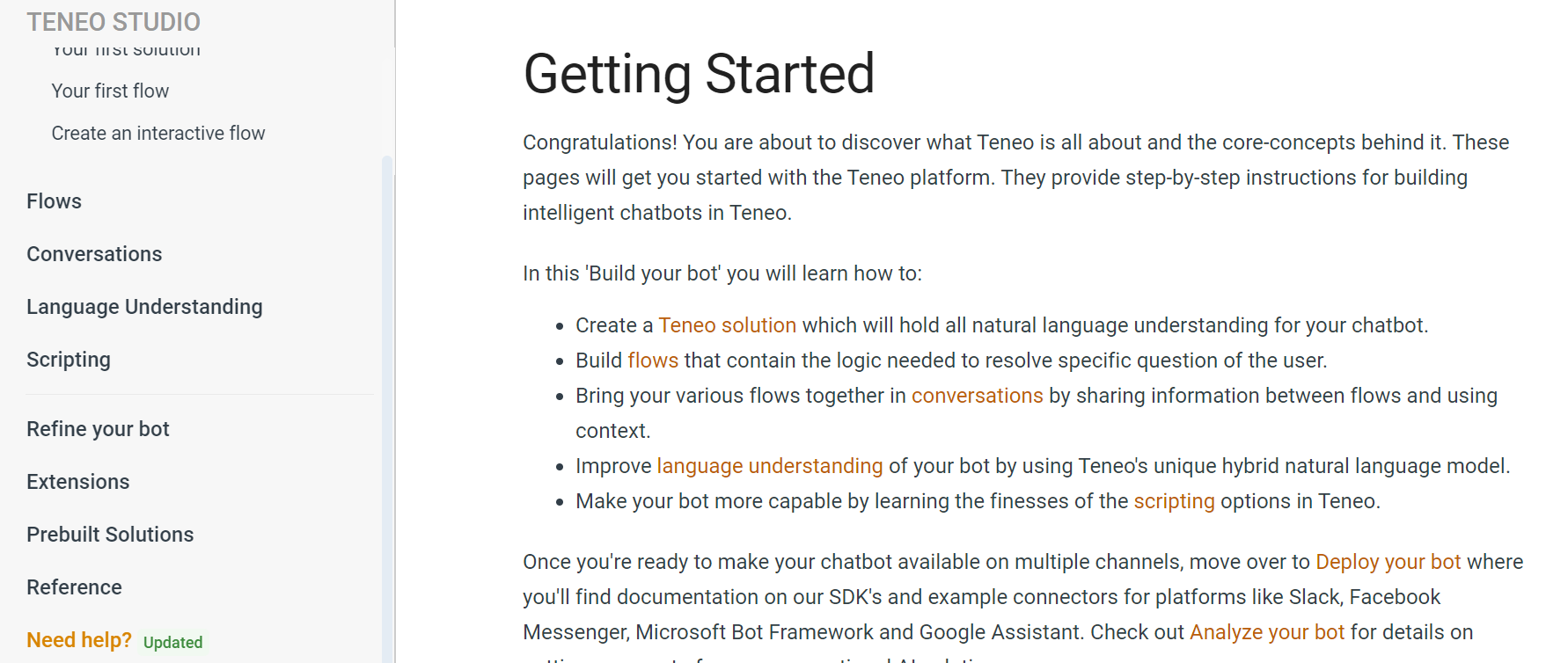
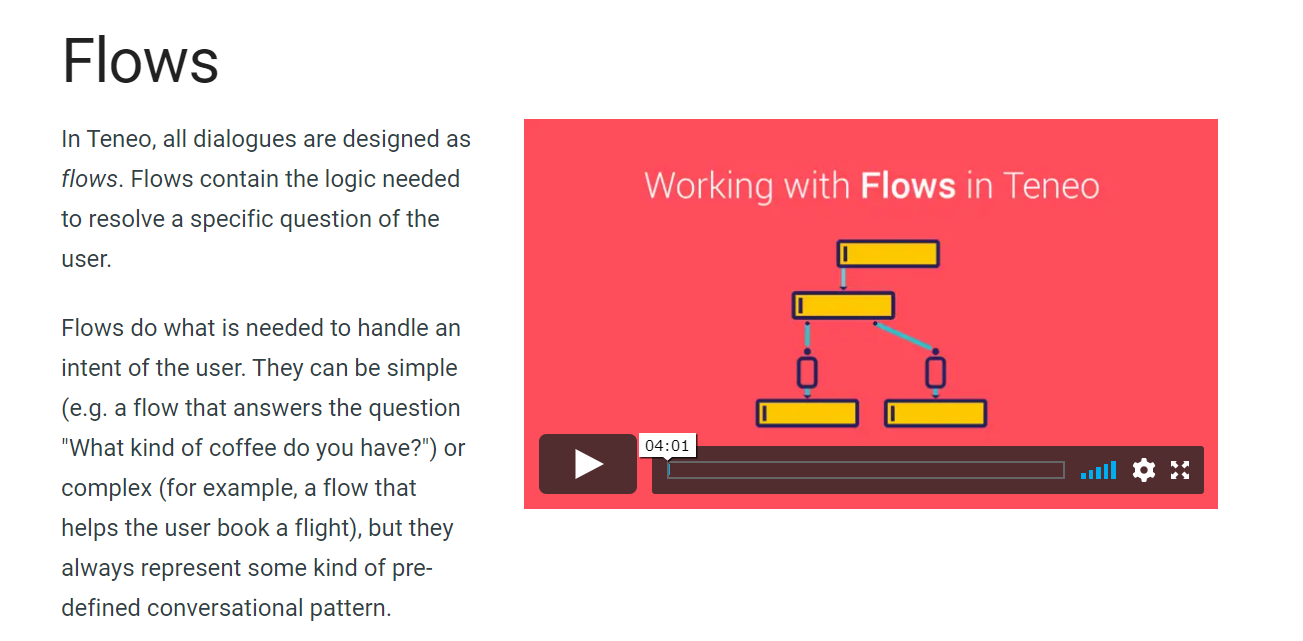
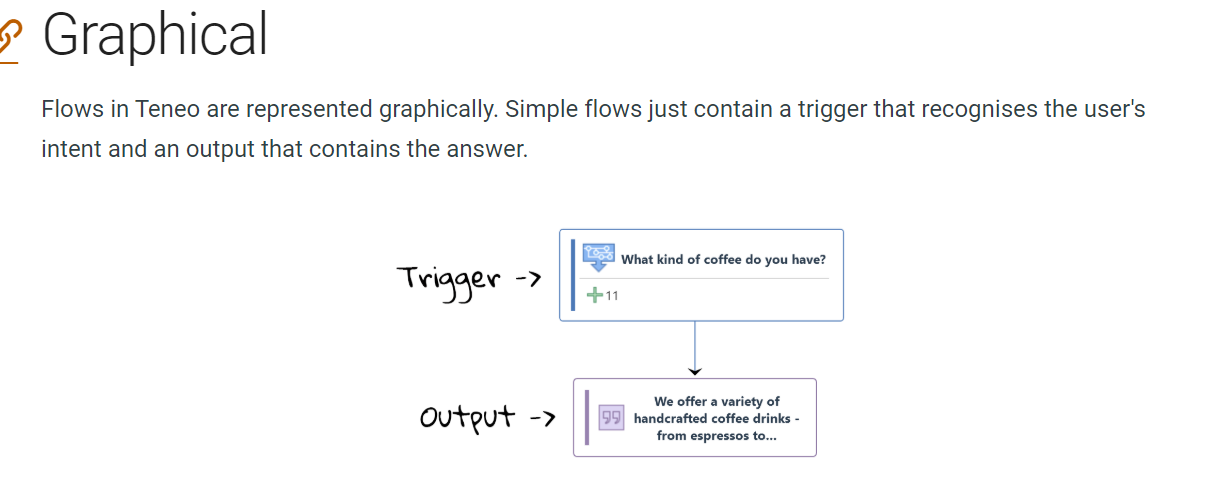
Reference : <https://www.teneo.ai/studio/flows/concepts/entities>

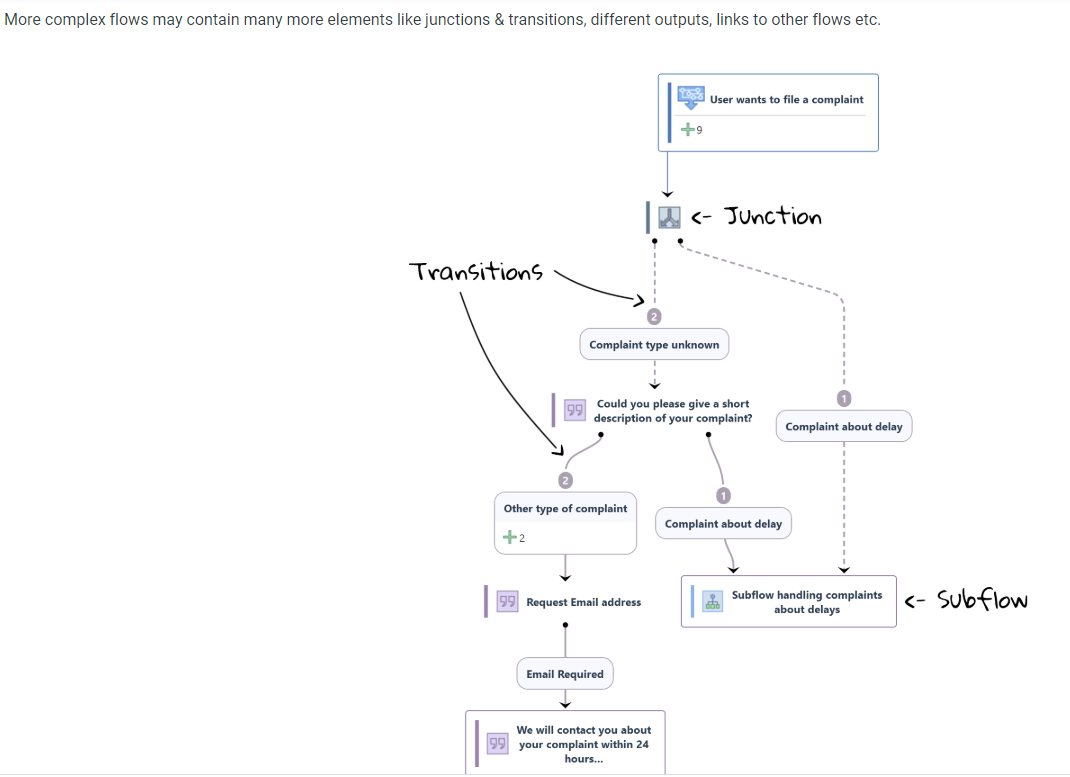






A flow in teneo consist of different component.

* Trigger : Which tell teneo what kind of input or intent activate the flow.
* Output : Specify how the bot should res



**Getting Started**

In Teneo, all dialogs are designed as flows. Flows contain the logic needed to resolve a specific question of the user. They can be simple (e.g. a flow that answers the question "What kind of coffee do you have?") or complex (for example, a flow that helps the user order a coffee beverage), but they always represent some kind of pre-defined conversational pattern.

1. **Hands On First Example :**

* Created first flow from icon on the left of what coffee do you offer.
* A simple Flow consist of
* Class Trigger which consist of user question/input or intent.
* Output which is the answer of the trigger intent.
* We can run this and test this flow.

1. **Hands On Create an interactive flow**

* In this we learn to create a flow for password reset
* Used here class Trigger Output and Branch which has condition to generate another output based on yes or no
* In these first tutorials you have familiarised yourself with some of the core concept of Flows in Teneo, like *triggers*, *outputs*, *transitions* and *language conditions*.

**Flows**

In Teneo, all dialogues are designed as flows. Flows contain the logic needed to resolve a specific question of the user.

Flows do what is needed to handle an intent of the user. They can be simple (e.g. a flow that answers the question "What kind of coffee do you have?") or complex (for example, a flow that helps the user book a flight), but they always represent some kind of pre-defined conversational pattern.

# Intent triggers

###### Table of contents

* + - [Class Triggers](https://www.teneo.ai/studio/flows/concepts/triggers#class-triggers)
    - [Syntax Triggers](https://www.teneo.ai/studio/flows/concepts/triggers#syntax-triggers)
    - [Linking triggers](https://www.teneo.ai/studio/flows/concepts/triggers#linking-triggers)
    - [Which trigger should I use?](https://www.teneo.ai/studio/flows/concepts/triggers#which-trigger-should-i-use)
    - [Context restriction](https://www.teneo.ai/studio/flows/concepts/triggers#context-restriction)
    - [Order group](https://www.teneo.ai/studio/flows/concepts/triggers#order-group)
    - [Multiple triggers](https://www.teneo.ai/studio/flows/concepts/triggers#multiple-triggers)

The main role of an intent trigger is to tell Teneo what kind of inputs, or intents, should activate a flow. There are two types of intent triggers in Teneo: *class triggers* and *syntax triggers*

**Class Triggers**

Class triggers (or *Machine Learning Class Intent Triggers* if you want to use the more official name) use a machine learning model (a classifier) to determine what an input is about. You train a class trigger by providing training phrases (called 'learning examples' in Teneo) that are as close to real user inputs as possible.

Ideally you would use actual user inputs, but to get started during development you can just use some example inputs that you can think of from the top of your head. We recommend to start with 10-20 example inputs per class trigger. This will result in good performance, but you will notice the model will improve when you add more examples once you start testing and collect more inputs

## **Syntax Triggers**

Syntax triggers (officially called *Language Condition Syntax Intent Triggers*) use a different method to find out what kind of inputs should activate a flow. They make use of a [language condition](https://www.teneo.ai/studio/language-understanding/concepts/language-conditions), which is expressed using a [condition syntax](https://www.teneo.ai/studio/reference/condition-syntax). You can either hand-craft the condition yourself, or let Teneo automatically create it for you from a set of examples, after which you can edit the condition. For an example of different ways to create and edit a language condition, go here: [How to create a syntax trigger](https://www.teneo.ai/studio/flows/how-to/create-a-syntax-trigger).

The possibility to use both *class* and *syntax* triggers provides you with the power of machine learning and the precision of condition syntax. This is what we call our [hybrid model](https://www.teneo.ai/studio/language-understanding#hybrid-model). It is explained in depth in the [Language Understanding](https://www.teneo.ai/studio/language-understanding) section.

1. **Hands On Create a Syntax Trigger ???**

* In this we create a flow with Syntax trigger.
* In this the user intent is entered and then the condition of the user input is generated using the (generate the language condition from condition bar.)

## **Linking triggers**

Class triggers can be linked to syntax triggers. By linking them, the example inputs of the syntax trigger are also used as training phrases for the class trigger.

**Context restriction**

You can restrict your trigger to only fire in certain contexts, for instance after a certain topic has already been introduced. Consider the following dialog:

**Order group**

When Teneo tests a user input against your solution's triggers, it does so in a certain [order](https://www.teneo.ai/studio/conversations/concepts/ordering), and stops at the first match that fully satisfies all criteria.

**Multiple triggers**

Flows can have several independent triggers. Using multiple flow triggers makes it possible to create diverse triggers with different degrees of complexity and specificity linking to the same flow. For example, it is possible to create a flow with a class trigger and also a syntax trigger with a more complex condition that matches very precise inputs. These triggers can have their own [listeners](https://www.teneo.ai/studio/flows/concepts/listeners) (to extract additional information from the input) and, in case of syntax triggers, would have their own ordering.

# Outputs

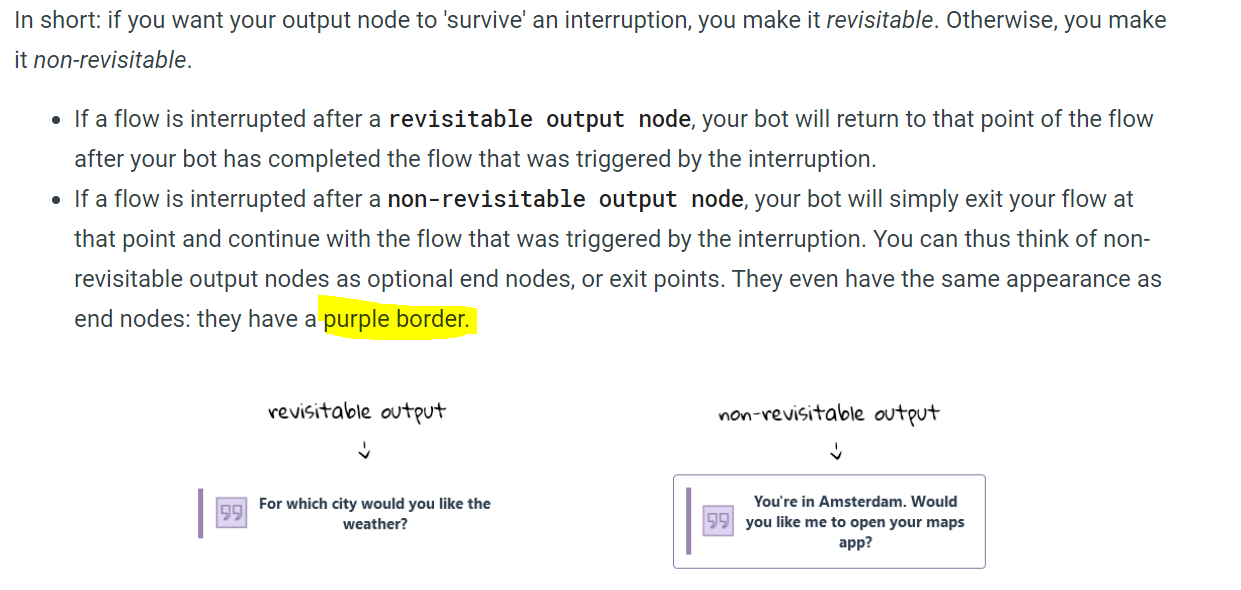
Output nodes are flow nodes that specify how the bot should respond. Most importantly they contain the answer texts, but they also let you specify various other propertie

**Answer texts**

The most common thing to add to an output node is an answer text: a response from your bot in natural language. Answer texts are specified in the 'Answers' panel. You may add several answer texts that are phrased slightly differently.

**Revisitability**

Still in the 'Answers' panel, you can specify if an output node can be revisited or not. Revisitability has to do with interruptions, for example situations where your bot has asked a question to the user and the user responds with a request for clarification, switches topic, or simply responds in a way that is not understood.



## **Resume prompts**

If you add a resume prompt to an output node, that prompt will be given instead of the ordinary answer texts when the node is revisted after an interruption. You add resume prompts in the 'Resume Prompts' panel.

## **Skip conditions**

Skip conditions are used to prevent your bot from asking for information that is already known.

For example, if a user says: I would like an ice cream please, you want your bot to ask: Ok, what flavor would you like?. But when the user says: I would like a ***strawberry*** ice cream please, you want your bot to **skip** the question: Ok, what flavor would you like?.

With a skip condition, you can specify that an output node should be skipped (or jumped over) if a particular condition is true.

You add skip conditions to your output node under its 'Skip conditions' panel. There you specify the condition when it should be skipped (e.g. if the variable for flavor has already been set), and where it should skip to, i.e. which transition to follow after the output has been skipped.

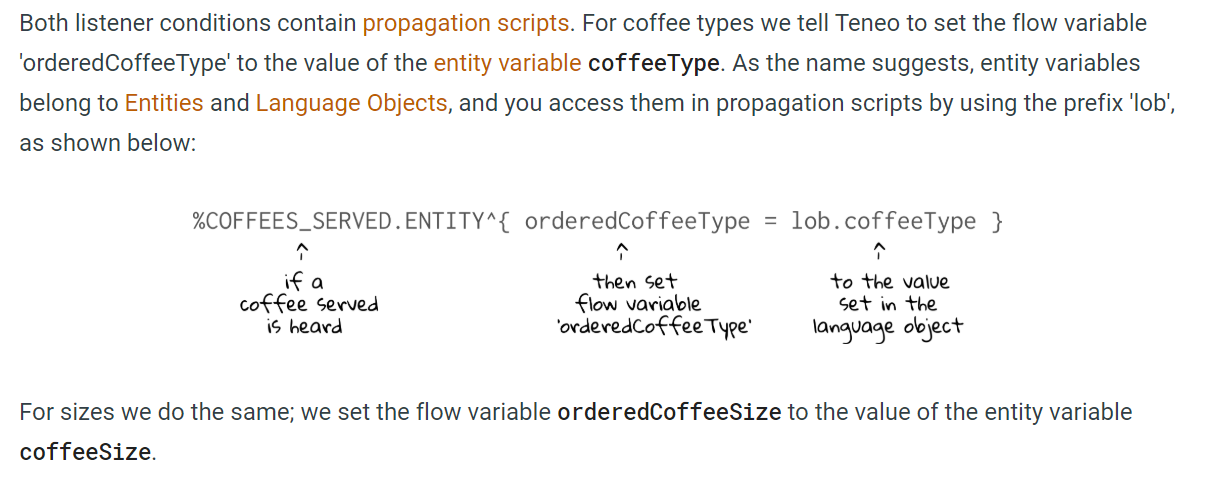
The page [How to build a slot filling flow](https://www.teneo.ai/studio/flows/how-to/build-a-slot-filling-flow) shows how to build a flow that can handle coffee orders and it uses skip conditions to avoid redundant questions.

1. **Handson : Build a slot filling flow**

The page [How to build a slot filling flow](https://www.teneo.ai/studio/flows/how-to/build-a-slot-filling-flow) shows how to build a flow that can handle coffee orders and it uses skip conditions to avoid redundant questions.

* Create Custom Entity like CoffeeType CoffeeSize
* Create Skip condition : this is added at output node to avoid redundancy.
* Use of transition condition here
* Add Flow Variable to print coffee type and coffee size selected by user
* Add Flow listener to match the condition from user input using Entity to its variable

This is done using Propagation Script



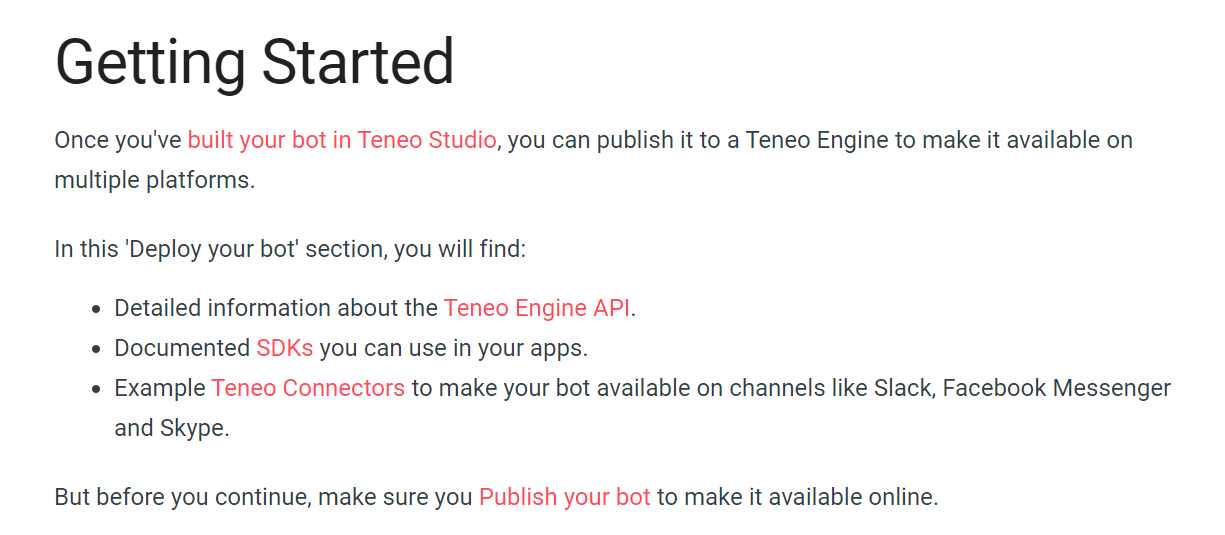
## **Output parameters**

Output parameters can be used to provide your app with information that has no place in the regular answer text. For example, suppose a bot can provide weather information in a mobile app. The answer text of an output might contain a humanlike description of the weather (*It's quite sunny in Amsterdam at the moment, make sure to wear sunscreen!*). But in addition to that, the app could display a widget with more detailed weather info. The information that needs to be displayed in the widget could be passed on as JSON in an output parameter. Each output node can have as many output parameters as you like, each parameter needs to have a name and a String as value.

You add output parameter to your output node under its 'Output Parameters' panel.

**URLs and emotions**

Each output node has a separate URL field. This URL is included as a separate field in your bot's response. The frontend application can use this field to for example automatically open a webpage. The same is true for 'emotions': each answer text can have an emotion associated with it. This emotion is included as a separate field in your bot's response and can for example be used by frontends that display avatars.



# Entities

In order for your bot to understand what the user said, some words of the user's utterance are more **important** than others. Typical examples for such important words include so-called **named entities** like cities or product names. One way of catching those is to use entities.

**What is an entity?**

Each entity covers a collection of one or more words called 'entries', which all have some characteristic in common. For example, they all denote cities in the Netherlands:

In order to enhance natural understanding, the entries can have variables carrying additional information about each entry. For example, different names may denote the same city, like "Den Haag" and "The Hague". With variables it is possible to map these different variants to the same city name. Another example is the code of the nearest airport, which is useful to have in flight booking scenarios.

In the entity examples above, all entries are strings. Note that entries may also consist of other entities or [language objects](https://www.teneo.ai/studio/language-understanding/concepts/language-objects).

**How to use entities**

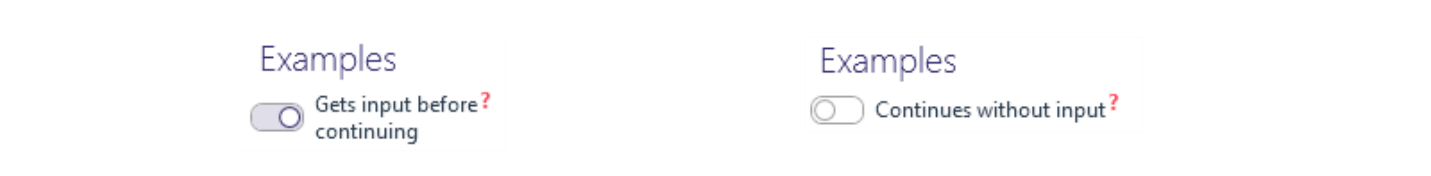
In order to use the entity covering the cities of the Netherlands in your flow, simply add %CITIES\_NETHERLANDS.ENTITY to your [language condition](https://www.teneo.ai/studio/language-understanding/concepts/language-conditions). The variables city and airportCode can be accessed using [attached scripts](https://www.teneo.ai/studio/language-understanding/concepts/attached-scripts).

# Transitions

Transitions connect flow nodes with each other. They determine the paths a dialogue can follow. If there are multiple transitions leaving a node, all transitions are tested in a certain order and the first one whose condition is satisfied is chosen.

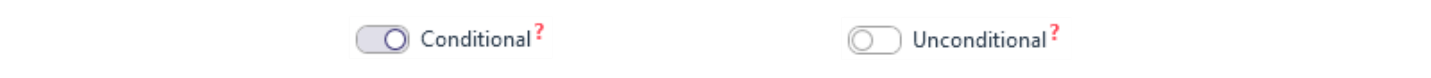
**New input or not?**

Some transitions wait for a user input while others do not. You specify whether a transition should wait for a new input or not with a toggle in the 'Examples' panel.

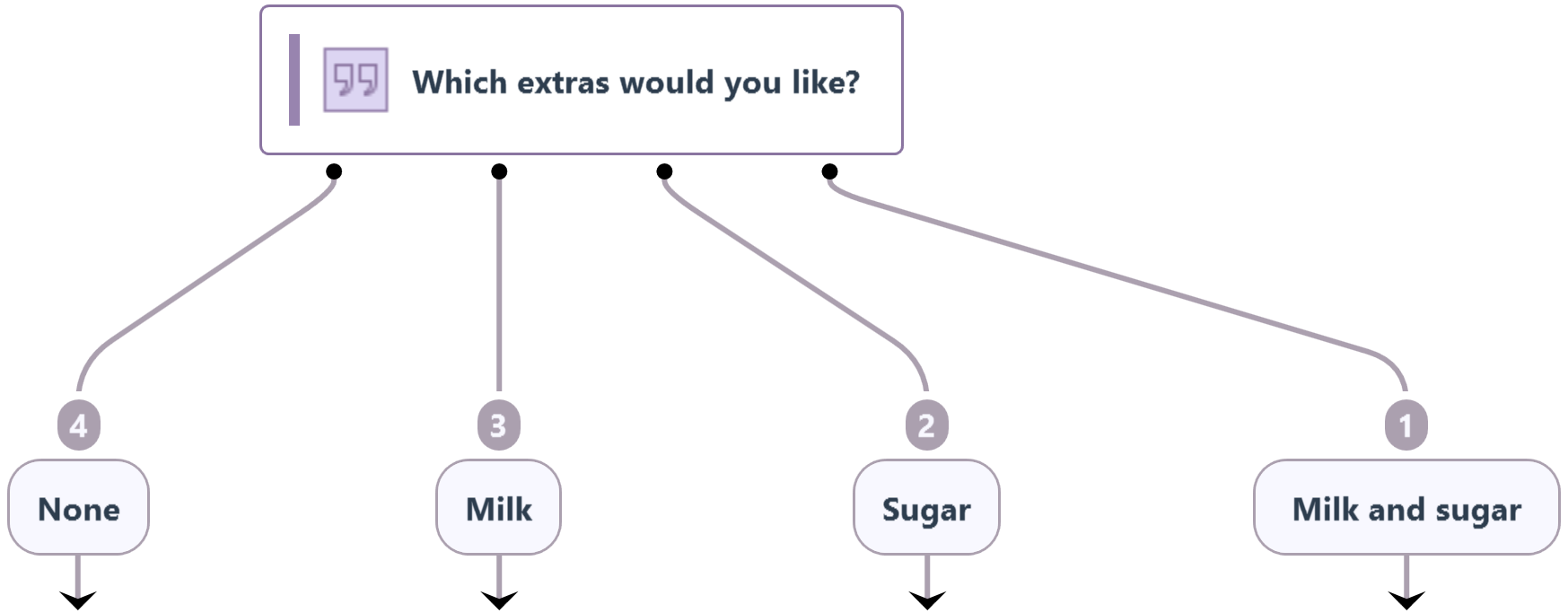
 

## **Conditional or not?**

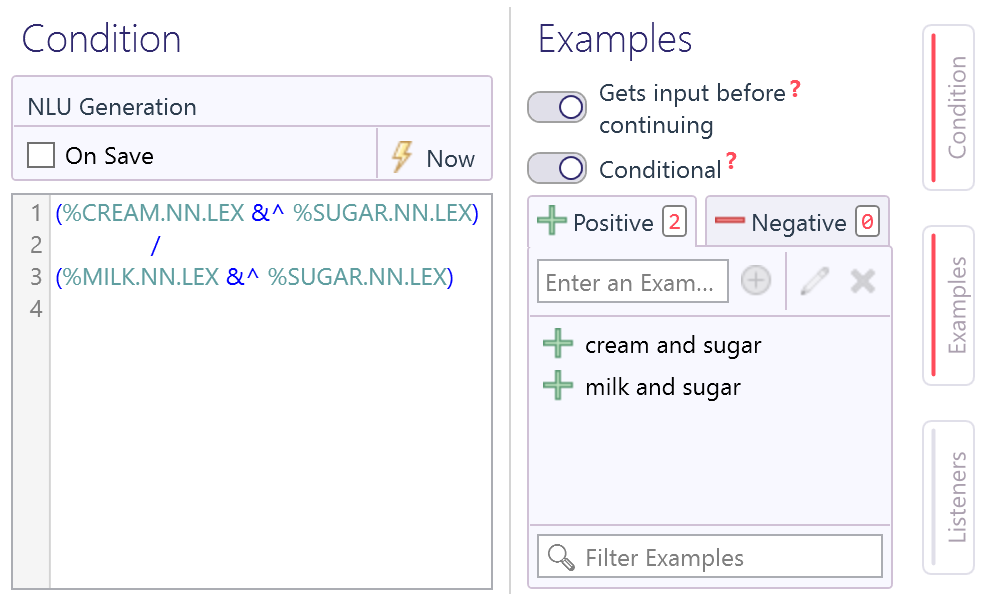
Transitions can be conditional or unconditional. If a transition is conditional, Teneo will evaluate if a condition is met before it follows the transition. You specify conditionality with a toggle in the 'Examples' panel:



Just like [syntax triggers](https://www.teneo.ai/studio/flows/concepts/triggers#syntax-triggers), transitions use [language conditions](https://www.teneo.ai/studio/language-understanding/concepts/language-conditions) to evaluate if the input of a user meets the criteria to follow a transition. You can either hand-craft the condition yourself, or let Teneo automatically create it for you from a set of examples, after which you can edit the condition. Say that you want to capture extras together with a coffee order:



The language conditions for the 'Milk and sugar' branch may look like this:

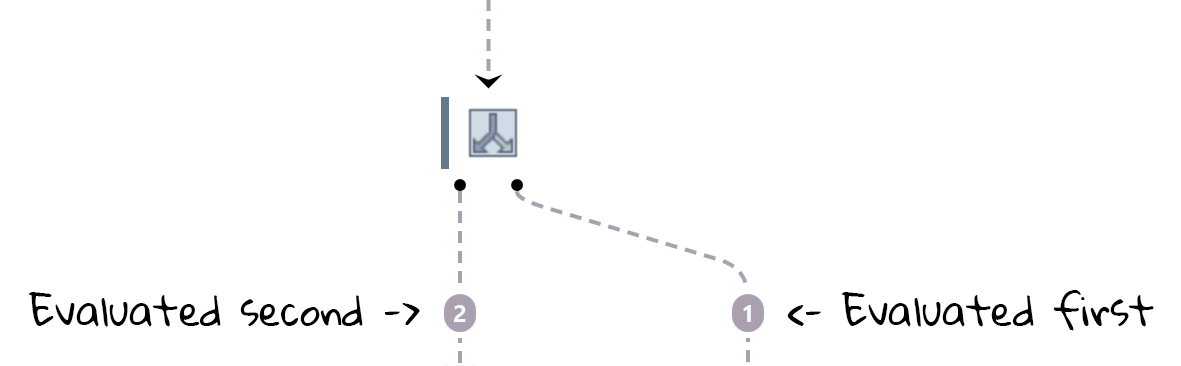


For language conditions in transitions it is usually enough to focus on the crucial content-bearing keywords. This way, less important words that might exist in the user's input, are ignored.

You can also use **Groovy code** in the condition field. The code should be embraced by curly brackets, e.g. {userAge > 18} (which requires the value of the variable userAge to be larger than 18). Language conditions that consist of Groovy code are often used in transitions that continue without user input.

## **Order**

If multiple transitions have the same node as their starting point, you specify the order in which they (or rather: their conditions) should be tested. The order is displayed in the flow graph, as a number on the transition. You change the order in the 'Examples' panel.



When setting up the ordering of the transitions, make sure that the more specific conditions are tested first and the more general ones last. Otherwise the more general ones will never be chosen.

## **Repeatability**

You can specify whether a transition should be considered just once per session, or always. This is useful when you want the behaviour of a flow (or a point of flow) to be different depending on how many times the user visits it.

# Listeners

Listeners help your bot pick up relevant pieces of information from user inputs, even when these have not been explicitly prompted for. You can think of a listener as a fly on the wall, silently listening and registering things.

Listeners are often used to extract entities from user inputs

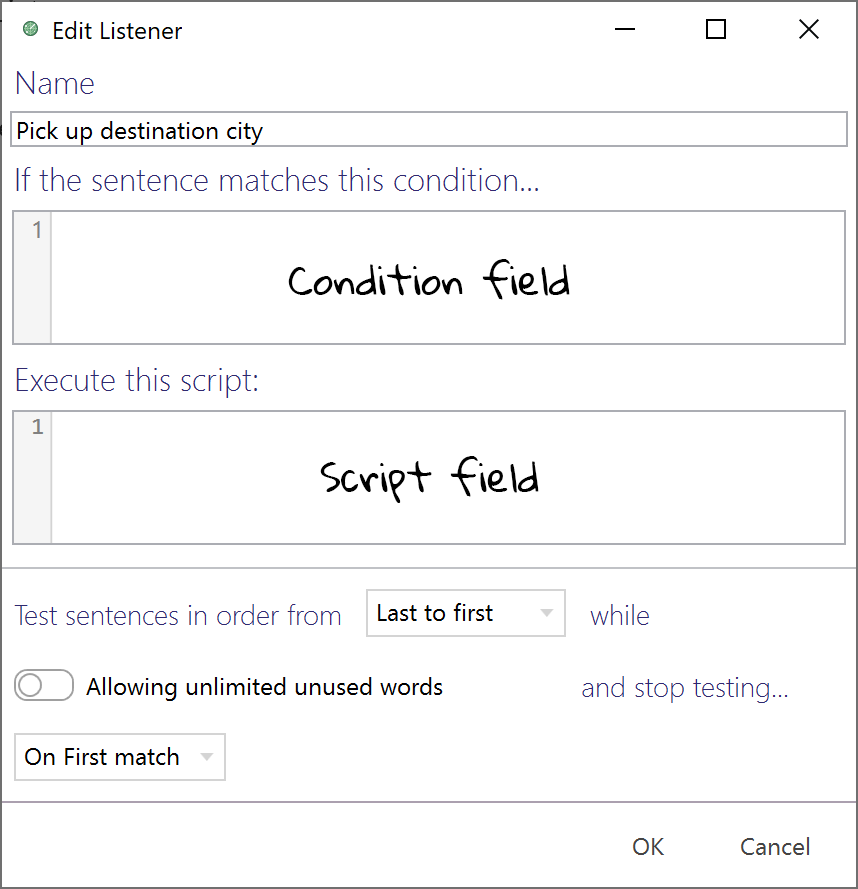
## **Types**

When you add a listener to a flow, you can choose to make it listen to all inputs that trigger or pass through the flow or to make it only listen on a particular trigger or transition.

| **Type** | **Description** | **How to add** |
| --- | --- | --- |
| **Flow listener** | Listens to all inputs that pass through a flow including inputs that trigger the flow. | - Open your flow in edit mode - Go to the 'Flow' tab on the top left - Select 'Listeners' from the purple bar on the left hand side. |
| **Trigger / transition listener** | Is defined in relation to a specific point of a flow; will only kick in for inputs of the trigger or transition where the listener was added. | - Open your flow in edit mode - Select the trigger or transition - Open the 'Listeners' panel on the right of the 'Flow' window. |

## **Components of a listener**

A listener is made up of two parts: a condition and an optional script. The condition determines what patterns should be listened for, e.g. the name of a city. The operation defines what should happen when the condition is met, for example setting a variable, or updating a counter.



**Use cases**

* **Flow listeners** are defined for a flow and listen to all inputs that trigger or pass through a flow. They are generally used when your bot should pick up certain pieces of information wherever they are mentioned in the flow dialog. [Slot filling flows](https://www.teneo.ai/studio/flows/how-to/build-a-slot-filling-flow) typically make use of flow listeners.
* **Trigger/transition listeners** are attached to a specific trigger or transition of a flow. A trigger/transition listener can be the best solution when you need to pick up something that can be interpreted differently depending on where you are in the flow. For example, if a flow should pick up two dates from a user (like an arrival and departure date for a holiday), you can add a listener to the transition where the user is asked for the arrival date and another listener to the transition for the departure date.

## **Global listeners**

In addition to flow listeners Teneo also allows you to create [global listeners](https://www.teneo.ai/studio/conversations/concepts/global-listeners) that listen to any input, independent of the flow.

# Flow variables

Flow variables allow you to remember details during the processing of a flow. Anything you store in a flow variable will be forgotten once the flow has finished processing. If you want to [share information between flows](https://www.teneo.ai/studio/conversations/how-to/share-information-between-flows), you have to use [global variables](https://www.teneo.ai/studio/conversations/concepts/global-variables) instead.

## **Create a Flow Variable**

To create a flow variable you open a flow in 'Edit' mode, go to the 'FLOW' tab of a flow and select 'Variables' in the navigation bar on the left. Click on 'Add' in the upper right corner, and provide a name and a default value. In the example below, we have two variables, flavor and size, which both have an empty string "" as the default value.

In Teneo, variables may be of any type that Groovy (and Java) supports, for example strings, booleans, maps, lists and custom objects.

## **Populate a Flow Variable**

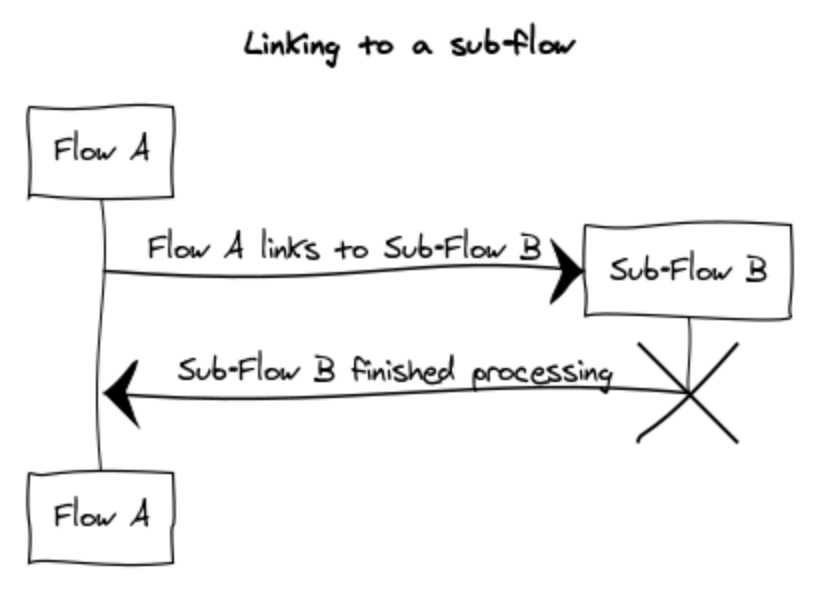
In your flow, you can populate a flow variable using a script or you can store the results of an integration or a sub-flow in a variable.

# Sub-flows

Sub-flows are often used for re-useable pieces of logic. For example, you might have a solution in which multiple flows need to ask for a date of birth. In that case, one single sub-flow that asks for the date of birth can be used. This sub-flow can then be called by any flow that needs the user's date of birth.

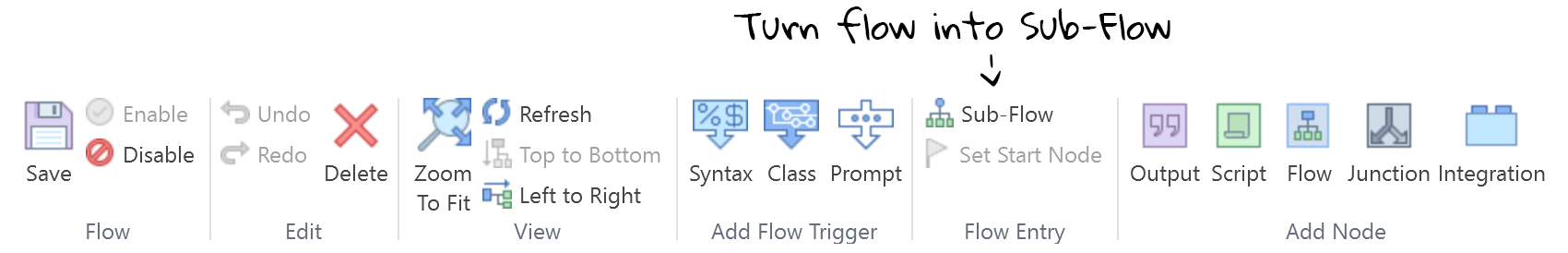
## **No triggers**

Unlike regular flows, sub-flows are not triggered by a user input. In fact, sub-flows do not have triggers at all. Sub-flows can only be activated when linked to from other flows.



Once the sub-flow has finished processing, execution will be handed back to the parent flow. One flow can link to multiple sub-flows, which will behave as if they were part of the parent flow. Sub-flows can also link to other subflows. When linking to a sub-flow you can specify which variables can be transferred back and forth. If an output in the sub-flow uses a revisitability limit and that limit is exceeded, the sub-flow and all calling flows are dropped.

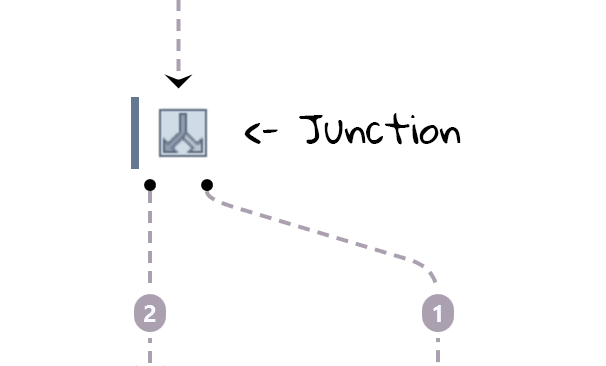
## **Creating a sub-flow**



Sub-flows are created in the same way as regular flows. However, to turn a regular flow into a sub-flow, click the 'Sub-Flow' icon in the 'Flow Entry' group of the ribbon when editing a flow. This will remove the trigger and turn it into a sub-flow entry point.

# Junctions

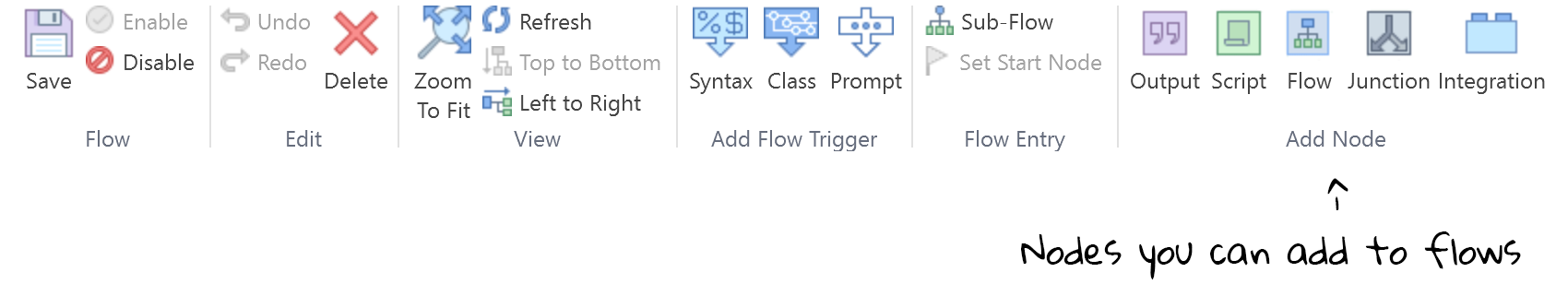
Junctions help you to model the layout and logic of your flow. Multiple transitions can point to the same junction, or a junction can be the starting point of multiple transitions.



Junctions themselves don't do anything, they don't execute scripts, don't display an output, they just offer an easy way to model your flow. By following the rules below you will get the most out of junctions.

# Other flow elements

Flows can contain many more elements than just triggers, outputs and transitions. This page highlights some of them.



1. **Hands On : Pick up an entity from an input**

Sometimes it's not enough to recognize which flow to trigger. Your bot may also need to extract some piece of information from the input to respond appropriately. Let's assume that the user wants to know where our Longberry Baristas stores are located. This is how such a conversation could go about:

**User:** Where are your stores located?  
**Bot:** Currently our only store is in New York.

Another possible conversation is the following, where the user asks whether Longberry Baristas have a store in a specific city. Here, the first part of the answer is the same, but then in the second part, the bot picked up the city the user is interested in and used it in the response:

**User:** Do you have a store in London?  
**Bot:** Currently our only store is in New York. We might consider opening additional stores in London in the future.



The LOCATION.ENTITY is pretty generous, it recognizes not only cities but also other locations like countries. If you want to be more precise and restrict the recognition to cities, you may use the [language object](https://www.teneo.ai/studio/language-understanding/concepts/language-objects) CITIES.LIST instead. However, you risk that a particular city might be missing on this list.

1. **Hands on : Add a script to your flow**

The flow we will create will suggest a random coffee drink, like so:

**User:** Suggest a drink!  
**Bot:** Well, you're putting me on the spot here. I'd go for a flat white!

To achieve this, the we need to create a flow that contains the following elements:

* A trigger
* A script that chooses a coffee drink
* A flow variable to store the value chosen by the script
* An output that will display the value of the variable

### **Populate the script node**

We have added the script node, but it is still empty. Now we'll need to add the script that picks a random coffee drink to suggest to the user. Select the script node in your flow, open the 'Script Action' tab on the right sight of the screen and paste the following code into the script node:

// define a list with coffee drinks to suggest

def coffeesToSuggest = ['cappuccino','flat white','macchiato']

// randomly choose a number 0, 1 or 2

def number = new Random().nextInt(3)

// use the number to pick a coffee from the list

coffeeSuggestion = coffeesToSuggest[number]

Scripts in Teneo are written in [Groovy](http://groovy-lang.org/index.html). You can find more details about Groovy and scripts in Teneo in the [Scripting](https://www.teneo.ai/studio/scripting) section.

As you can see in the script node above the first two variables (coffeesToSuggest and number) are preceded by def which means they are [local variables](https://www.teneo.ai/studio/scripting/concepts/variables#local-variables) that are only available inside the script node. They will be forgotten immediately after the script is executed, in this case this will happen when Teneo moves on to the output node. However, we want the outcome of the script (the coffee chosen by the script node) to be available outside of the script so we can use it in our output text. So we will need to store the value in a flow variable that we can use later on.

The last line of the script assumes a variable called coffeeSuggestion exists in which the chosen coffee drink can be stored. This is determined by the fact that coffeeSuggestion is not defined by preceding it with def. However, we don't have a flow variable called coffeeSuggestion yet, so let's add it.