# Alexa 2: Fact-checking

Welcome to the second of six posts taking you from zero to hero on Alexa with Ruby! In our first module, we:

* Set up a simple Alexa skill
* Set up a tunnelled Sinatra application
* Hooked the two together to say "Hello World"

In this module, we'll handle **variable data** from users, using **Slots**. This module introduces:

* Slots
* Custom Slot Types

And this module uses:

* Sinatra
* Ruby's JSON library
* Ruby's HTTP library
* the [Numbers API](http://numbersapi.com/)

### Project Overview

We’re going to build a fact-checking mechanism so users can ask for facts about particular numbers. Here are some things users will be able to ask Alexa:

Alexa, ask Number Facts to tell me a trivia fact about 42 Alexa, ask Number Facts to tell me a math fact about 5

Users will be able to choose: - a number (any number!), and - a fact type.

Alexa will respond with an interesting fact about that number, specific to that type of fact.

## 1. Scaffolding a new skill

By the end of this section, we’ll be able to interact with a simplified version of our final feature. Users will be able to ask:

Alexa, ask Number Facts about 42

Alexa will respond with a trivia fact about the number **42**. Later, we’ll build on this feature to add trivia responses for all sorts of numbers.

Sign in to the [Alexa Developer Portal](https://developer.amazon.com/alexa), and set up a new skill. Let’s call it **Number Facts**, with an invocation name of **Number Facts**.

You could edit the skill you set up in module 1, or practice making a new one.

### Set up a simple skill

Let’s start by defining a minimal Intent Schema, with a single Intent:

{  
 "intents": [  
 {  
 "intent": "NumberFact"  
 }  
 ]  
}

Let's add a simple Utterance to invoke this Intent:

NumberFact about forty two

Before we go any further, let's check that our Utterance correctly invokes our Intent. We will set up up an HTTPS-tunnelled Sinatra application just as we did in module 1.

If you have a preferred way to provide an HTTPS endpoint to Alexa, by all means use that. We'll proceed under the assumption you're using ngrok with a locally-running Sinatra application.

### Setting up Sinatra with tunnelling

In our Sinatra application, we'll set up a single POST route, and respond with a minimal response:

require 'sinatra'  
require 'json'  
  
post '/' do   
 {   
 version: "1.0",  
 response: {  
 outputSpeech: {  
 type: "PlainText",  
 # Here's what Alexa will say  
 text: "The number 42 is pretty cool."  
 }  
 }  
 }.to\_json  
end

Great! Once we start the local server, start ngrok, and provide our ngrok HTTPS Endpoint to our Skill, we can use the Service Simulator to test a basic interaction. Or, if you have an Alexa device registered to your account, you can test the connection using any Alexa-enabled hardware.

Struggling to remember how to set up a local, tunnelled environment with Alexa? There's a guide in the first module blog post [here](https://developer.amazon.com/blogs/post/105df30e-9890-4a8c-9caf-5de1c8ff86cb/makers-academy-s-alexa-series-how-to-build-a-hello-world-skill-with-ruby).

### Connecting them together

When we ask:

Alexa, ask Number Facts about 42

Alexa responds with:

The number 42 is pretty cool.

We have a connection! Now, let’s upgrade our Ruby code to give us an interesting fact about the number 42.

### Connect to the Number Facts API

In our Ruby code, let’s make a call to the [Numbers API](http://numbersapi.com/) to grab an interesting fact about the number 42:

require 'sinatra'  
require 'json'  
  
# Since we're making a call to an external API,  
# we need Ruby's Net::HTTP library.  
require 'net/http'  
  
post '/' do   
 # First, we encode the Numbers API as a URI  
 number\_facts\_uri = URI("http://numbersapi.com/42")  
  
 # Then we make a request to the API  
 number\_fact = Net::HTTP.get(number\_facts\_uri)  
  
 {   
 version: "1.0",  
 response: {  
 outputSpeech: {  
 type: "PlainText",  
 # We provide the fact we received as  
 # plain text in our response to Amazon.  
 text: number\_fact  
 }  
 }  
 }.to\_json  
end

Head to the Service Simulator (or any Alexa device) and see that you can now make a request to Number Facts for random trivia about the number 42. This is a great start! But what about trivia for other numbers?

## 2. Using built-in Slots to pass parameters to our application

In section 1, we weren't able to pass any parameters to our application. As a result, we could only ask Number Facts to tell us about ’42’. If a user wanted to know about any other number, Alexa would just tell them about 42.

We’d like our users to receive trivia facts for all sorts of numbers. Users should be able to ask:

Alexa, ask Number Facts about {Number}

Where {Number} is any number. Our users should hear a response concerning that number, and that number only.

To do this, we need to provide a parameter to our Intent and Utterance. In Alexa’s terminology, we provide parameters as **Slots**. Slots need a **name** and a **type**.

### Using Built-In Slots to pass a Number

Let's start from the Utterance: what we want our users to be able to say. To reference a Slot in an Utterance, we use curly brackets ({}) around the name we will use for that Slot.

NumberFact about {Number}

Now let's add that Slot into our Intent Schema, with the same name we're referencing in our Utterance: Number.

{  
 "intents": [  
 {  
 "intent": "NumberFact",  
 "slots": [  
 {  
 "name": "Number",  
 "type": "AMAZON.NUMBER"  
 }  
 ]  
 }  
 ]  
}

* slots is an array of JSON objects, each with:
* a name: so we can reference the Slot in the Utterances
* a type, which limits the possible return values a Slot can have.

We’ve chosen the AMAZON.NUMBER type, which will convert any spoken numbers to digits:

Alexa, ask Number Facts about fifteen

Becomes

Alexa, ask Number Facts about 15

There are many [built-in Slot types](https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/built-in-intent-ref/slot-type-reference). You can also define custom slot types (we’ll do this in the next section).

Built-in Slot types are really handy for capturing user input for common use cases, such as numbers, dates, cities, and so on.

### Handling the Slot value in Sinatra

Before we change any Ruby code, let’s observe how adding a Slot modifies the incoming request from Amazon’s Service Simulator. We can log the incoming request in our Sinatra application using request.body.read within our post '/' route.

Using the Service Simulator, let's cause Amazon to send a request by using the phrase:

Alexa, ask Number Facts about 4

In our Sinatra application, notice the Slot value passed in the request:

"intent": {  
 "name": "NumberFact",   
 "slots": [   
 "Number": {   
 "name": "Number",   
 "value": "4"  
 }  
 }  
 ]  
}

At the moment, we're not doing anything other than logging the value for this slot: we're always returning information about the number 42. In our Sinatra application, let’s grab the Slot value out of Amazon’s request and pass that to the Numbers API:

require 'sinatra'  
require 'json'  
require 'net/http'  
  
post '/' do   
 # Grab the slot value from the incoming request  
 number = JSON.parse(request.body.read)["request"]["intent"]["slots"]["Number"]["value"]  
  
 # Pass that number to the numbers api  
 number\_facts\_uri = URI("http://numbersapi.com/#{ number }")  
 number\_fact = Net::HTTP.get(number\_facts\_uri)  
  
 {   
 version: "1.0",  
 response: {  
 outputSpeech: {  
 type: "PlainText",  
 # And respond with the new fact  
 text: number\_fact  
 }  
 }  
 }.to\_json  
end

Testing in the Service Simulator: users can now provide an arbitrary number to our Number Facts skill, and receive trivia about that number. Great!

Let’s make one final upgrade: asking for facts of a specific type.

## 3. Using custom slots

So far, users can ask Alexa:

Alexa, ask Number Facts about {Number}

Where {Number} is any number. Users will hear an interesting fact about any given number.

However, Numbers API can provide two different kinds of facts: trivia facts, and math facts. We’d like users to be able to ask:

Alexa, ask Number Facts to tell me a {FactType} fact about {Number}

{FactType} is either a ‘trivia’ fact or a ‘math’ fact.

### Using a Custom Slot to pass in a Fact Type

Because we’re passing another piece of variable information to our Intent, we’ll need to define another Slot.

Slots *do not* restrict user input to certain values. Instead, they guide interpretation of the user's words towards those terms.

However, where the built-in AMAZON.NUMBER slot restricted us to numbers only, there are no built-in Slots that will restrict us to the words ‘trivia’ or ‘math’. We have to make our own.

Let's head to the Interaction Model pane in the Alexa Skills Developer Console, and add a new Custom Slot type. We'll call this Custom Slot type FACT\_TYPE. There are two possible Values for this Custom Slot type: trivia and math. They must be provided separated by a newline, like so:

trivia  
math

These values act as training data for Alexa's voice recognition. They don't restrict users to just the given words: users can use different words in addition to these two. For instance, if a user said "Alexa, ask Number Facts to tell me a bicycle fact about 42", the word 'bicycle' would be sent as part of the request.

Now we've defined our Custom Slot type, we can go ahead and rewrite our Utterance to include the Slot:

NumberFact tell me a {FactType} fact about {Number}

And provide the Slot to our Intent:

{  
 "intents": [  
 {  
 "intent": "NumberFact",  
 "slots": [  
 {   
 "name": "Number",  
 "type": "AMAZON.NUMBER"  
 },  
 {  
 "name": "FactType",  
 "type": "FACT\_TYPE"  
 }  
 ]  
 }  
 ]  
}

Remember to hit 'Save' in the developer portal to update the interaction model.

### Handling the Custom Slot Value in Sinatra

Now that we're passing a fact type through to our Sinatra Application, we can grab the fact type similarly to how we did the number. Once we have it, we can pass it directly to the Numbers API:

require 'sinatra'  
require 'json'  
require 'net/http'  
  
post '/' do   
 parsed\_request = JSON.parse(request.body.read)  
  
 number = parsed\_request["request"]["intent"]["slots"]["Number"]["value"]  
 fact\_type = parsed\_request["request"]["intent"]["slots"]["FactType"]["value"]  
  
 number\_facts\_uri = URI("http://numbersapi.com/#{ number }/#{ fact\_type }")  
 number\_fact = Net::HTTP.get(number\_facts\_uri)  
  
 {   
 version: "1.0",  
 response: {  
 outputSpeech: {  
 type: "PlainText",  
 text: number\_fact  
 }  
 }  
 }.to\_json  
end

Let’s test in the Service Simulator, or in any Alexa-enabled device:

Alexa, ask Number Facts to tell me a math fact about 17

If we've hooked everything together, we should hear an interesting piece of knowledge about the mathematics of the number 17.

Join us next module to learn how to handle contextual requests and conversation, using the Session!

### Extra Credits

EXTRA CREDIT: Notice how entering an undefined phrase, e.g. "Alexa, ask Number Facts about 12 math", still invokes the Intent. As a Ruby exercise, try to handle users' poorly-formed phrases gracefully via the Sinatra application.

EXTRA CREDIT: Remember that Slot Values do not list the *possible* values a user can enter, only the *valid* values your application accepts. Since users can say whatever they like in addition to "trivia" and "math", upgrade your Sinatra application to handle cases where a user asks for a Fact Type you don't recognise.