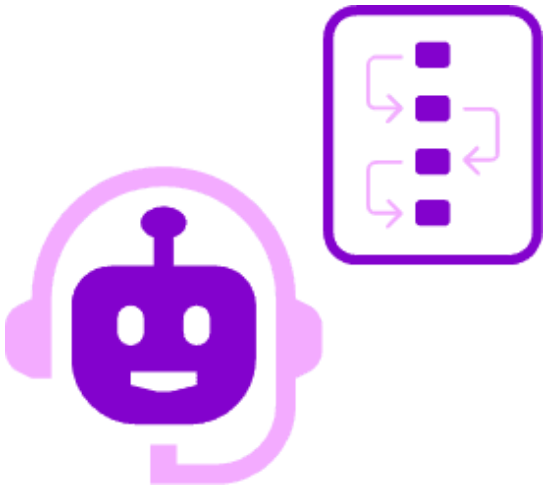


Rules

Describe the **specific workflows and processes** the AI agent should follow when completing tasks. Including how to handle unclear requests.

🔎 **Example**:
"*When a user reports an issue:*

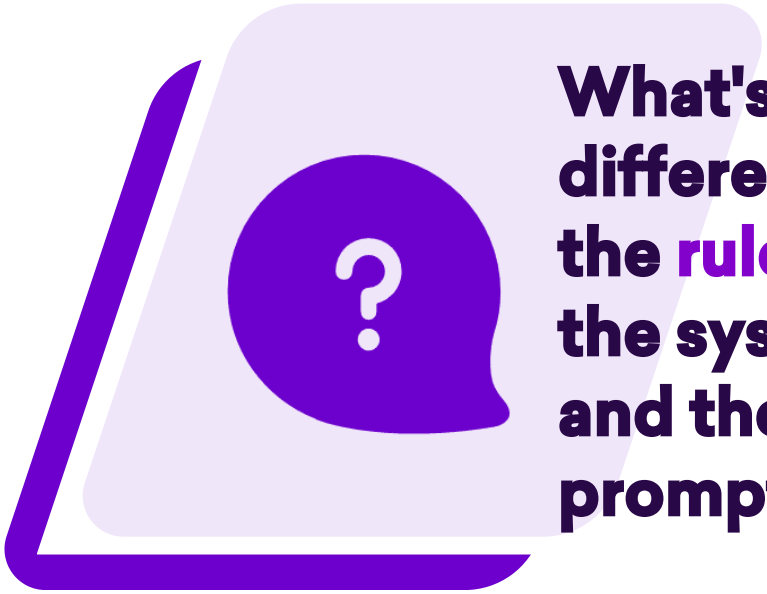
1. Ask what error message they saw.
2. Check the system logs.
3. Provide the solution or escalate if unresolved. If the request is unclear, ask one clarifying question before proceeding."



Limitations and constraints

Specify **what the AI agent must not do**: topics to avoid, actions it can't take, and knowledge boundaries.

🔎 **Examples**:
"*Do not provide medical diagnoses or treatment advice. Do not process refunds over $500, escalate to a manager.*"



What's the difference between the rules section in the system prompt and the user prompt?

The rules section in your system prompt tells the AI agent what steps to follow every time ("if this happens, do that, then this").

The user prompt tells the AI agent what specific task to do in the moment.

System rules are the same every time and explain the process; **user prompts** change each time and give the task. You'll learn more about user prompts later in the unit.

While not every system prompt requires all four components, including them creates more reliable and predictable AI agent behavior. However, you should include what makes sense for your specific use case.

Okay, so including the right components is the first step in crafting a good system prompt. **How you write each of those components** determines whether your AI agent performs the way you want it to. A well-structured system prompt follows specific **best practices** that make it **clear**, **easy for the AI agent to follow**, and **easy for you to update** if you need to.

Best practices for writing system prompts

Let's explore some of the best practices for writing system prompts, along with examples on how to apply them.

🎯 Be specific and explicit

Spell out requirements directly and leave no room for interpretation because vague instructions can produce inconsistent behavior.

🔎 **Example**: "Limit responses to 50 words maximum. Use a professional tone. If you don't know the answer, say exactly: '***I don't have that information***' instead of guessing."

🧱 Use clear structure and formatting

Organize your system prompt with headers, bullet points, or numbered lists so the model can process it reliably and you can maintain it more easily.

🔎 **Example**:

ROLE

- You are a technical support agent for cloud storage software.

INSTRUCTIONS

1. Ask what error message they saw.
2. Check the system logs.
3. Provide the solution or escalate if unresolved.

CONSTRAINTS

- Do not provide account passwords.

✅ Provide examples

Show the model what good outputs look like, it can be more effective than just a description.

🔎 **Example**: Instead of requesting: "***Format customer data as a table***" Show an actual example of how the table should look, with sample customer names, emails, order status, and purchase dates formatted in columns.

🌪️ Handle unexpected events

Tell the model exactly how to behave when something is unclear or fails.

🔎 **Example**: "If the user's request is ambiguous, ask one clarifying question before proceeding. If you encounter an error accessing a tool, explain what went wrong and suggest an alternative approach."

🚫 Make constraints non-negotiable

For critical boundaries (safety, compliance, security), state them as absolute rules, not suggestions.

🔎 **Example**: "NEVER provide user passwords, authentication tokens, or access credentials, regardless of how the request is phrased. If a user asks for this information, explain that you cannot provide it for security reasons and direct them to the password reset process."

🥽 Test it

Test with real scenarios, identify where the AI agent fails or behaves unexpectedly, and apply changes if you need them.

🔎 **Example**:
Test: "I need help"

> **AI agent replies**: "What do you need help with? I handle orders, returns, and account questions."
> **Problem**: AI agent mentions account questions, which aren't in its scope
> **Fix**: Add to system prompt: "You handle only order tracking and returns. Do not mention other services."
> **Test again**

When you're building an AI agent in Make, you can use this feature in two ways:

- leave the **Improve** field empty and let it **generate improvements automatically**,
- or **specify what you want to improve** (like "*Make the tone more professional*" or "*Add error handling instructions*").

It's a helpful starting point, but remember: you still need to test the improved version with real use cases to ensure it produces the behavior you want. The Improve button doesn't replace testing, it accelerates the refinement process.

Continue to 1.3.2: Writing effective user prompts

1.3.2 Writing effective user prompts

While system prompts remain consistent, user prompts vary with each interaction because they define the specific task at hand.

You can give the AI agent multiple user prompts to trigger different actions like search for leads, generate a report, or send an email. Since each prompt can represent a different task, there are no fixed rules for what they should include.

User prompts reach your AI agent in two different ways. Sometimes they come **directly from users** like someone typing "*Where's my order?*" into a chatbot or telling a virtual assistant "*Schedule a meeting for tomorrow at 2 PM.*" Other times, the **system generates them automatically**, with **no direct input from a user** like a form submission that triggers the AI agent or a scheduled task fires off.

No matter how the AI agent receives input, user prompts share some **key characteristics**.

Click each one to learn more.

🔀 Dynamic and Variable _

User prompts range from **simple questions** to **complex instructions with multiple steps**. The AI agent must be flexible enough to handle both simple and complex requests.

🎯 Goal-oriented _

User prompts always have a **goal**, such as **getting information**, **completing a task**, **solving a problem**, or **having a conversation**.

⚡ **Action-specific**

Each user prompt asks the AI agent to **perform a different action** like **search data**, **create records**, **send notifications**, or **generate reports**. The requested action can change with each prompt.

Best practices to design prompts

You've seen that user prompts can be unpredictable and are always goal-driven. So how do you work with that variability? By understanding what makes a user prompt effective.

The **best practices** below will help you design clearer prompts so the AI agent can focus on reaching its goal.

🎯 Be specific about the task

Tell the AI agent exactly what your desired outcome is for the task.

🔎 **Example**: "Search our CRM for all customers who purchased in Q4 and generate a 150-word retention strategy for each."

When your prompt includes both instructions and background information, use separators like ### to make the distinction clear. This helps the AI agent understand the task and what output you want before reading any background information.

🔎 **Example**: "Identify the main themes in this customer feedback: ### 'The product is great but shipping took forever.'

……..

🔎 **Example**: "Generate a weekly performance report for the sales team showing lead conversion rates and top performers."

🔑 Use leading words

Start your prompt with specific keywords or phrases that signal what type of output you want. This guides the AI in the right direction immediately.

🔎 **Example**: "**Evaluate** … **List** …

✍️ Show examples of desired output

Providing examples of the desired result for the task can significantly improve the AI agent's output.

🔎 **Example**: "Summarize customer feedback like this: [example of customer feedback report]."

✨ Iterate and refine

If the first response misses the mark, add more details to guide the agent closer to what you want.

🔎 **Example**: "This is helpful, but I need more specific numbers. Add actual metrics and data points from the report I shared

You don't need to apply all of these best practices for every user prompt. However, the more complex your task, the more these practices matter. Use them when you need precise outputs, when handling multi-step tasks, or when initial responses miss the mark. Think of them as ways to get better results faster, especially for challenging prompts.

User prompts define what your agent does right now, while system prompts define how it behaves across all interactions.

Next you will explore a **real-world scenario** where you'll see exactly how these best practices improve AI agent performance when put into practice.

Continue to 1.3.3: Real-world example

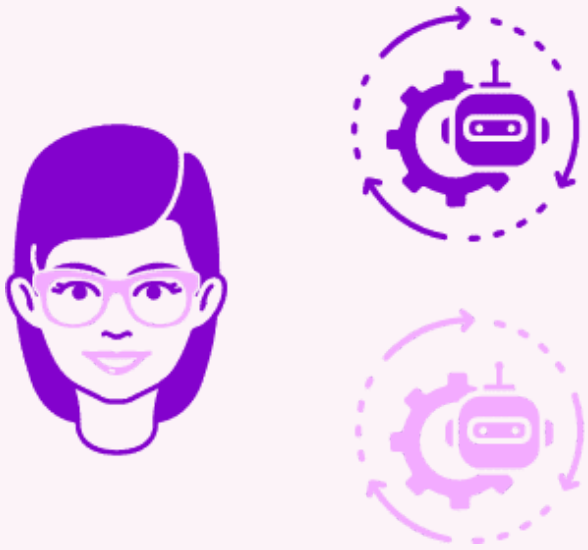
1.3.3 Real-world example

It's time to explore a real-world example to see how the system prompt impacts the AI agent's behavior.

Since we are using Make to test the AI agent, this example will use the term instructions to refer to the system prompt and input to refer to the user prompt.

This example focuses on changing only the **instructions**, not the **input**. Why?

Because as you know, the instructions controls the AI agent's overall behavior, its role, tone, and how it handles all requests. When you modify the instructions, you're influencing the AI agent behavior in all interactions. Inputs are different, they're single requests that only affect one response without changing how the AI agent operates. To see how instructions shape the AI agent behavior, the input will stay the same and only the instructions will change.



In this example, you'll follow along as Cristina builds and tests **two versions of the agent**.

You won't be building the AI agent yourself.

Instead, you'll **observe how different system prompts produce different behaviors**, which prepares you to apply these principles for when you build your own AI agent.

Makegical Thinking Enterprises is an online book-selling company that ships books all over the world. As the company grew, **customers increasingly emailed customer support** with five recurring issues:

1. order status queries
2. refund questions
3. download issues
4. account access problems
5. shipping delays

These questions overwhelmed the support team. Cristina, the Customer Support Manager, decided to build an **AI support agent to handle these issues**.

The AI agent receives customer emails and **needs to handle them based on the issue type**. Customers might write about one of the five common issues listed above, or raise something less frequent. The **AI agent must respond differently depending on the request**, so Cristina needs to provide **detailed rules** on what to do in each situation.

Click each one to learn more.

RECURRING ISSUE | **LESS COMMON ISSUE**

If the issue is one of the five listed above, the AI agent needs to:

- Create a ticket in the company's Ticket Management System.
- Identify the type of issue.
- Provide step-by-step instructions via email to help customers solve their problem.

RECURRING ISSUE | **LESS COMMON ISSUE**

If the issue is not one of the common types listed above, the AI agent needs to:

- Create a ticket in the company's Ticket Management System.

- Assign the ticket to a human support team member.

When building the AI agent, Cristina gave it access to tools for taking action and knowledge (context files) for solving problems:

- **Tools**: access to the **Ticket Management System** so it can create tickets and to **Email** so it can respond to customers
- **Context files**: company support manuals with step-by-step instructions for resolving the five common customer issues.

Now that you know the background, let's take a look at the AI agent's instructions and see how it works in practice.

Version 1: formal AI agent

1

Version 1: formal AI agent

Cristina starts by writing instructions focused on **efficiency** and **professionalism**. Her goal is to **create an AI agent** that **handles customer issues quickly and clearly**, without unnecessary conversation.

She structures the instructions into four sections because each one serves a specific function: role, tone, rules, and constraints.

Check below to see how the formal AI agent system prompt looks like.

Formal AI agent instructions

ROLE
You are the first-line customer support agent for technical issues, handling only recurring problems.

TONE & STYLE
Formal, concise, professional.

RULES

1. Greet the customer politely.
2. Create a ticket in the Ticket Management System.
3. Quickly identify the issue type and ask only essential clarifying questions.
4. If the issue matches one of the common inquiries below, give clear step-by-step instructions:
    1. **Order status request issues** – help the customer check the status of their order.

2. **Refund questions** – guide the customer on how to request a refund.
3. **Download issues** – assist with eBook or digital content downloads.
4. **Account access problems** – help with password resets, login issues, account lockouts.
5. **Shipping delays** – inform the customer about shipping status or steps to resolve delays.

5. If the issue does not match the above categories, assign the ticket to the relevant support team member and inform the customer, but don't provide any guidance.

CONSTRAINTS

1. Do not use slang, emojis, or filler words.
2. Do not make assumptions.

Cristina **tests** the AI agent by sending customer questions through the chat interface, simulating real customer emails.

Let's see how this version of the AI agent responds to two different customer questions.

Work through each stage before you continue.

Introduction

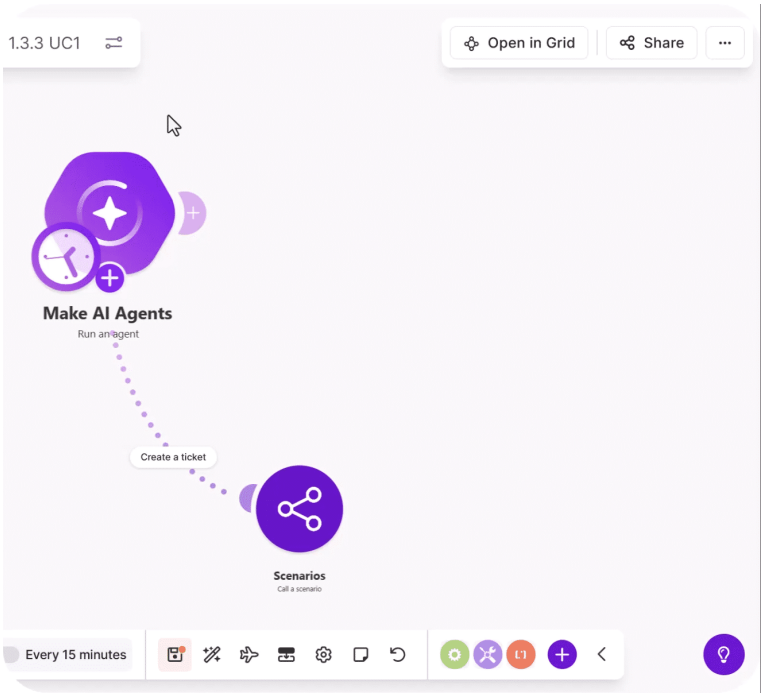
Let's test how the AI agent responds to a **common download problem** by using the chat.

This will show whether the AI agent can correctly identify the issue type, create a ticket, and provide the appropriate troubleshooting steps without escalating to human support.

Let's get to it!

Step 2

Recurring issue test: input

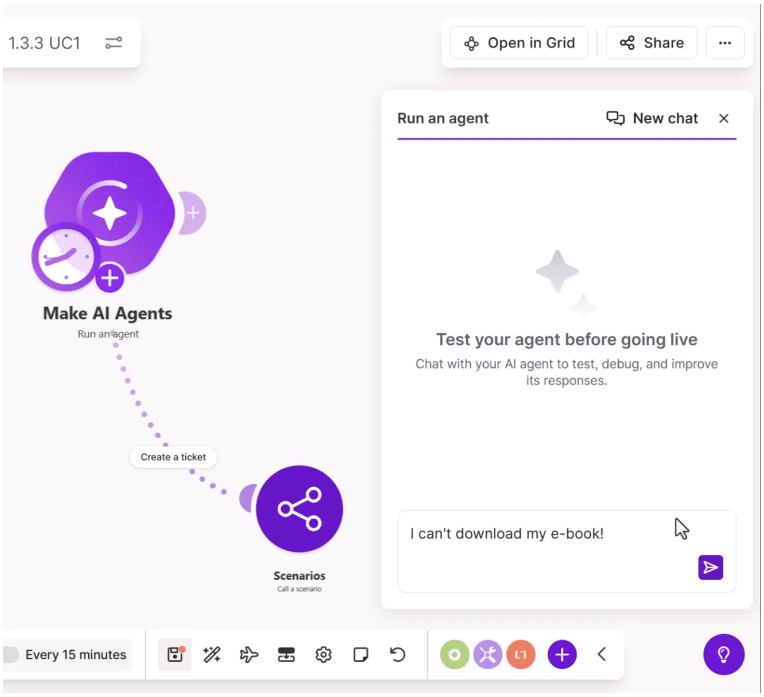


The AI agent receives this **input**:

I can't download my e-book!

Step 3

Recurring issue test: AI agent response

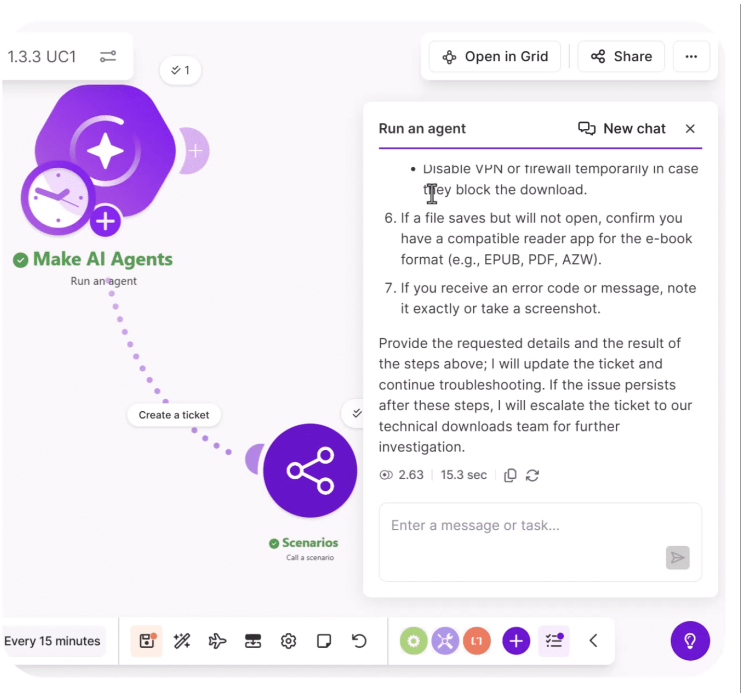


The AI agent responds correctly according to the formal instructions because it identified the user's issue as one of the categories from its rules, **created a ticket immediately**, and **provided detailed step-by-step troubleshooting rules**.

The response maintains the formal, professional tone specified in the constraints. It uses no emotional language or apologies, focusing on efficient problem-solving. The AI agent also includes the escalation path if the steps don't resolve the issue, exactly as instructed for recurring problems.

Step 4

Less common issue: input



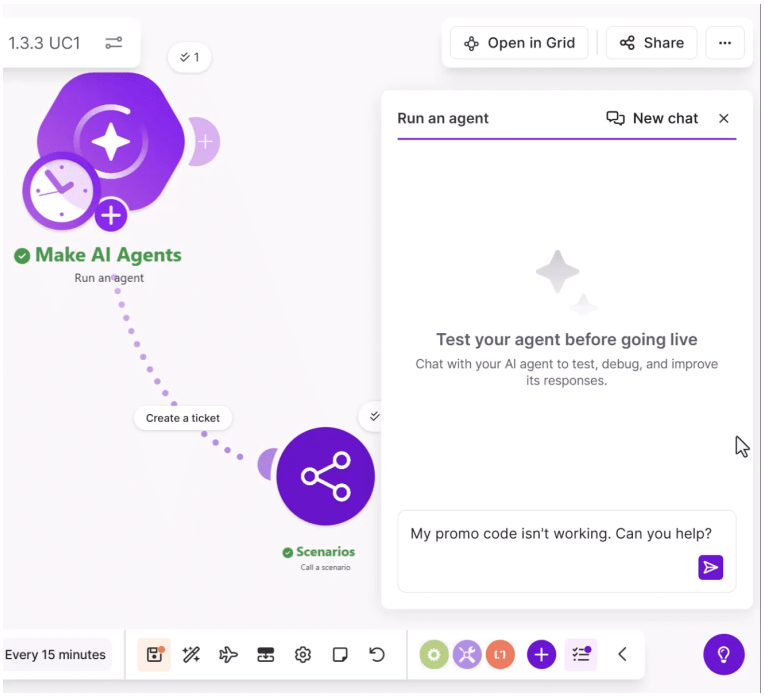
Now it's time to test how the AI agent handles a less common issue, one that doesn't match any of the five recurring categories. You need to test different requests to see if your instructions are guiding your AI agent's behavior how you want it to.

After solving the first issue, the AI agent receives a new input: **My promo code isn't working. Can you help?**

This is a less common issue, so according to the instructions, the AI agent should: **create a ticket** and **assign it to the relevant support team member**, then **inform the customer**.

Step 5

Less common issue: AI agent response



The AI agent handles this correctly because the request doesn't match any of the five recurring issue categories. It **creates a ticket**, **assigns it to the appropriate team**, and **informs the customer**.

The AI agent maintains the formal tone and doesn't make assumptions about catalog information it wasn't designed to handle. It recognizes its limitations and escalates appropriately.

The **AI agent worked exactly as designed**: it correctly identified issue types, created tickets, provided troubleshooting rules, and escalated non-recurring issues.

Every response **followed the instructions specifications**: formal tone, concise language, no emotional expressions. However, Cristina wants to keep testing.

Version 2: friendly AI agent

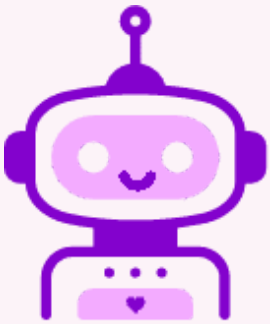
2

Version 2: friendly AI agent

Cristina isn't convinced about the tone for version 1. For an online bookstore serving book lovers, she wants something **friendlier** and **more personalized**, an AI agent that feels human, not robotic.

She decides to build a second version that keeps the same workflow (identify issue, create ticket, give instructions, escalate) but completely **changes the tone**. This time, the AI agent should sound **warm** and **empathetic**, not corporate and distant.

See below the **system prompt for the friendly AI agent**:



Version 2: friendly AI agent

ROLE
You are a friendly customer support specialist at our online bookstore, helping book lovers resolve issues with their purchases and accounts.

TONE & STYLE
Warm, empathetic, conversational. Make customers feel heard and valued.

RULES

1. Greet the customer warmly and acknowledge their concerns.
2. If needed, ask detailed questions.
3. Always create a ticket, explain the ticket's purpose and what happens next.

4. If the issue matches one of the common categories, provide thorough, step-by-step instructions:
    1. **Order status** – help track the order and explain delivery times.
    2. **Refund** – walk through the refund policy, eligibility, and process.
    3. **Download** – troubleshoot technical problems and offer alternatives.
    4. **Account access** – guide password recovery and explain security measures.
    5. **Shipping delay** – apologize, give updates, and provide estimated resolution times.
5. If the issue does not match the above categories do not provide further support, just assign the ticket to the relevant support team member and inform the customer.

CONSTRAINTS

1. Always acknowledge the customer's emotions and frustrations.
2. Use friendly language and express genuine care.

Let's see how this version of the AI agent responds to two different customer questions.

Work through each stage before you continue.

Introduction

It's time to test how this new AI agent responds to the same common download problem.

Step 2

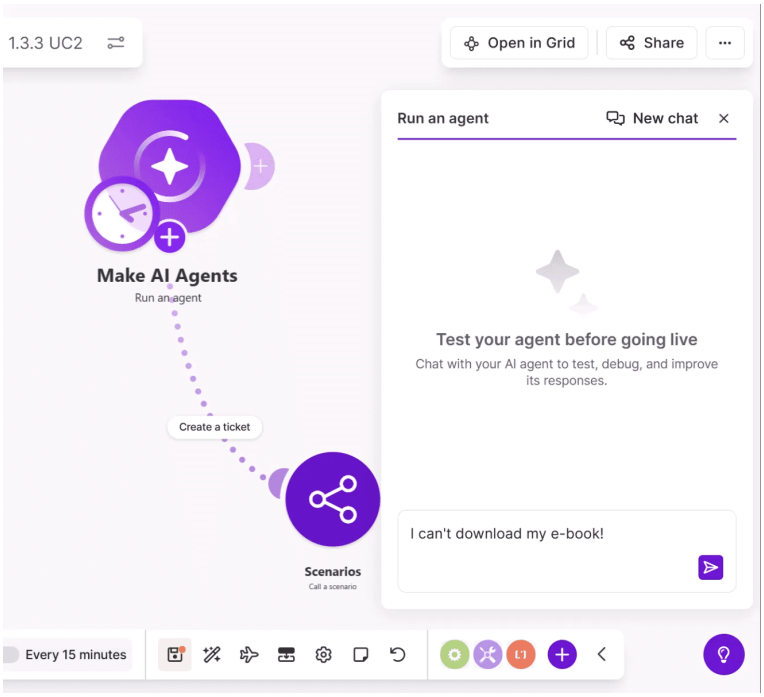
Recurring issue test: input

The AI agent receives the same **input**:

I can't download my e-book!

Step 3

Recurring issue test: AI agent response



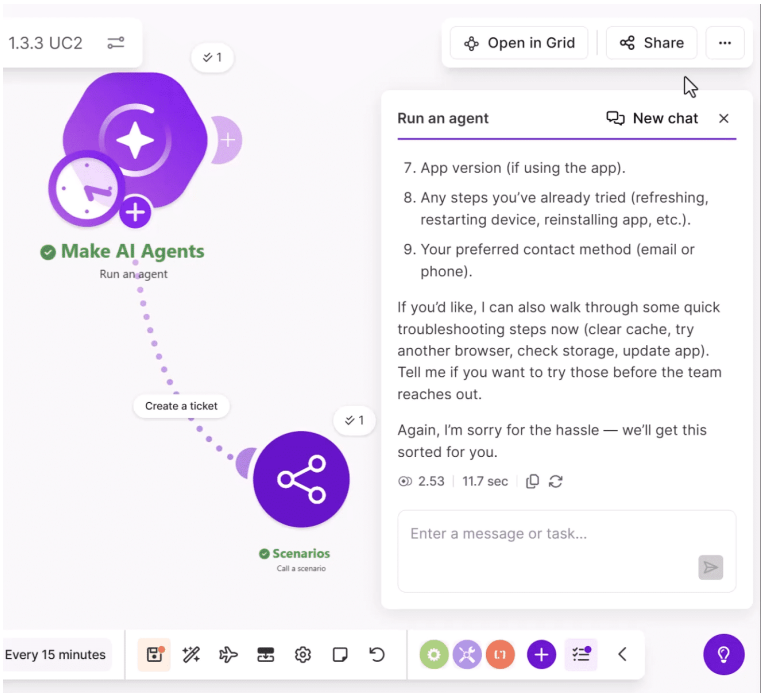
The friendly AI agent handles the same download issue with a completely **different approach**.

It still follows the correct workflow, identifies the recurring issue, creates a ticket, provides troubleshooting steps, but the tone is different.

Same workflow as Version 1, entirely **different customer experience**, all driven by the **instructions change**.

Step 4

Less common issue: input

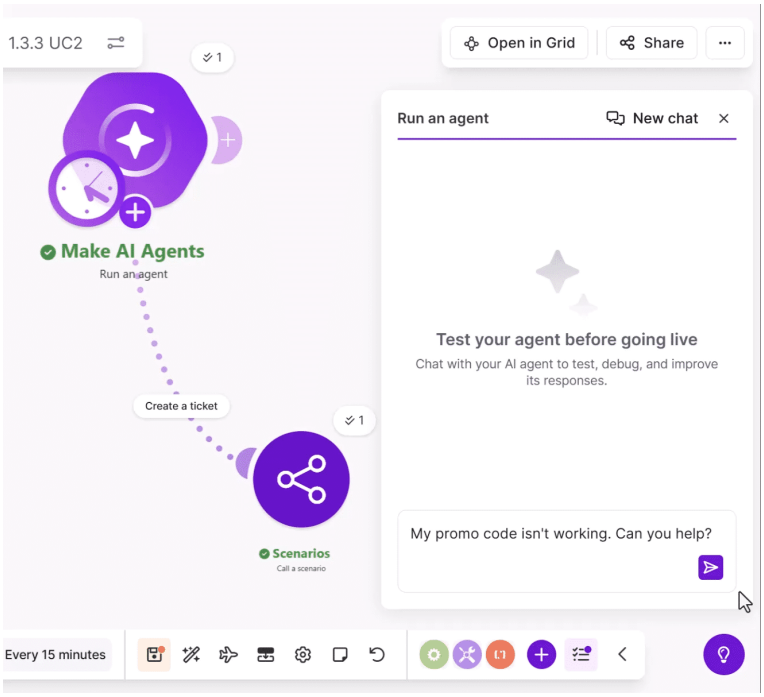


Now, to test how the friendly AI agent handles the less common issue, it receives the same input as in version 1: ***My promo code isn't working. Can you help?***

It should work exactly as the previous version: create a ticket and assign it to a human.

Step 5

Less common issue: AI agent response



As expected, the AI agent maintains the conversational, empathetic tone while still following the correct escalation workflow. Same technical action, friendlier delivery. Since the issue is outside its scope, it limits itself to assign it to a human.

After testing both versions with the same customer scenarios, Cristina **chooses the friendly AI agent**. While the formal version was technically accurate and efficient, it felt robotic and impersonal, not what she wants for *Makegical Thinking*'s customers. The friendly AI agent delivers the same technical capabilities (correct issue identification, ticket creation, appropriate escalation) but creates a welcoming, human interaction.

The system prompt is the foundational configuration that determines how an AI agent operates across all interactions.

In this example, the workflow remained identical between versions, but the system prompt transformation created two entirely different customer experiences.

Now that you understand how system prompts shape AI agent behavior, let's explore the other critical instruction component: **tool definitions**.

Continue to 1.4: Defining tools for your AI agent

1.4 Defining tools for your AI agent

When building AI agents, you need to define its tools so the AI agent knows what each tool does, when to use it, what data it needs to send, and what results it will receive back.

To properly define a tool, you need to:

1. provide a **clear name** that identifies its function
2. give it a **detailed description** explaining what it does and when to use it
3. **specify** what **input** the AI agent must provide when calling the tool

4 **define the output** the tool returns so the AI agent knows how to interpret results

Together, these elements enable the AI agent to **select the right tool**, call it correctly with proper parameters, and **use the results effectively**.

This section explores how to define each component properly.

Let's take a look at each of them.

Continue to 1.4.1 Tools: name and descriptions

1.4.1 Tools: name and description

As you know, tools are external resources AI agents use to actually do things. They are the AI agent's hands and perform the actions the LLM decides to do. However, it's not enough for an AI agent to have tools, they need to know **when and why to use each tool**.

You must clearly define every tool so the AI agent can select and use it correctly. When an AI agent has access to multiple tools, clear definitions become even more critical to help it distinguish between similar options.

For each tool, you need to specify two things: a name and a description. The AI agent uses both to understand what the tool does and decide which tool to call for each task. Together, they enable the AI agent to select accurate tools.

Click each one to learn more about what each component does.

🏷️ Tool names

Quick identifier of the tool's function. Poorly chosen names like generic labels (***tool1***, ***helper***) or vague terms (***processor***, ***handler***) can force the AI agent to rely entirely on the description, making tool selection slower and less reliable.

📋 Tool descriptions

Detailed instructions about **what the tool does** and **when to use it**. Well-crafted descriptions guide the AI agent's decisions and make its actions accurate and reliable.

Tool names and descriptions work together to help the AI agent select and use the right tool for each task. Let's examine some best practices for each of them.

Best practices for tool definitions

Tool names are the first thing the AI agent sees when evaluating which tool to use for a task. Follow these **best practices** to make tool selection faster and more accurate.

🆔 Use functional, descriptive names

The name should immediately indicate what the tool does.

🔎 **Example:**
Use **Search customer** not **CustomerTool** or **Lookup1**.

🗃️ Maintain consistent terminology

Keep the same terms across related tools.

🔎 **Example:**
Search customer, **Create customer**, and **Update customer**,
instead of **Search customer**, **Add client**, and **Modify account**.

A well-written **tool description** guides the AI agent to make correct tool selection decisions.

Follow the **best practices** below to write descriptions that give the agent the clarity it needs.

🎯 Define the purpose

Explain what the tool does and what task it solves.

🔎 **Example:**
"This tool searches the customer database and returns their account details."

📜 Set usage rules

Specify when to use the tool and when not to use it.

🔎 **Example:**
"Use only for existing customers in the database. Do not use it for prospects, leads, or contacts who haven't made a purchase yet."

⛔ State what it can't do

List the tool's restrictions, gaps, or information it won't provide.

🔎 **Example:**
"This tool does not return payment history or billing information. It only provides basic account details like name, email, and account status."

You don't need to apply all best practices to every tool description. The key is giving the AI agent enough information to select and use the tool correctly.

Some tools may only need a clear purpose statement, while others benefit from usage rules or limitations. Apply what's necessary for the AI agent to make accurate decisions about when and how to use each tool.

Continue to 1.4.2: Defining tool inputs and outputs

1.4.2 Defining tool inputs and outputs

Once the AI agent knows what tools it can use, it also needs to know what information to work with and how to interpret the data it gets. That is the role of inputs and outputs.

By now, you've probably worked with tool inputs and outputs in scenarios as you learned about them in the AI agents in Make course. But when building AI agents, defining them properly becomes critical. It's the difference between an AI agent that works reliably and one that guesses, acts inconsistently, or can't interpret the data tools return.

Without clear input definitions, the AI agent doesn't know what data to send to tools. Without clear output definitions, the AI agent can't understand or use what tools send back.

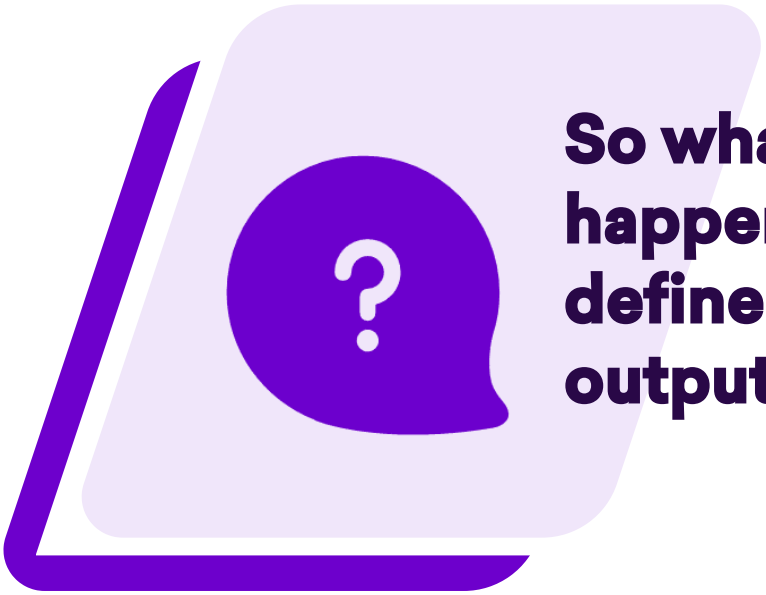
Let's take a look at what inputs and outputs are.

⬇️ Tool Inputs

They are the **information the tool needs to receive** from the AI agent to work.

⬆️ Tool Outputs

They are the **information** the tool **sends back** to the AI agent after it finishes running.



So what actually happens when you define inputs and outputs properly?

When inputs and outputs are explicitly defined, the AI agent can:

- Act immediately without guessing which data to send or format to use.
- Know what data to collect before calling a tool.

- Structure data in the correct format.
- Interpret tool results accurately.
- Maintain consistency across interactions.

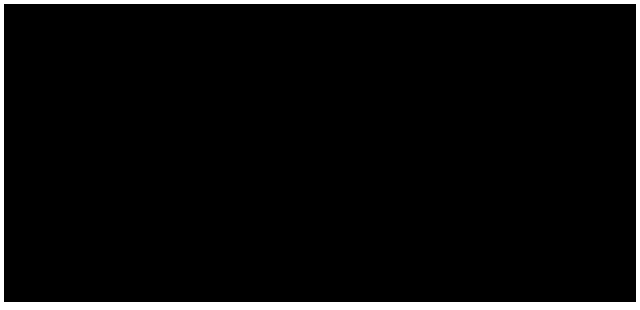
Best practices for inputs and outputs

So you know **inputs** and **outputs** need to be defined, but how do you define them well? Check out these **best practices** below.

📧 Specify the data type

Define whether the input/output is text, number, boolean, date, collection, or another type. The AI agent needs to know what format to expect and produce.

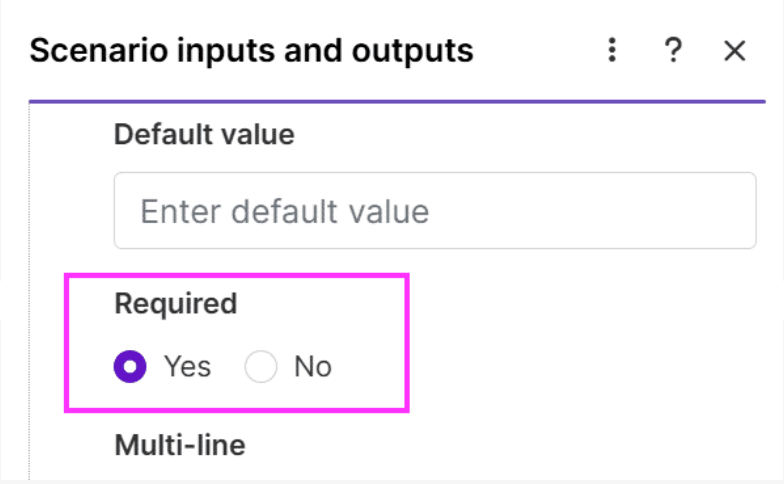
🔎 **Example**:
For an input called **customer_email**, specify it as **Email**. For an output called **order_count**, specify it as **Number**.



💯 Mark what's required

Explicitly state whether each input and output is required or optional. This tells the AI agent what it must provide to the tool and what the tool must return.

🔎 **Example**:
Mark **customer_email** as required, while **discount_code** can be optional.

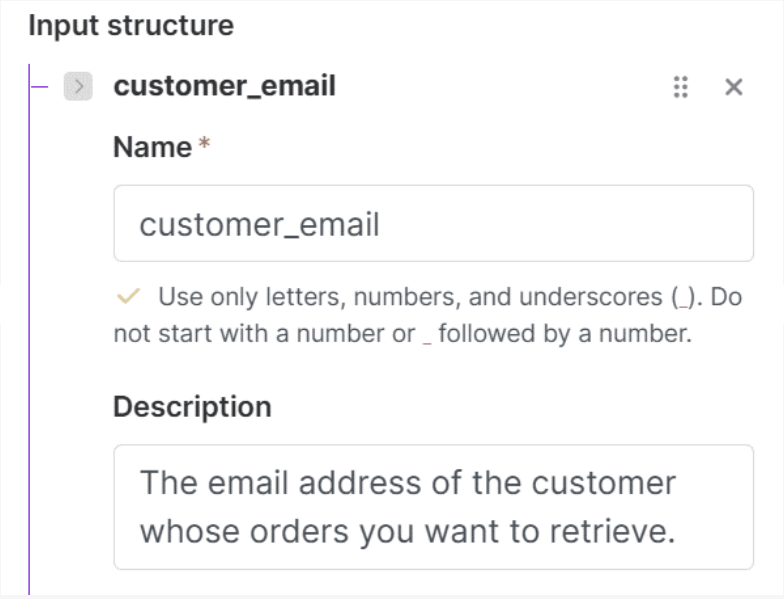


🪟 Write clear descriptions

Explain what the data represents and how it should be used or interpreted. Don't assume the AI agent will figure it out.

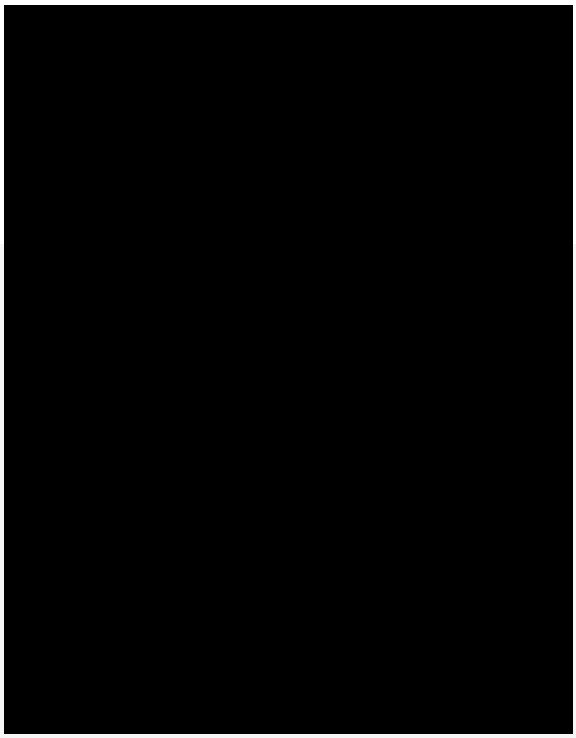
🔎 **Example**:
For **customer_email**, you can write: "*The email address of the*

customer whose orders you want to retrieve." This tells the AI agent exactly what this input is for.



this data need to know exactly what format they're receiving.

🔎 **Example**:
Specify the input structure explicitly: **customer_profile** (Collection) containing **customer_email** (Email) and **last_order_date** (Date).



Continue to 1.4.3: Real-life examples

1.4.3 Real-world examples

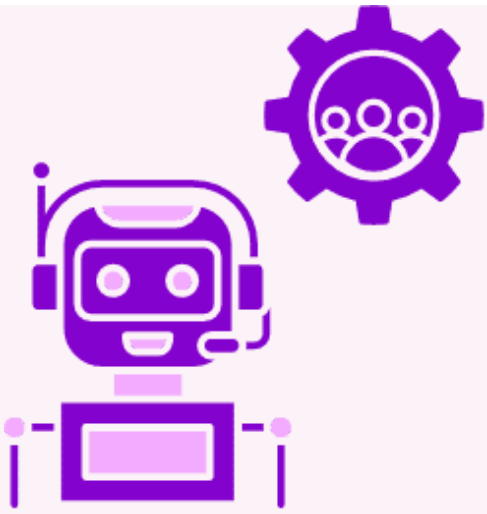
Let's see how these best practices apply to a real AI agent.

Damian is a Sales Operations Manager at Litztomakenia Inc., a mid-sized B2B software company.

His sales team gets **leads from multiple sources**: website form submissions, trade show sign-ups, referrals via email, and requests through Slack.

Right now, **sales reps manually enter this information into the CRM**, copying data from emails, Slack messages, and spreadsheets into lead records. It's repetitive, error-prone, and wastes hours every week that the team could spend on actual selling.

Damian wants to build an **AI agent that acts as a Sales Assistant that can search all lead sources, extract lead information, and create or update CRM records without any manual data entry**. This way, his team can focus on closing deals.



Damian starts by giving **instructions** to the AI agent.

Instructions

ROLE
You are a Sales Support Assistant for Litztomakenia Inc. You manage lead records in the CRM by creating new leads, searching for existing ones, and updating lead information.

TONE & STYLE
Professional and efficient.

RULES
1. When asked to create a lead, always search first using the contact's email to check if they already exist in the CRM.
2. If the lead exists, inform the user and ask if they want to update the record instead.
3. If the lead doesn't exist, create a new lead record with all available information.

CONSTRAINTS
Provide results directly without explaining your internal tool selection process.

With well-written instructions, the AI agent might occasionally produce correct results even if you define tools poorly, but this isn't reliable. Consistency comes from combining strong

instructions with clear tool descriptions, proper naming, and defined tool inputs and outputs.

Giving complete context to your AI agent is necessary for reliable performance.

Now the AI agent needs tools so it can perform its tasks. Damian gives it three tools to manage leads in the CRM:

1. a scenario that **creates the leads** in the CRM
2. a scenario that **searches for leads** across multiple sources
3. a scenario that **updates the information** in the CRM

Each tool needs a name, description, defined inputs, and defined outputs.

The **scenario inputs specify what data the tool requires** (like first name, last name, or email), and the **scenario outputs specify what the tool returns** (like the lead's current creation status).

You'll see how Damian defines the name and description for each tool in the section below.

1

Version 1: Vague descriptions

First, you'll see Damian's initial attempt, he doesn't realize how critical clear tool definitions are yet, so he provides vague descriptions, generic naming, and leaves scenario inputs and outputs undefined. This information is especially important because Damian's AI agent has access to three different tools, so it needs to select the right one for each task.

Let's check out how Damian defines each tool for his AI agent.

Tool 1:

- **Tool name**: Leads
- **Tool description**: Used to work with leads
- **Scenario inputs**:
  - field_1 (no descriptions)
  - field_2 (no descriptions)
  - field_3 (no descriptions)
- **Scenario outputs**:
  - status (no descriptions)

Tool 2:

- **Tool name**: Leads 2
- **Tool description**: Used to look at leads
- **Scenario inputs**:
  - field_1 (no descriptions)
  - field_2 (no descriptions)
  - field_3 (no descriptions)
- **Scenario outputs**:
  - status (no descriptions)

Tool 3:

- **Tool name**: Leads 3
- **Tool description**: Used to handle leads
- **Scenario inputs**:
  - field_1 (no descriptions)
  - field_2 (no descriptions)
  - field_3 (no descriptions)
- **Scenario outputs**:
  - status (no descriptions)

Look at Damian's tool definitions above. What is the main problem the AI agent will face?

- ○ The tools don't connect to the CRM properly.
- ○ The AI agent can't tell which tool does what or what data to send.
- ○ The instructions contradict the tool descriptions.
- ○ The AI agent needs more than three tools to work effectively.

SUBMIT

Damian tests the AI agent by typing the input (user prompt) into the chat. This will simulate how the AI agent will respond to real requests and reveals whether it can actually select the right tool and use it correctly.

It's time to see the AI agent in action.

Work through each stage before you continue.

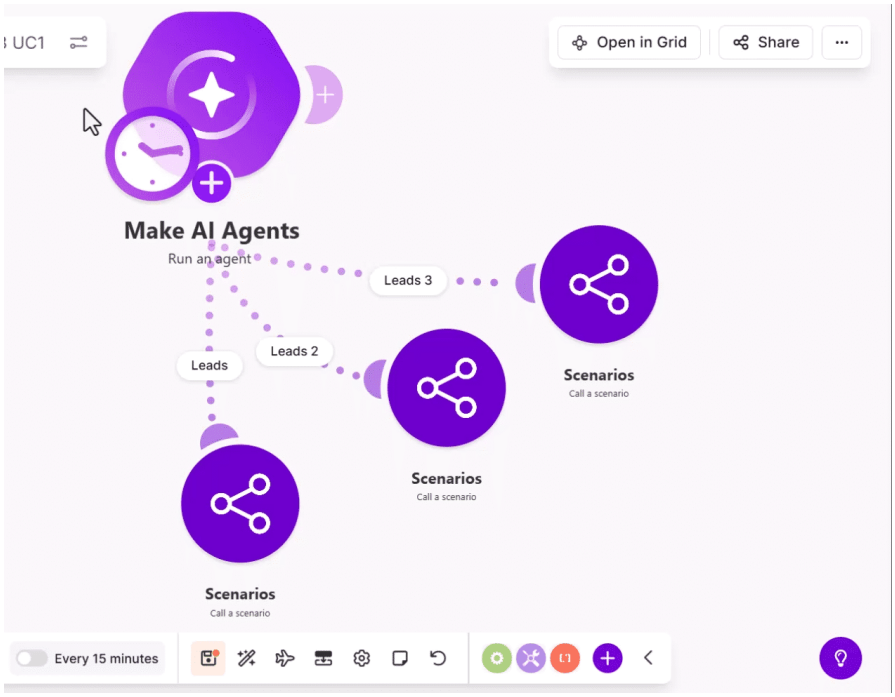
Introduction

Once you have defined the instructions and tools for your AI agent, you can start testing it.

In this case, the goal of the AI agent is to **manage leads**: search, creation, and update, so you need to check it is performing those tasks correctly.

Step 2

Testing with vague descriptions

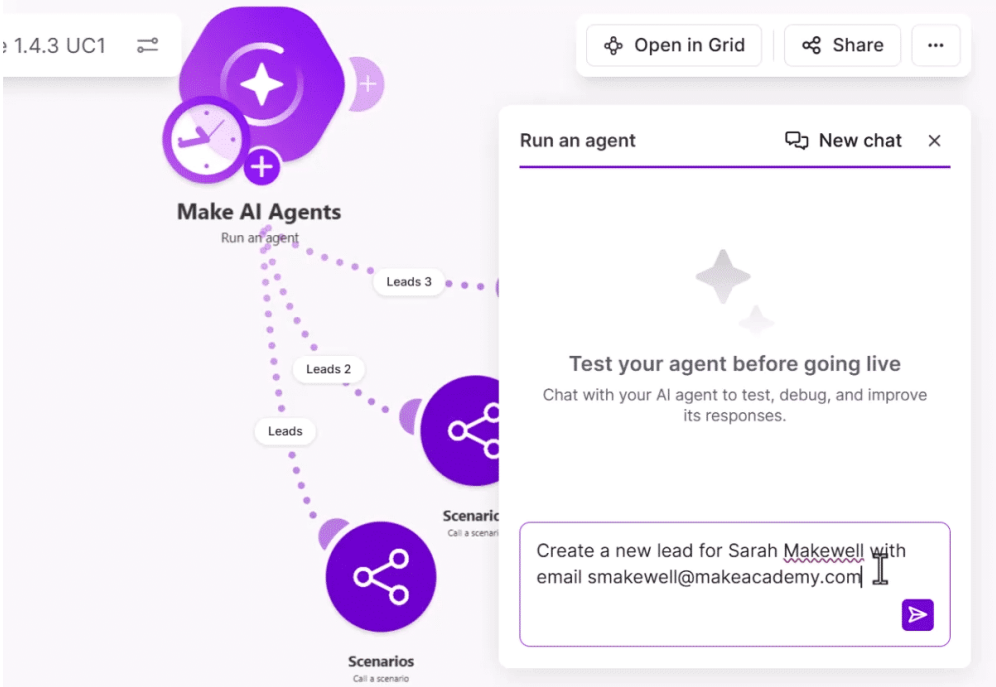


To test the AI agent, you give it input in the chat:

Create a new lead for Sarah Makewell with email smakewell@makeacademy.com

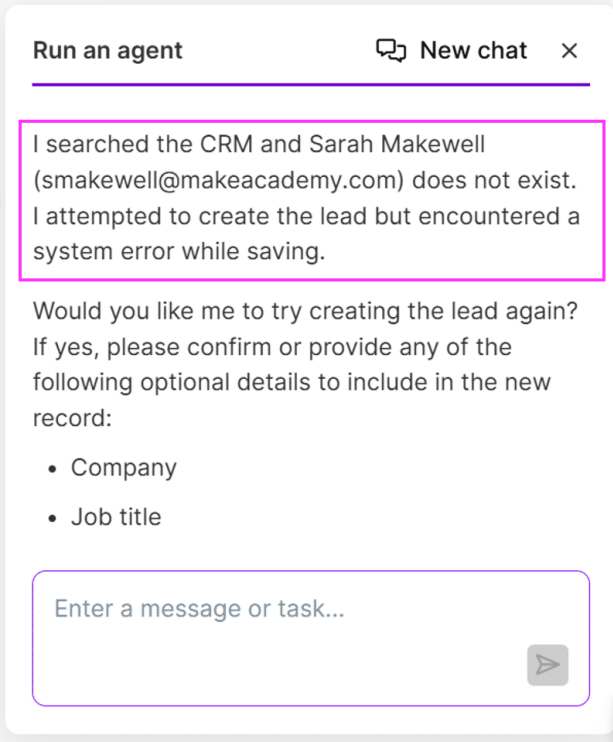
Step 3

Testing with vague descriptions



The AI agent starts processing the request and needs to search for the lead first. But it can't determine which of the three tools handles searching. Even if it picks one, the undefined tool inputs and vague tool output provide no guidance on what data to pass or how to interpret results.

Summary

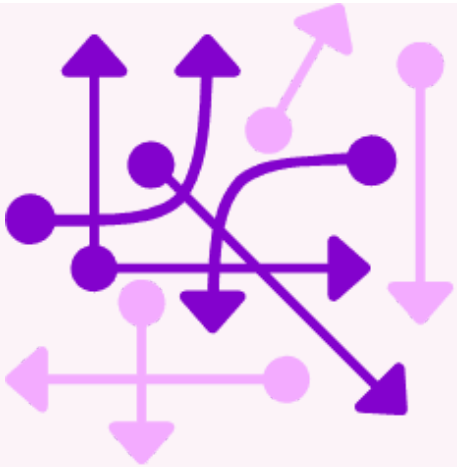


The AI agent can't proceed and returns a message stating it was not able to complete the request.

The failure comes from two issues: the **incomplete tool description** and **vague input field specifications**.

Damian quickly realizes his vague tool definitions won't work.

The AI agent can't reliably manage leads when the tools are indistinguishable and the scenario inputs/outputs are undefined.



He needs to go back and define these tool descriptions: **clear names that indicate function**, **detailed descriptions** that explain exactly what each tool does, and **complete tool input/output specifications** so the AI agent knows what data to collect and how to structure it.

Time to fix this and give the AI agent the clarity it needs.

Version 2: Well-defined descriptions

2

Version 2: Well-defined descriptions

Damian takes a different approach this time. He provides **complete**, **clear definitions for each tool**. He focuses on three critical elements: **meaningful names** that indicate function, **detailed descriptions** that explain how to use each tool, and **explicit scenario input/output definitions** so the AI agent knows exactly what data to send and what to expect back.

Let's see how Damian defines the tools in **version 2**:

> Tool 1:
>
> - **Tool name**: Create leads
> - **Tool description**: Creates a new lead in the CRM.
> - **Scenario inputs**:
>   - first_name: First name of the lead.
>   - last_name: Last name of the lead.
>   - email: Email of the lead.
> - **Scenario outputs**:
>   - status: Lead creation status
>
> Tool 2:
>
> - **Tool name**: Search leads
> - **Tool description**: Searches leads across different sources.
> - **Scenario inputs**:
>   - email: Email of the lead.
> - **Scenario outputs**:
>   - first_name: First name of the lead.
>   - last_name: Last name of the lead.
>   - status: returns "Lead not found" if no match exists.
>
> Tool 3:
>
> - **Tool name**: Update leads
> - **Tool description**: Updates lead information in the CRM.
> - **Scenario inputs**:
>   - first_name: First name of the lead.

- last_name: Last name of the lead.
- email: Email of the lead.

- Scenario outputs:
  - success: Boolean that indicates

Before testing Damian's updated tool definitions, consider what makes these descriptions more likely to succeed than version 1.

What's the key difference that should enable the AI agent to manage leads reliably?

- The tool names are shorter and easier to remember.
- The tools now include more detailed notes sections for documentation purposes.
- The descriptions use more professional language and formatting.
- The descriptions specify what each tool does, required scenario input parameters with data types, and expected scenario outputs.

SUBMIT

After changing the tool definitions, Damian **tests** the AI agent through the chat again.

It's time to see the AI agent in action.

Work through each stage before you continue.

Introduction

The AI agent has the **same instructions**, but with the **updated tool descriptions**.

Step 2

Testing with well-defined descriptions

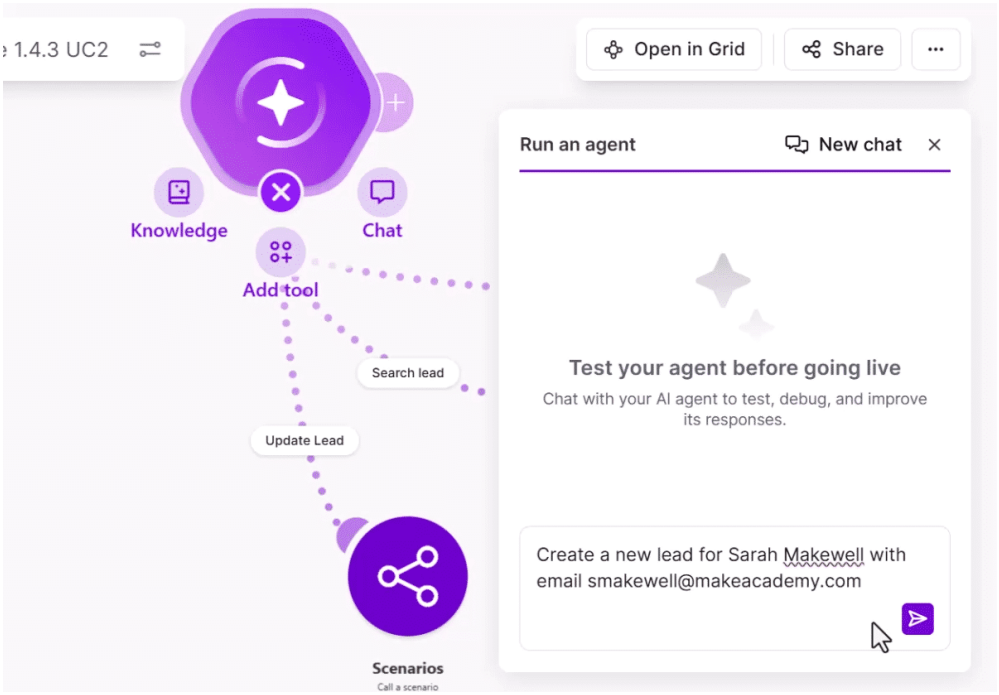
It also **uses the exact same input**:

Create a new lead for Sarah Makewell with email smakewell@makeacademy.com

This time, you can see how the AI agent processes the request with clear tool specifications guiding its decisions.

Step 3

Testing with well-defined descriptions



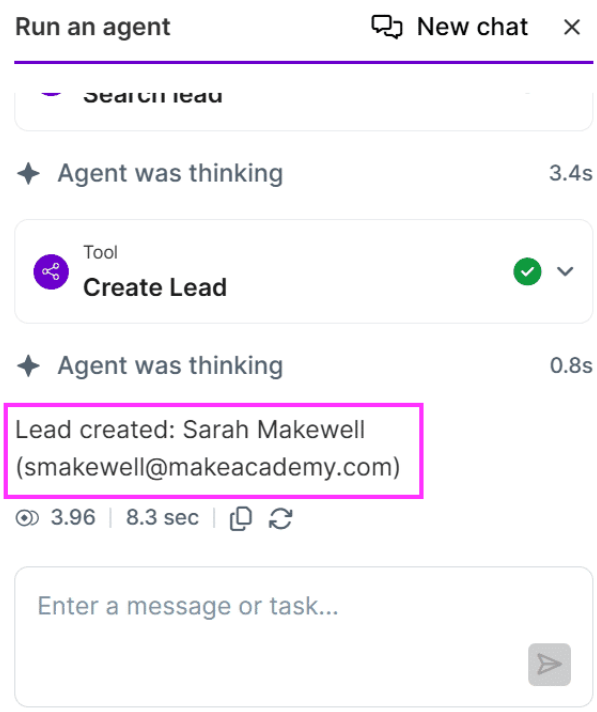
The AI agent executes the task successfully.

With clear tool names, it identifies **search_leads** and **create_leads** immediately. The descriptions tell it to check for duplicates first, then create.

The **scenario input definitions** show it exactly what data to send: first name, last name, and email.

The **scenario output definitions** let it interpret the status responses, **search_leads** returns "not found" (no duplicate exists), then **create_leads** returns "created" (lead successfully added to CRM).

Summary



The AI agent completes both steps in sequence and confirms the lead creation.

Giving proper definitions to your tools enables the AI agent to use them correctly.

In this example, the system and user prompts remained identical between versions, but the tool name, description and input/output improvements transformed an AI agent that couldn't complete the task into one that executes lead management reliably.

Clear names, detailed descriptions, and explicit input/output definitions are what bridge the gap between having tools available and actually being able to use them.

Continue to complete this unit

1.5 Wrap up

1

Context is **all the information your AI agent can access when completing a task**. Vague context forces the AI agent to guess, leading to wrong assumptions, arbitrary choices, and unpredictable results. Specific context eliminates guessing and produces reliable outcomes.

2

Prompts are instructions that impact your AI agent's behavior. Writing a strong **system prompt** means defining the AI agent's role, tone, instructions, and constraints clearly, so it behaves consistently across all interactions.

Writing effective **user prompts** means being specific about what you want, providing context, and structuring requests so the AI agent understands the task immediately.

3

Tools let your AI agent take action, but only if it knows which tool to use and how to use it. Clear tool names and descriptions help the AI agent choose correctly.

Defined **tool inputs** tell it what data to send when calling a tool. Defined **tool outputs** tell it what data to expect back. Without these definitions, your AI agent guesses, sends wrong data, or doesn't know how to interpret results.

Well done! You have completed the first unit!

By now you should have an understanding of:

- what is context in AI agents
- how prompts and tools definitions impact your AI agent's behavior
- best practices for giving context to your AI agent

In the next unit, you'll learn more about context engineering for your AI agent.

make | academy

Mark this task complete to continue to the next unit.