

Exercise 1: Follow-up questions and slots

What we just did, beside learning about `<? input.text ?>` is handle a follow-up question in a child node. This is a common pattern in which a parent node asks for information or clarification from the user and then one of its child nodes handles the response to the user.

If multiple follow up questions that are dependent on each other have to be asked by the chatbot, you'll end up with a cascade of child nodes, each asking the next question in the chain and having their child process it. This works but it's not ideal in terms of reasoning about or structuring your chatbot dialog flow.

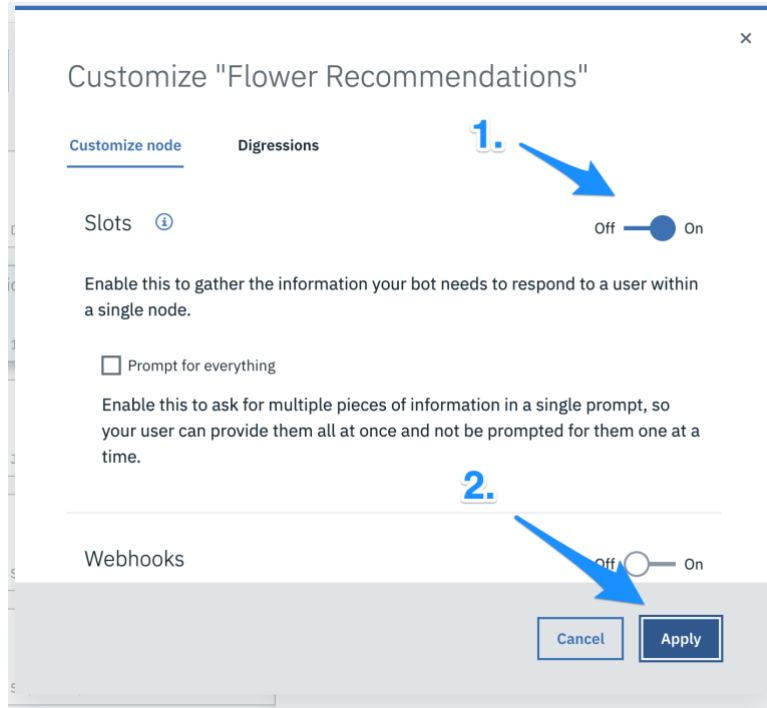
Another shortcoming of this approach is that if the user asks a side question or just says, *wait a second*, instead of replying to what we asked, we'll end up losing our "position" in the dialog cascade and therefore end up treating the delayed answer as a brand new input, failing (most likely) to provide an appropriate response or collect the information we wanted.

There is a much better tool to help us collect information from the user and store it in context variables. Namely, I'm talking about *Slots*.

Let's see a practical example of how they work.

1. **Define an intent called `#flower_recommendations`** with at least 5 examples of ways people might ask for flower suggestions (e.g., Flower recommendations, flowers suggestions for my girlfriend, Which flowers for Valentine's Day?, etc.). Watson will train on it as usual.
2. **Create a node called Flower Recommendations** below the *Welcome* node (as a peer, not a child node). **Set the condition to `#flower_recommendations`**. This is the node that will handle our flower recommendations.
3. Click on the *Customize link* in the node and **turn on the *Slots* feature**. Leave *Prompt for everything* unchecked, as this option is only useful if you have multiple slots/questions for the user and you want to ask them all at once, rather than one at the time. Not a common scenario. Finally, click on the *Apply* button.

Lab 11: Master Slots



4. This will automatically add one empty slot to our node. We use slots to collect information from the user and store it in a context variable.

Flower Recommendations

Customize

If assistant recognizes:

#flower_recommendations



Then check for: 0 Manage handlers

	CHECK FOR	SAVE IT AS	IF NOT PRESENT, ASK	TYPE
1	<div>Enter condition</div>	<div>Enter variable</div>	<div>Enter prompt</div>	Optional

Add slot

Lab 11: Master Slots

The three key components of a slot are *CHECK FOR* (often an entity), *SAVE IT AS* (a context variable), and *IF NOT PRESENT, ASK* (an optional question to explicitly request the information if not provided). **Enter @occasion, \$occasion, and What occasion are the flowers for? respectively.** You'll notice that the slot type changes from *Optional* to *Required* the moment we add a question.

	CHECK FOR	SAVE IT AS	IF NOT PRESENT, ASK	TYPE		
1	@occasion	\$occasion	What occasion are the	Required		

This node will be executed when its condition #flower_recommendations is detected. In other words, when the user is asking for flower suggestions. However, we want to know for which occasion the flowers are meant, so as to have an appropriate response for different occasions.

The slot will automatically assign @occasion to the \$occasion context variable if the user provided an entity value in their original question (e.g., *flowers suggestions for Valentine's Day*) and not ask the question in that case.

If the @occasion entity is not detected, because the user simply asked, *I'd like some flower recommendations* without specifying a particular occasion, then the slot will ask *What occasion are the flowers for?* until the user replies with a relevant @occasion. The slot is like a dog with a bone and will keep asking the question until the user enters a valid occasion. So, if the user enters an irrelevant reply, the slot will ask the question again.

By the way, a node can have multiple slots (through that *Add slot* button), if multiple pieces of information need to be collected.

5. After the slot does its job of clarifying with the user which occasion we are talking about, it will store it in the \$occasion context variable. So, we can use it directly in the response section of the same node, without the need to create a child node. We want to provide a different answer for each occasion, so **enable *Multiple conditioned responses for the node*** from the *Customize* link as well.











6. Now you can add different answers leveraging the content of the context variable \$occasion, as shown in the image below. Go ahead and **replicate it in your *Flower Recommendations* node**, handling at least a few occasions from @occasion. If you don't implement them all, make sure you add a *true* fallback response for the occasions you don't handle otherwise the user will receive no response at all (a cardinal sin of chatbot design).

For the generic response, you might recommend a mixed bouquet that is versatile enough for different occasions. (Admittedly I know much more about chatbots than flowers.)

Lab 11: Master Slots

The slot sets the context variable `$occasion` for you. Make sure you use `$occasion` not `@occasion` in your multiple responses. This way, if the user specified the occasion earlier in the conversation, we still have memory of it and can reply appropriately.

Assistant responds

	IF ASSISTANT RECOGNIZES	RESPOND WITH		
1	<code>\$occasion:Christmas</code>	I'd go with all-time classic: a beautif		
2	<code>\$occasion:Birthday</code>	Opt for a fun bouquet of flowers, chc		
3	<code>\$occasion == "Valentine's Day"</code>	You can never go wrong with a dozer		
4	<code>\$occasion == "Mother's Day"</code>	Moms are awesome and worth celebt		
5	<code>true</code>	I'd recommend a beautiful mixed bo		

To save you some time, here are the responses you could use:

- For Christmas: I'd go with all-time classic: a beautiful Red Poinsettia.
- For Birthdays: Opt for a fun bouquet of flowers, choosing a colorful one from [our catalog](https://example.org/catalog).
- For Valentine's Day: You can never go wrong with a dozen red roses.
- For Mother's Day: Moms are awesome and worth celebrating every day. Consider our [Mother's Day special bouquet](https://example.org/mothers-day).
- For the fallback, `true` case: I'd recommend a beautiful mixed bouquet from [our catalog](https://example.org/catalog).

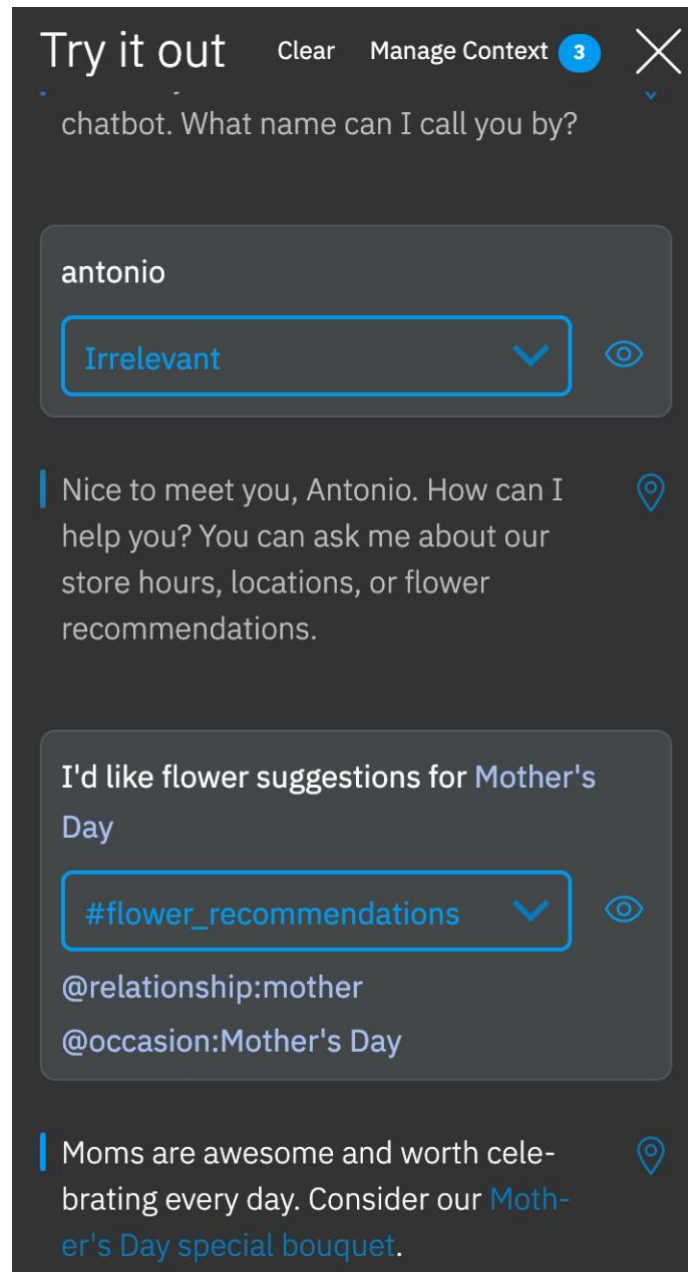
7. Once you've added a few, open the *Try it out* panel, press the *Clear* link if needed, and **test that this is actually working**. For example, for your turns, try entering:

(Your name)

Lab 11: Master Slots

I'd like flower suggestions for Mother's Day

You should get the response you specified (provided you added one for the condition `$occasion == "Mother's Day"`). Something similar to the conversation shown in the image below.



As a reminder, we can normally use the `:` shorthand when working with entity values that have no spaces. So `$occasion:Birthday` is equivalent to explicitly saying `$occasion == "Birthday"` which means *the value stored in \$occasion is Birthday*. However, if the entity value

Lab 11: Master Slots

contains a space, as it's the case for `@occasion:(Mother's Day)`, you'll want to use the explicit form with the “equal equal” symbols (e.g., `$occasion == "Mother's Day"`).

Using a slot saved us from having to implement collecting `$occasion` somewhere (e.g., in a passthrough node like we did for *Assign City*), handling everything neatly in one node. With a required slot (so one for which you defined a question to collect information from the user), we can count on `$occasion` existing as we formulate our responses.

Note that if you don't specify a question, the slot becomes optional, which means that the entity value will be stored in the context variable of your choice only if detected in the user input, but the user won't be asked explicitly for it (since you didn't provide a question).

If you add two required slots to a node, then the node will ask the first question, store the information in your first context variable, then proceed with asking the second question and storing that answer in the second context variable you specified. In our case, we could have used the second slot to ask for the `@relationship` entity. Knowing both occasion and relationship would then allow us to come up with really fine-tuned answers. In the responses, we would be able to combine the two through logical AND and OR logical operators (e.g., `$occasion:Birthday AND $relationship:wife`).

The classic example of multiple slots in a node is a chatbot that makes a restaurant reservation. Let's say that the information it needs to collect is the name, phone number, date and time, and party size. The node can define a slot for each of these values with their respective questions. This greatly simplifies the dialog flow, as it reduces what would require several nodes, to a single node that does all the work. It also ensures that the answers are collected before the conversation proceeds further which is crucial in a scenario where, say, you are making a reservation.

To handle complex logic, you can use a combination of slots and child nodes. Slots to collect the info, child node to do the processing of that information according to your logic/preferences.

And since slots collect the information in context variables, we can refer to their values throughout the conversation with the user. So, in the example of the reservation, we might be able to provide a confirmation of the reservation as we wave the user goodbye.



Exercise 2: Reimplement Hours of Operation

Now that we know how to work with slots, we can greatly simplify our *Hours of Operation* (and eventually the *Location Information*) node.









1. **Get rid of the *Assign City* node** by clicking on the more options menu in that node, and then selecting *Delete*.

Lab 11: Master Slots

2. **Define a slot with the condition @location inside of *Hours of Operation*.** Assign the value to \$city. Make the slot required, that is, explicitly ask the user For which city?, if they didn't specify it in their original question.

	CHECK FOR	SAVE IT AS	IF NOT PRESENT, ASK	TYPE		
1	@location	\$city	For which city?	Required		

3. **Enable *Multiple conditioned responses* for the node.** Then **move the response information from the *Our Locations* child node** into these responses within *Hours of Operation*, as shown in the image below.

	IF ASSISTANT RECOGNIZES	RESPOND WITH		
1	\$city:Toronto	Our Toronto store is open Monday to Satu		
2	\$city:Montreal	Our Montreal store is open Monday to Fri		
3	\$city:Calgary	Our Calgary store is open Monday to Satu		
4	\$city:Vancouver	Our Vancouver store is open Monday to F		

4. Since *Hours of Operation* now does issue a response, we **need to change the *And finally* action to *Wait for user input*** so that the conversation with the user can continue. In other words, we are no longer going to use the child nodes to handle the interaction.

5. Copy the responses for the system location and no location child nodes somewhere (e.g., in Notepad) and then **delete *Hours of Operation's* child nodes (all three of them)**.

At this point, you will have the basic scenario for our locations handled by the combination of the slot and the multiple conditioned responses. If you test it with what are your hours of operation the chatbot will ask you *For which city?* and if you reply with one of our cities such as Vancouver, you'll get the right response. Great.

Lab 11: Master Slots

Replacing all three scenarios in one node

Since we made the slot required with a question, we don't need to worry about the fallback case where the user doesn't specify a city (for now). The slot will ask for one and won't settle until a *@location* is provided. In other words, we can ignore the scenario that was handled by the *Location Not Provided* child node that we just deleted.

But, we should be able to handle the *@sys-location* case. Right now, the slot will demand *@location* and ignore *@sys-location*. **Try replying Seattle when asked for which city.**

The screenshot shows a chatbot interface with a dark background and light text. At the top, there's a header 'Try it out' with buttons for 'Clear' and 'Manage Context' (which has a blue circle with the number '2' next to it) and a close button (an 'X' icon). Below the header, the chat history is shown. The first message is from the system: '@sys-person:Frasier'. The second message is from the system: 'Nice to meet you, Frasier. How can I help you? You can ask me about our store hours, locations, or flower recommendations.' followed by a location pin icon. The third message is a user input: 'what are your hours of operation?' followed by a dropdown menu showing '#hours_info' with a blue checkmark and an eye icon. The fourth message is from the system: 'For which city?' followed by a location pin icon. The fifth message is a user input: 'Seattle' followed by a dropdown menu showing 'Irrelevant' with a blue checkmark and an eye icon. Below this, the system message '@sys-location:Seattle' is shown. The final message is from the system: 'For which city?' followed by a location pin icon.

The chatbot will ignore us entirely and ask us for the city again. Not good!

Lab 11: Master Slots

We provided a valid city! Why won't the chatbot take that for an answer?

Because we are not handling the *@sys-location* scenario.

We can approach this issue in a few ways, including reverting to a dedicated node that handles non *@location* cases or configuring a *Not Found* response in the slot to inform the user of which locations we have.

We'll cover *Not Found* in the next module, so for now, we'll take the easiest route here and just make the slot optional. This way we'll be able to handle all three scenarios we had before, but this time from a single node.

Let's see this in action.

1. **Remove the question from the slot** within *Hours of Operation* to make the slot optional.
2. In the response section, **add a response with the condition @sys-location** and the response:

Unfortunately, we don't have a store in @sys-location. 😞 To date, we have stores in Toronto, Montreal, Calgary, and Vancouver.

3. **Add a final response with the condition true**, and the response:

Our hours of operations are listed on our Hours page.

The resulting *Hours of Operations* node will look as shown in the image below.

Lab 11: Master Slots

Then check for

[Manage handlers](#)

	CHECK FOR	SAVE IT AS	IF NOT PRESENT, ASK	TYPE		
1	@location	\$city	Enter prompt	Optional		

[Add slot](#)

Assistant responds

	IF ASSISTANT RECOGNIZES	RESPOND WITH		
1	\$city:Toronto	Our Toronto store is open Monday to		
2	\$city:Montreal	Our Montreal store is open Monday t		
3	\$city:Calgary	Our Calgary store is open Monday to		
4	\$city:Vancouver	Our Vancouver store is open everyda		
5	@sys-location	Unfortunately, we don't have a store		
6	true	Our hours of operations are listed on		

Lab 11: Master Slots

4. Test that the node is working as expected. Try the following conversations (clicking *Clear* to start a fresh conversation):

(Enter your name)

hours of operation

hours of operation in Seattle

hours of operation for Vancouver

All three scenarios should work as you expect. And they were all handled with a single node thanks to the power of slots and multiple conditioned responses. So much better than having a tree of nodes.

Exercise 3: Reimplement Location Information

Repeat the whole process in Exercise 2 for the *Location Information* node adjusting the responses accordingly.

If you'd like to speed up the process, you could duplicate *Hours of Operation*, rename it to *Location Information*, then change the condition and responses to be about locations instead of hours, and finally delete the old *Location Information* node and its children.

Finally, test the *Location Information* node with the following text (again, click *Clear* in between each test):

(Enter your name)

list of locations

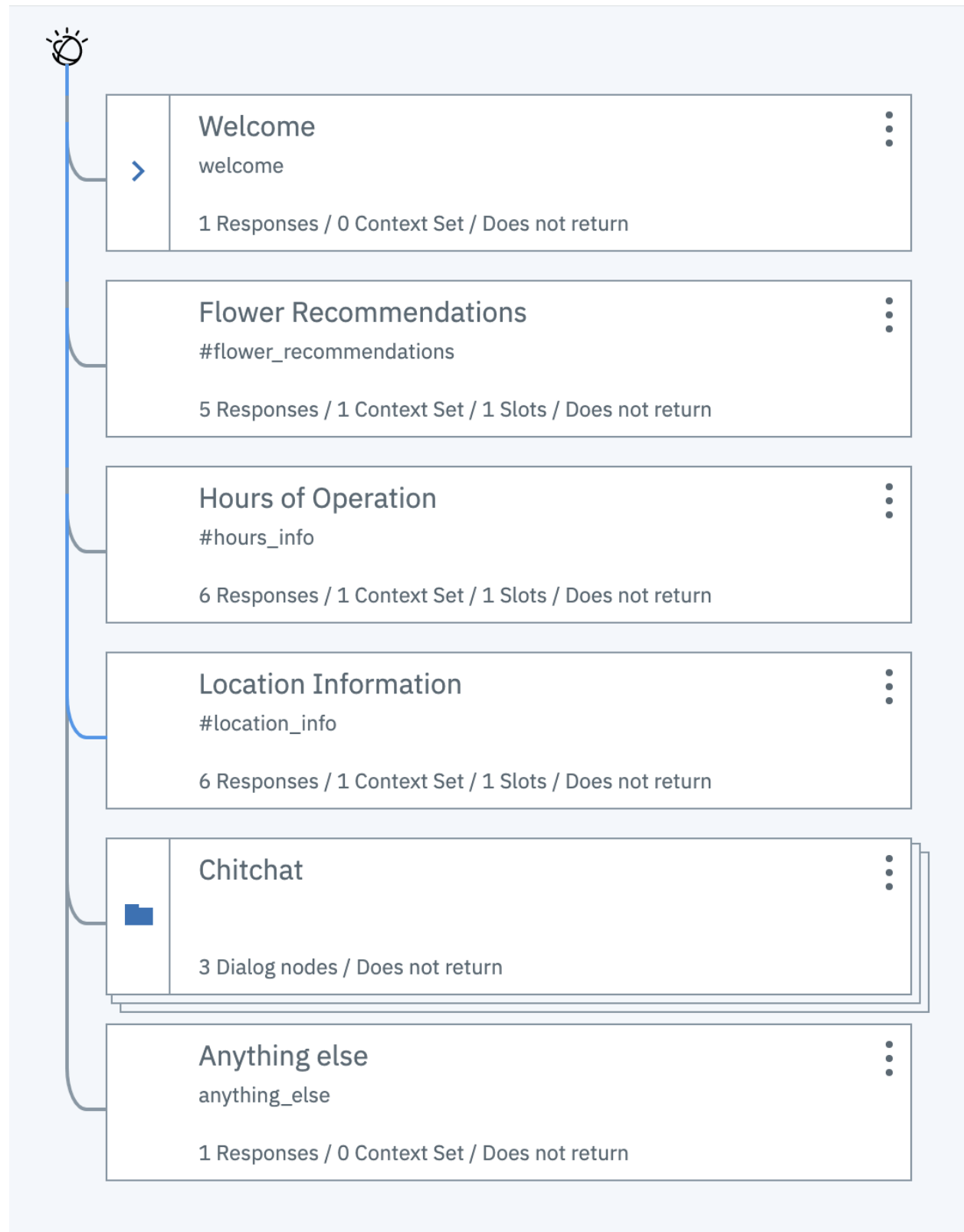
what's the address of your Seattle store?

what's the address of your Vancouver store?

All three scenarios should also work as expected.

The image below shows what the dialog will look like after all the changes have been correctly implemented.

Lab 11: Master Slots



Lab 11: Master Slots

If you are lost or encountered problems when testing the chatbot, you can [download the JSON file](#) for the dialog skill we developed so far.