Exercise 1: Respond to hours of operation requests

Chit chat interactions are necessary to make our chatbot more pleasant and human-like. However, what makes the chatbot actually useful is its ability to answer domain-specific questions. That is business-related questions.

We defined intents for people inquiring about hours of operation and addresses of our fictional florist chain and even created an entity to be able to provide location-specific answers. However, much like the chit chat intents, intents alone don't offer responses to customers. We'll need to create nodes to handle these two business-specific intents.

Create the parent node

We'll start by creating a node for hours of operation. Follow these steps:

1. **Select the *Welcome* node** and **click on *Add node***. This will create an empty node just below the first node in the dialog.

2. **Set the node name to Hours of Operation** and **use #hours_info for the node condition**. This will ensure that the node will be executed when the user is inquiring about shop hours.

3. **In the response, enter:**

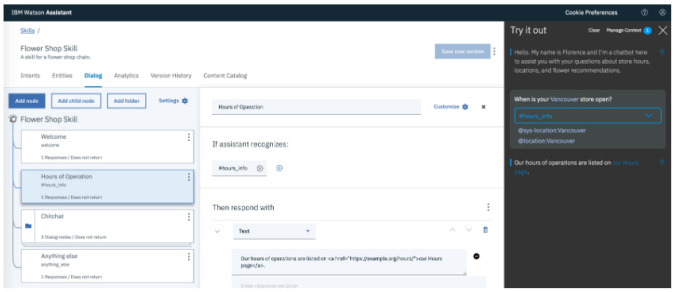
Our hours of operations are listed on our Hours page.

Notice how HTML code is allowed in responses, enabling us to provide more interactive and useful textual answers.

Next, head over to the *Try it out* panel and test that it works by asking:

When is your Vancouver store open?

as shown in the image below.



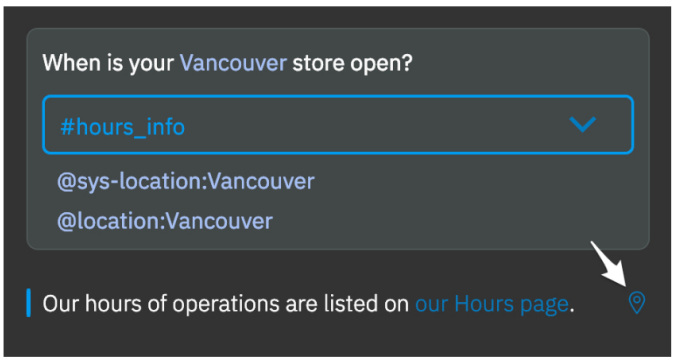
This works and it provides a somewhat useful answer to the user (assuming we are pointing them to a page with the right information listed). However, it feels... not very smart.

After all, the user asked us about a specific location. We even detected it with the @location entity (as well as the system entity) and then proceeded to ignore it, opting instead for a generic answer. We can do better than that. (Close the *Try it out* panel to gain some breathing room as we work on the dialog.)

In order to handle this case properly, we'll have to consider three possible scenarios. One in which a location is specified and it's one of our locations, a second one in which the city is specified but we don't have a location, and a third one in which the user just asks about hours of operation in general without indicating a city.

This is a classic use case for child nodes. We'll use our current node to capture the hours of operation request, and then jump to the child nodes to decide how to handle the request on the basis of the specific location information that was or wasn't provided.

By the way, as the complexity of your dialog grows, you might have a hard time finding which node executed the response you're seeing during troubleshooting. To help you out, you can click on the map pin icon next to the response, and the current node will be highlighted for you.



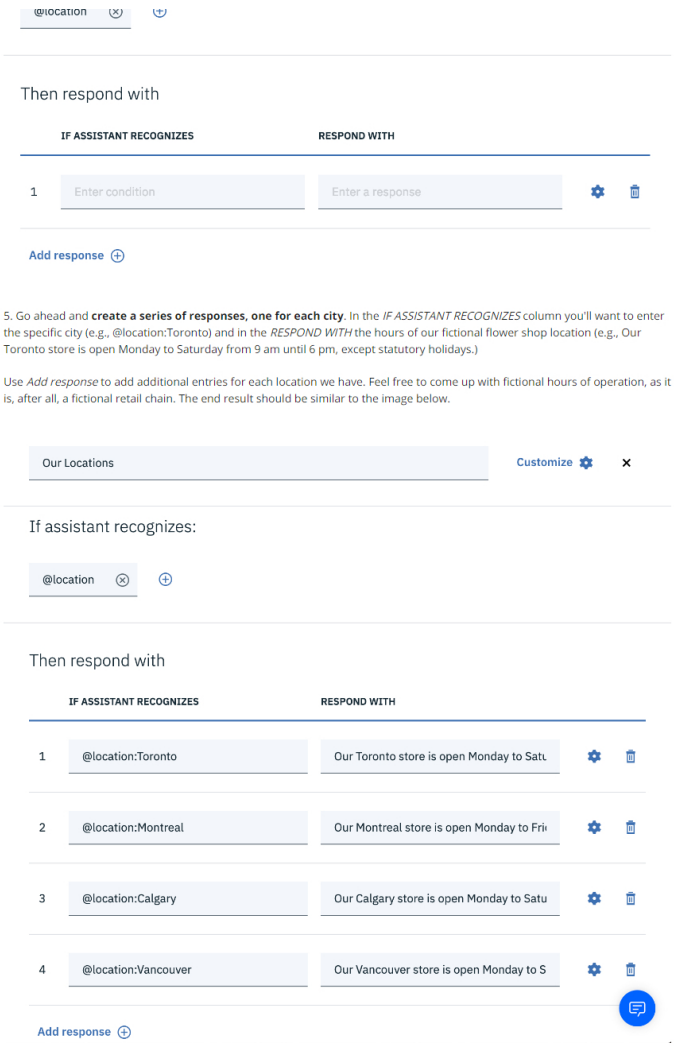
Create the Our Locations child node

1. **Delete the response from our *Hours of Operation* node** by clicking on the trash bin icon in the *Then respond with* section. We do that because we don't want this parent node to provide the answer. We'll let the child nodes decide what's the right response.

2. With the *Hours of Operation* node selected, **click on *Add child* node**. This creates the first child node. We'll use it for the case of the user providing us a city for which we have a flower shop. So go ahead and name it Our Locations.

3. **Set the condition to @location**, as we want to execute this node only if the user is inquiring about hours of operation for one of our locations. As a reminder, a child node is only executed if the parent node's condition is true or if it's jumped to from elsewhere. This means that if we are executing this child node, two conditions will actually be true as far as the user input is concerned: 1) The intent will be #hours_info; 2) The input will contain the @location entity. Knowing this allows us to provide very specific responses.

4. We need a way to offer a different response for each city, so we need to **enable *Multiple conditioned responses***. To do so, click on the *Customize* link within our child node. Switch on *Multiple conditioned responses* and click *Apply*. You'll notice that now we have the ability to attach a condition to each response, as shown below.



5. Go ahead and **create a series of responses, one for each city**. In the *IF ASSISTANT RECOGNIZES* column you'll want to enter the specific city (e.g., @location:Toronto) and in the *RESPOND WITH* the hours of our fictional flower shop location (e.g., Our Toronto store is open Monday to Saturday from 9 am until 6 pm, except statutory holidays.)

Use *Add response* to add additional entries for each location we have. Feel free to come up with fictional hours of operation, as it is, after all, a fictional retail chain. The end result should be similar to the image below.

It's worth noting that if the hours of operations where the same for all locations, we could have saved the trouble of switching to multiple responses and simply included @location in our response. (e.g., Our @location store is open Monday to Saturday from 9 am until 6 pm, except statutory holidays.)

This would automatically output the detected entity value back to the user in the response. So when enquiring about Calgary, the user would receive the response Our Calgary store is open Monday to Saturday from 9 am until 6 pm, except statutory holidays. Of course, if stores have different hours of operation, we need to opt for the multiple response approach like we did here.

Create the No Such Location child node

Now that we have a child node capable of handling hours of operation for our particular locations, we can focus on the case in which a user provides a city (detected through @sys-location), but we don't have a store in that city.

1. With the *Our Locations* node selected, **click the *Add node* button** to add a peer/sibling node below it.

2. **Call this node *No Such Location* and add @sys-location as the condition.**

3. In the text response, enter: Unfortunately, we don't have a store in @sys-location.☹ To date, we have stores in Toronto, Montreal, Calgary, and Vancouver.

Notice that @sys-location will also detect our own locations, however, this node will never be executed for our own locations, because it comes after the *Our Locations* node. The execution of the dialog always goes top to bottom and stops at the first node for which the condition is true.

This is one instance were the order of the nodes matter. If we placed *No Such Location* above *Our Locations*, the very generic @sys-location condition would be true even for cities in which we do have a store, and therefore the execution would stop at this node, overshadowing *Our Locations*.

As a general rule, when organizing your dialog, always place nodes with specific conditions at the top and ones with more generic conditions at the bottom.

(If you are wondering why we bother with @location at all and not just use @sys-location even for our own locations, revisit Module 3 where we discussed the tradeoffs of *@sys-location*.)

Create the Location Not Provided child node

We now have two child nodes to handle users asking about hours of operation for a specific location (whether we have a store there or not). However, we also need a child node to handle the case in which the user didn't specify a location.

1. With the *No Such Location* node selected, **click the *Add node* button** to add a peer/sibling node below it.

2. **Call this node Location Not Provided.** Set the condition to true.

Here is why. When the user asks *What are your hours of operation?* the #hours_info intent is detected, so we enter the parent node *Hours of Operation*.

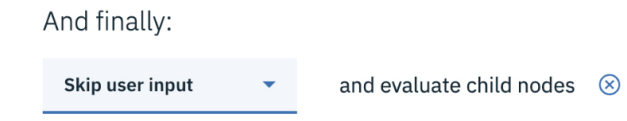
The *Our Location* child node is then first evaluated. We fail its condition because the user didn't specify any location, so the next child node is considered for execution. We fail that condition as well because no @sys-location is detected either. So we finally consider this third child node.

Since the condition is set to true it will automatically be executed. This is exactly what we want to happen since at this point we know the user wants to know the hours of operation but no location was provided. (If we left the condition empty, we'd get an error because no child node was able to match the user request.)

3. We need a generic answer for when no location is specified, so **go ahead and reuse the message we had originally.** Our hours of operations are listed on our Hours page.

4. Before we can test it all out, we need to make sure that the parent node (i.e., *Hours of Operation*) hands off control to the child nodes.

Select the parent node, and you'll notice that the *And finally* section is set to *Wait for user input*. This is not what we want. The user has already provided us with the question and we haven't responded yet. **Change this section of *Hours of Operation* to *Skip user input*.** This will hand off the execution to the two child nodes we just created.



5. Click on the *Try it* button and try the following inputs (one at a time):

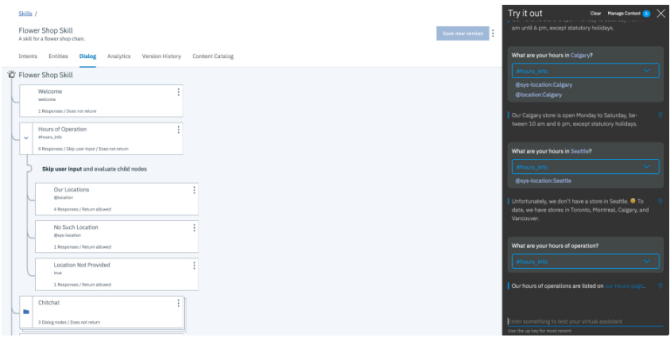
What are you hours of operation in Toronto?

What are your hours in Calgary?

What are your hours in Seattle?

What are your hours of operation?

You should see a proper response for each of these inputs.



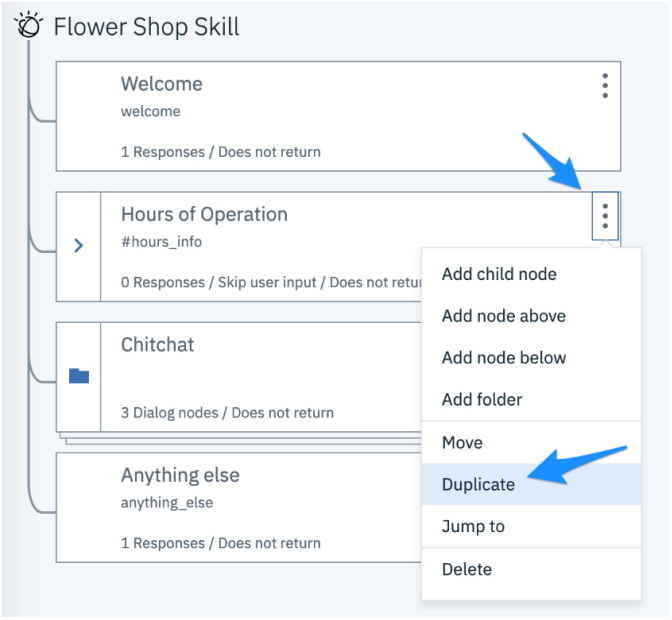
Exercise 2: Respond to address requests

Our little chatbot is getting more useful by the minute. We now need to handle location address requests. And guess what? It's no different in terms of how this works. We'll have a parent node and three children to distinguish each scenario. By the way, at any time, you can click on the arrow next to the *Hours of Operation* node to collapse or expand its children. **Collapse its child nodes now** to gain some breathing room.

In the new parent and child nodes to handle address request will need to change one condition (#location_info instead of #hours_info) and change the responses from hours of operation to actual addresses. Overall though, the structure is identical. So we could redo the process in Exercise 1 above, step by step, or we can be more efficient and simply duplicate *Hours of Operation* and change this copy to our needs.

We'll opt for this more efficient route:

1. Click on the three vertical dot icon to the right of the *Hours of Operation* node and **select *Duplicate* to make a copy of *Hours of Operation* and its children.**



2. Select the *Hours of Operation - copy1* node that was generated. **Change its name to *Location Information* and its condition to #location_info.** Its name will allow us to distinguish it from its remarkably similar sibling *Hours of Operation*, and the condition will ensure that Watson will only execute the node when the user asks for an address not hours of operation.

3. Next, we'll need to **change the responses in two child nodes within the *Location Information* tree**.

We don't need to change the *No Such Location* response because the one we have applies to both requests for hours of operation and for address requests from the user (i.e., Unfortunately, we don't have a store in @sys-location. 😟 To date, we have stores in Toronto, Montreal, Calgary, and Vancouver.)

Feel free to get creative but here is the type of response you should assign to each city in *Location Information > Our Locations*:

Our Toronto store is located at 123 Warden Avenue.

Add fictitious address for all the cities in that node.

Likewise, in the more generic *Location Information > Location Not Provided* change the response to:

Our store locations are listed on our site on the stores page.

4. Open the *Try it out* panel, press *Clear* to start a new conversation, and **test out a full conversation** a user might have with our chatbot. Enter in succession the following input.

hello

where are you stores located?

what are your hours of operations in Montreal?

thank you

bye

If you followed the instructions so far, you should have a pretty neat conversation. We can, of course, flesh out our chatbot much more, but if you got to this point, you have mastered the fundamentals of what you need to know to create something useful that cuts down of many common inquiries from your customers.

We'll soon see how to deploy the chatbot, and then tackle more advanced topics in the process of improving the chatbot usefulness and apparent degree of intelligence.

Are child nodes really necessary here?

Technically speaking we don't need child nodes to handle the two scenarios we implemented above. We could simply add multiple conditional responses to the parent nodes, and add responses for each of the cities, @sys-location, and the catch-all true case, all from within the same node.

However, I wanted to show you how to work with child nodes, the importance of their ordering, and their flexibility. If the logic was more complex than just a generic response, having a dedicated child node to handle it would likely be a good idea, anyway. In some complex chatbots, you might even have child nodes that have their own child nodes.

Later in the course, we'll get rid of child nodes in favor of something called *Slots*. For now, please keep the three child nodes below both *Hours of Operation* and *Location Information*, as you defined them in this lab.

Mark as completed