Fun Number Facts - Rest API Example

Rishab Gaddam

Introduction

In this skill, we are going to make use of a free RESTful API from http://numbersapi.com/. At this website it shows all the endpoints of the API.

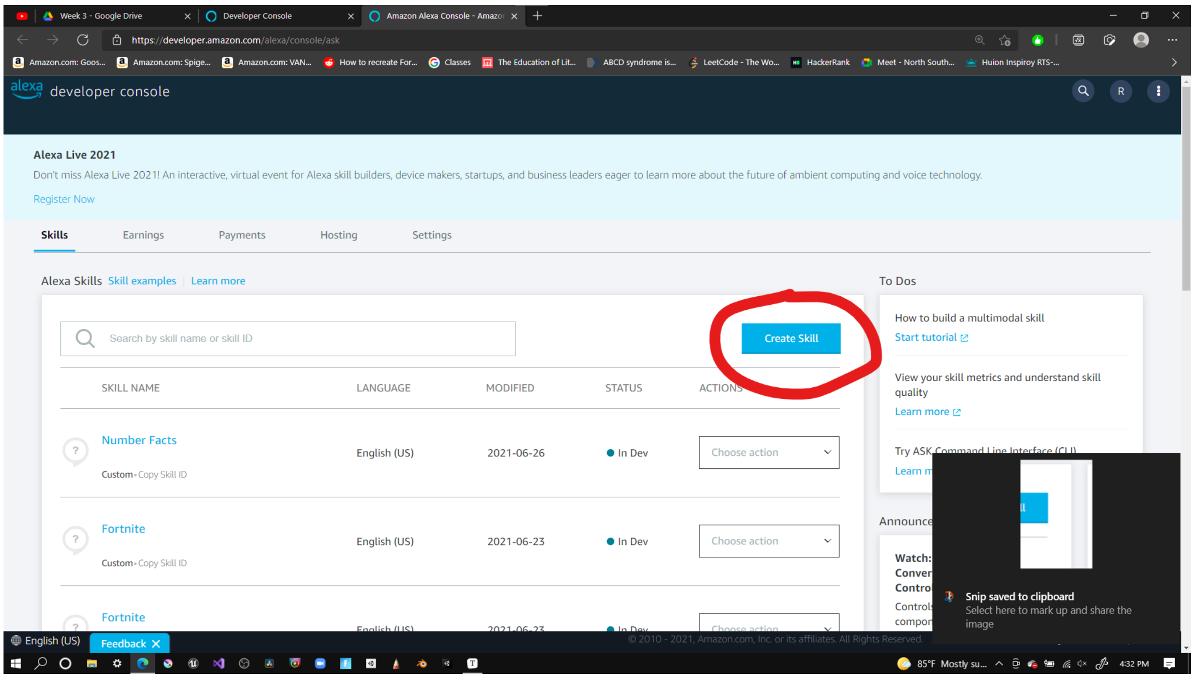
API Endpoints:

- http://numbersapi.com/random/trivia
- http://numbersapi.com/random/year
- http://numbersapi.com/random/date
- http://numbersapi.com/random/math
- http://numbersapi.com/{number}/trivia
- http://numbersapi.com/{number}/year
- http://numbersapi.com/{number}/date
- http://numbersapi.com/{number}/math

Creating 'Number Facts' skill

Begin by navigating to on https://developer.amazon.com/alexa/console/ask.

Click on **Create Skill**



Alexa Live 2021
Don't miss Alexa Live 2021! An interactive, virtual event for Alexa skill builders, device makers, startups, and business leaders eager to learn more about the future of ambient computing and voice technology.
Register Now

SKILL NAME	LANGUAGE	MODIFIED	STATUS	ACTIONS
Number Facts Custom · Copy Skill ID	English (US)	2021-06-26	● In Dev	Choose action
Fortnite Custom · Copy Skill ID	English (US)	2021-06-23	● In Dev	Choose action
Fortnite	English (US)	2021-06-23	● In Dev	Choose action

How to build a multimodal skill
Start tutorial

View your skill metrics and understand skill quality
Learn more

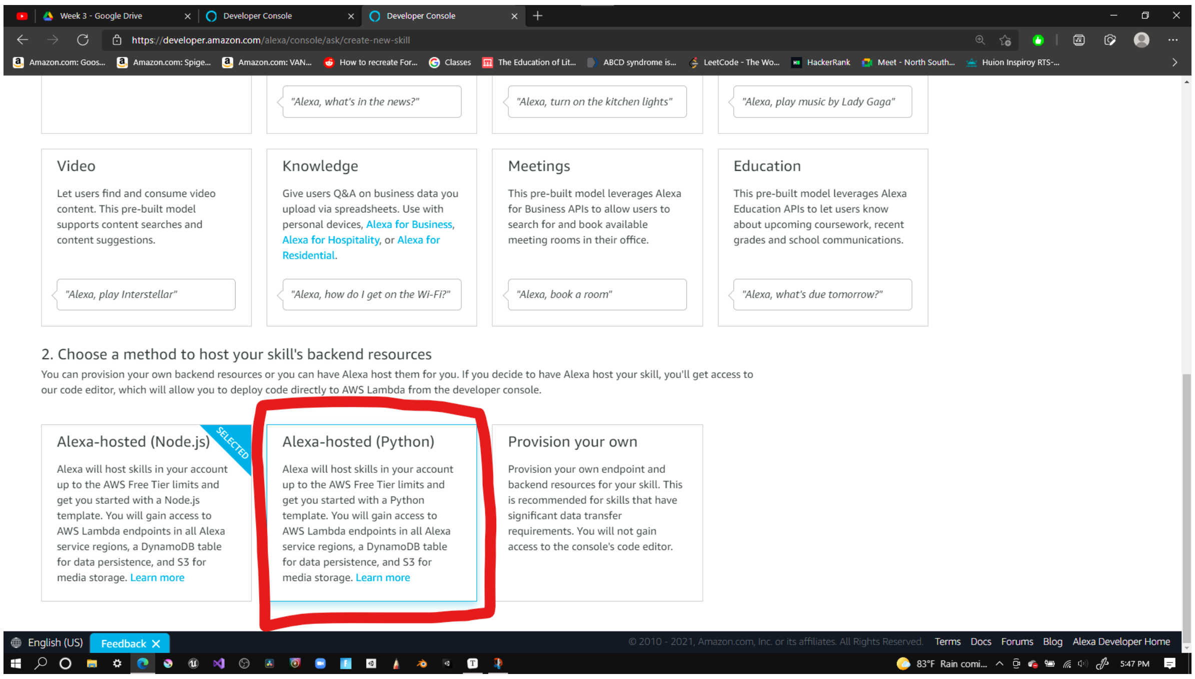
Try ASK Command Line Interface (CLI)

Select here to mark up and share the image

© 2010 - 2021, Amazon.com, Inc. or its affiliates. All Rights Reserved.

Name it **Number Facts** (or anything else that you want to name it)

Choose **Python** for language



"Alexa, what's in the news?"

"Alexa, turn on the kitchen lights"

"Alexa, play music by Lady Gaga"

Video

Let users find and consume video content. This pre-built model supports content searches and content suggestions.

"Alexa, play Interstellar"

Knowledge

Give users Q&A on business data you upload via spreadsheets. Use with personal devices, Alexa for Business, Alexa for Hospitality, or Alexa for Residential.

"Alexa, how do I get on the Wi-Fi?"

Meetings

This pre-built model leverages Alexa for Business APIs to allow users to search for and book available meeting rooms in their office.

"Alexa, book a room"

Education

This pre-built model leverages Alexa Education APIs to let users know about upcoming coursework, recent grades and school communications.

"Alexa, what's due tomorrow?"

2. Choose a method to host your skill's backend resources

You can provision your own backend resources or you can have Alexa host them for you. If you decide to have Alexa host your skill, you'll get access to our code editor, which will allow you to deploy code directly to AWS Lambda from the developer console.

Alexa-hosted (Node.js) SELECTED

Alexa will host skills in your account up to the AWS Free Tier limits and get you started with a Node.js template. You will gain access to AWS Lambda endpoints in all Alexa service regions, a DynamoDB table for data persistence, and S3 for media storage. Learn more

Alexa-hosted (Python)

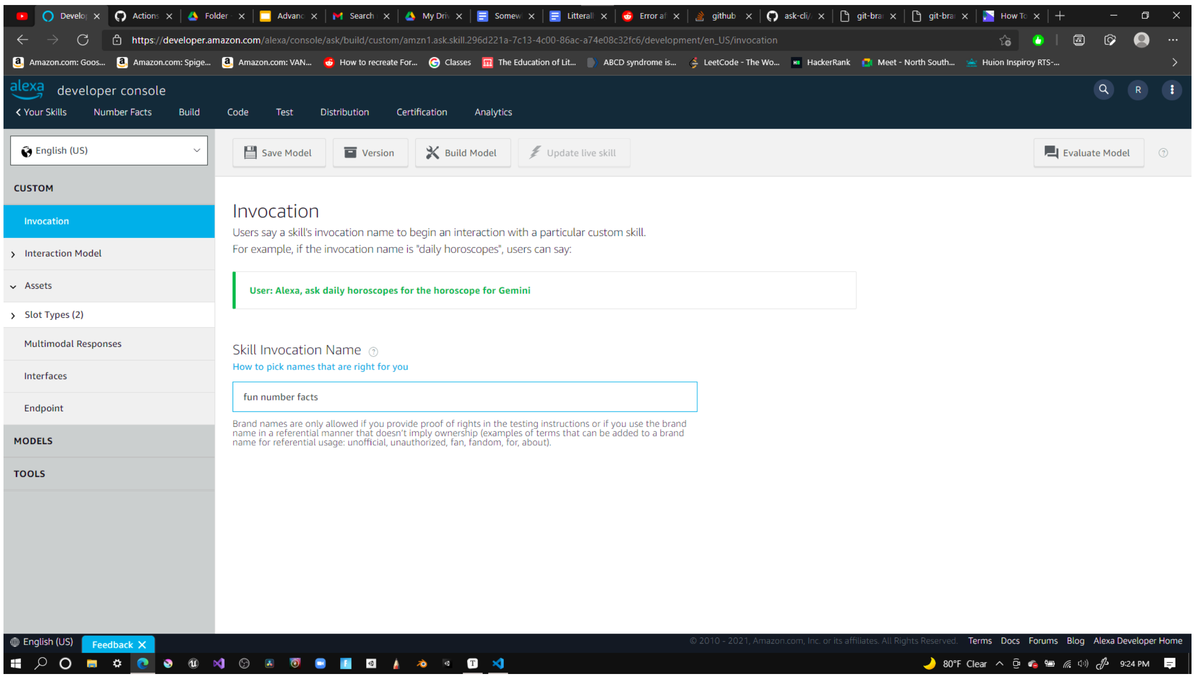
Alexa will host skills in your account up to the AWS Free Tier limits and get you started with a Python template. You will gain access to AWS Lambda endpoints in all Alexa service regions, a DynamoDB table for data persistence, and S3 for media storage. Learn more

Provision your own

Provision your own endpoint and backend resources for your skill. This is recommended for skills that have significant data transfer requirements. You will not gain access to the console's code editor.

Invocation Name

The first thing to do is to change the invocation name to "**fun number facts**"



Invocation

Users say a skill's invocation name to begin an interaction with a particular custom skill.
For example, if the invocation name is "daily horoscopes", users can say:

User: Alexa, ask daily horoscopes for the horoscope for Gemini

Skill Invocation Name

How to pick names that are right for you

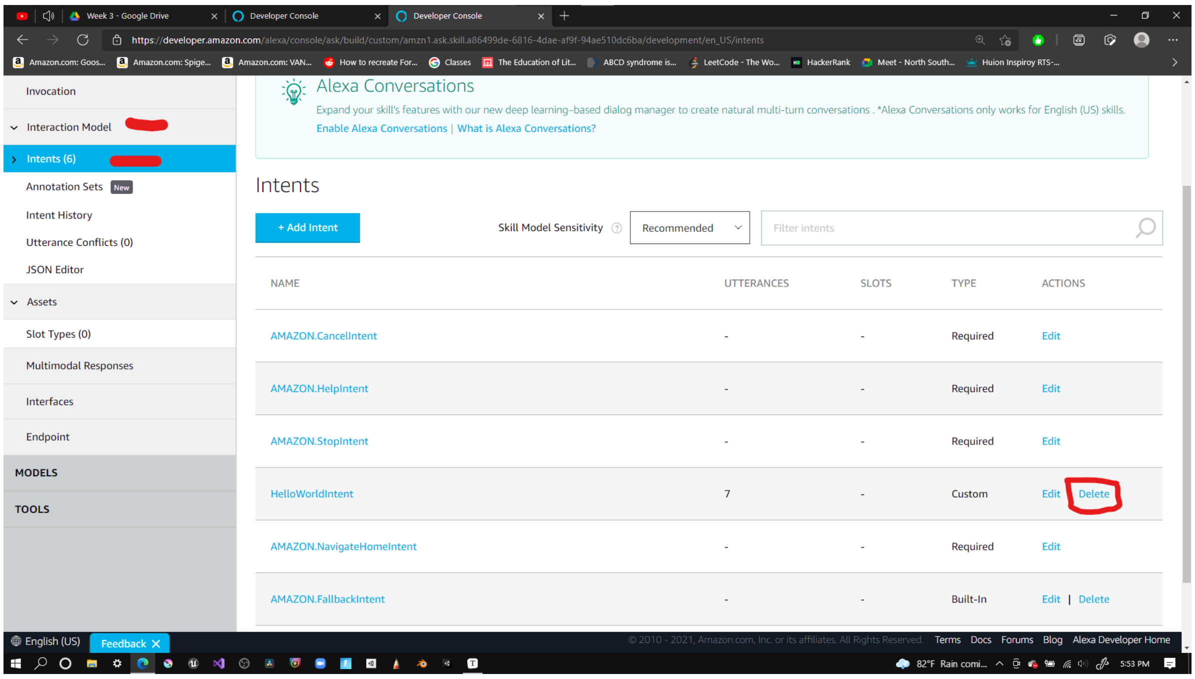
fun number facts

Brand names are only allowed if you provide proof of rights in the testing instructions or if you use the brand name in a referential manner that doesn't imply ownership (examples of terms that can be added to a brand name for referential usage: unofficial, unauthorized, fan, fandom, for, about).

Intents

In the build section, expand the Interaction Model tab on the side-bar. Go to **Intents**.

The first step is to remove `HelloWorldIntent`



Alexa Conversations

Expand your skill's features with our new deep learning-based dialog manager to create natural multi-turn conversations . *Alexa Conversations only works for English (US) skills.
Enable Alexa Conversations | What is Alexa Conversations?

Intents

+ Add Intent   Skill Model Sensitivity: Recommended   Filter intents

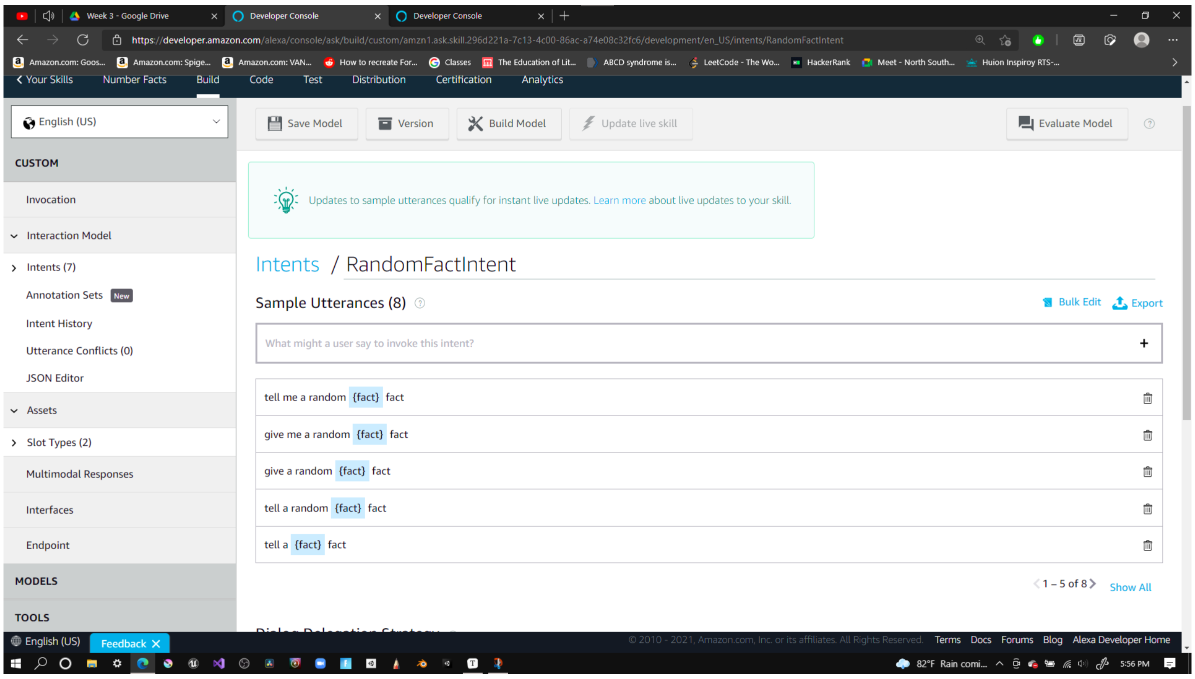
NAME	UTTERANCES	SLOTS	TYPE	ACTIONS
AMAZON.CancelIntent	-	-	Required	Edit
AMAZON.HelpIntent	-	-	Required	Edit
AMAZON.StopIntent	-	-	Required	Edit
HelloWorldIntent	7	-	Custom	Edit Delete
AMAZON.NavigateHomeIntent	-	-	Required	Edit
AMAZON.FallbackIntent	-	-	Built-In	Edit | Delete

RandomFactIntent

Next **Add an Intent** called "RandomFactIntent"

This Intent will take care of random numbers. This is when the user does not have a specific number to get a fact for and they want Alexa to get a fact for a random number for them.

Add some **Sample Utterances**



Updates to sample utterances qualify for instant live updates. Learn more about live updates to your skill.

Intents / RandomFactIntent

Sample Utterances (8)   Bulk Edit   Export

What might a user say to invoke this intent?

tell me a random {fact} fact
give me a random {fact} fact
give a random {fact} fact
tell a random {fact} fact
tell a {fact} fact

1 – 5 of 8   Show All

Add as many as you think you need.

Slots:

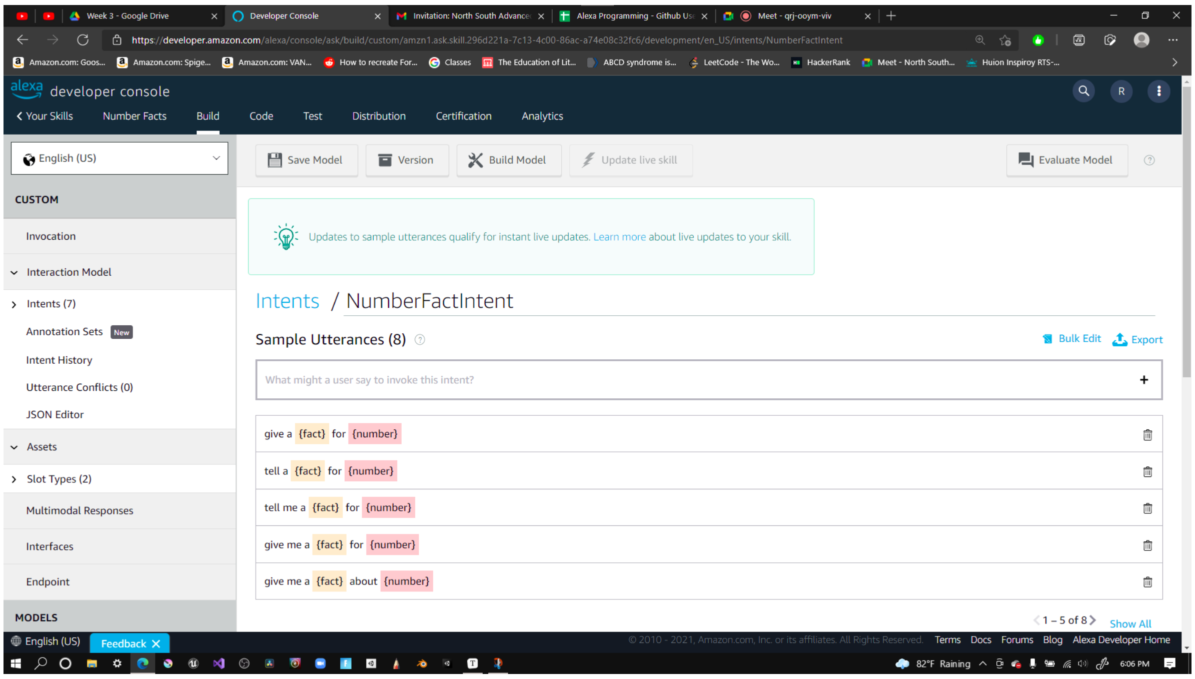
You need a **"fact"** slot. We will assign the slot type later. This holds the specific types of fact which are: trivia, year, date, and math.

NumberFactIntent

Add another Intent called "NumberFactIntent"

This intent will be used when the user has a specific number that they need a fact for. So, this will need another slot.

Add some **Sample Utterances**



Updates to sample utterances qualify for instant live updates. Learn more about live updates to your skill.

Intents / NumberFactIntent

Sample Utterances (8)

What might a user say to invoke this intent?

give a {fact} for {number}
tell a {fact} for {number}
tell me a {fact} for {number}
give me a {fact} for {number}
give me a {fact} about {number}

1 – 5 of 8

Slots:

You need a "**fact**" slot. We will assign the slot type later. This holds the specific types of facts which are: trivia, year, date, and math.

You also need a "**number**" slot. Assign the slot type as **AMAZON.NUMBER**. This is the slot which will hold the number to get the fact for.

These are all the Intents we will need!

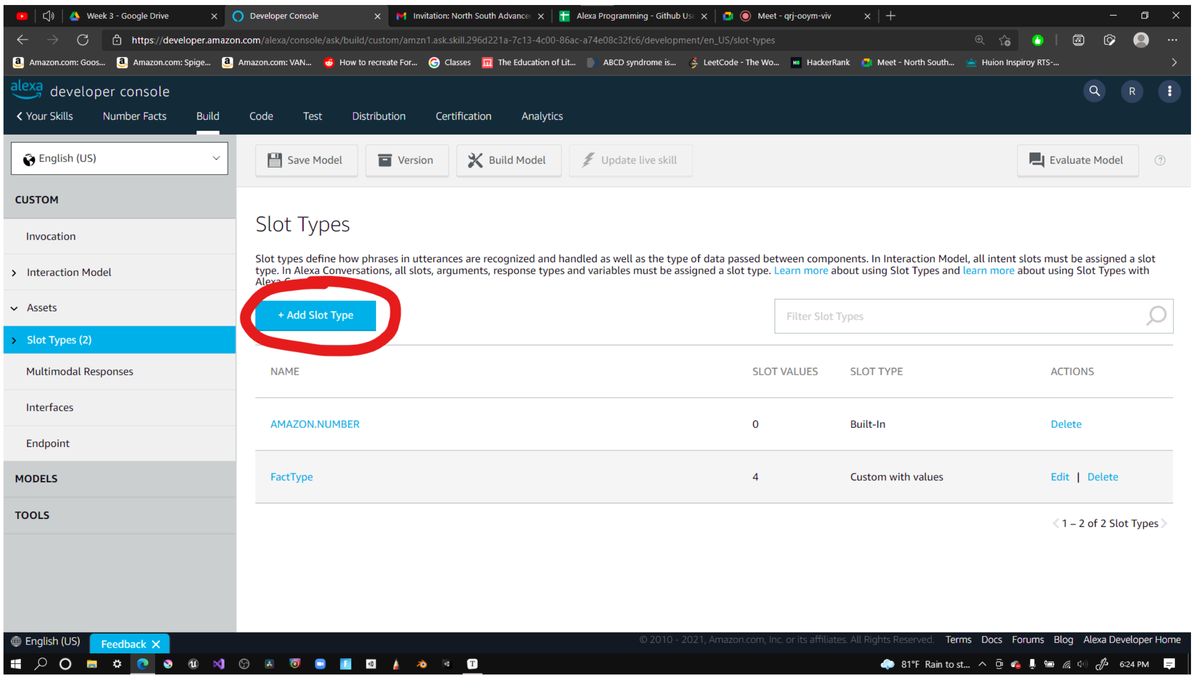
Slot Types

Moving on to the slot types!

We only need one slot type for this skill. This slot type will hold the fact type that the user wants.

FactType

Create a new **Slot Type** called "**FactType**"



Slot Types

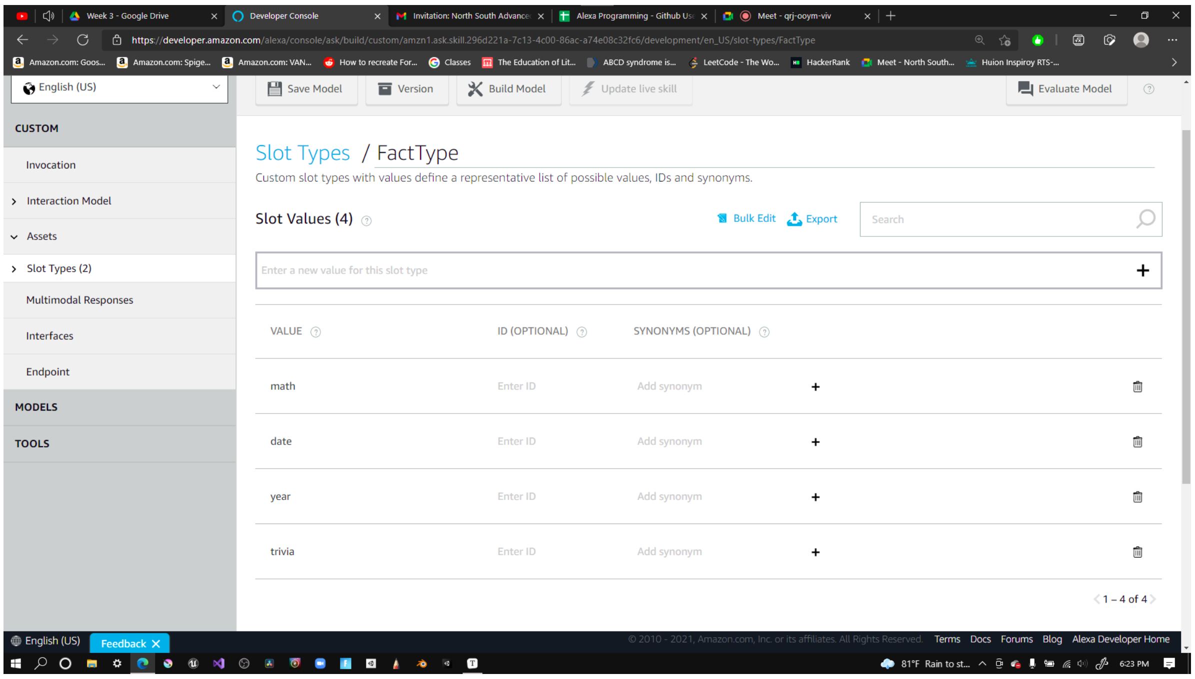
Slot types define how phrases in utterances are recognized and handled as well as the type of data passed between components. In Interaction Model, all intent slots must be assigned a slot type. In Alexa Conversations, all slots, arguments, response types and variables must be assigned a slot type. Learn more about using Slot Types and learn more about using Slot Types with Alexa C

NAME	SLOT VALUES	SLOT TYPE	ACTIONS
AMAZON.NUMBER	0	Built-In	Delete
FactType	4	Custom with values	Edit | Delete

1 – 2 of 2 Slot Types

Then create **Slot Values** of trivia, year, date, and math

This is what it should look like:



Slot Types / FactType

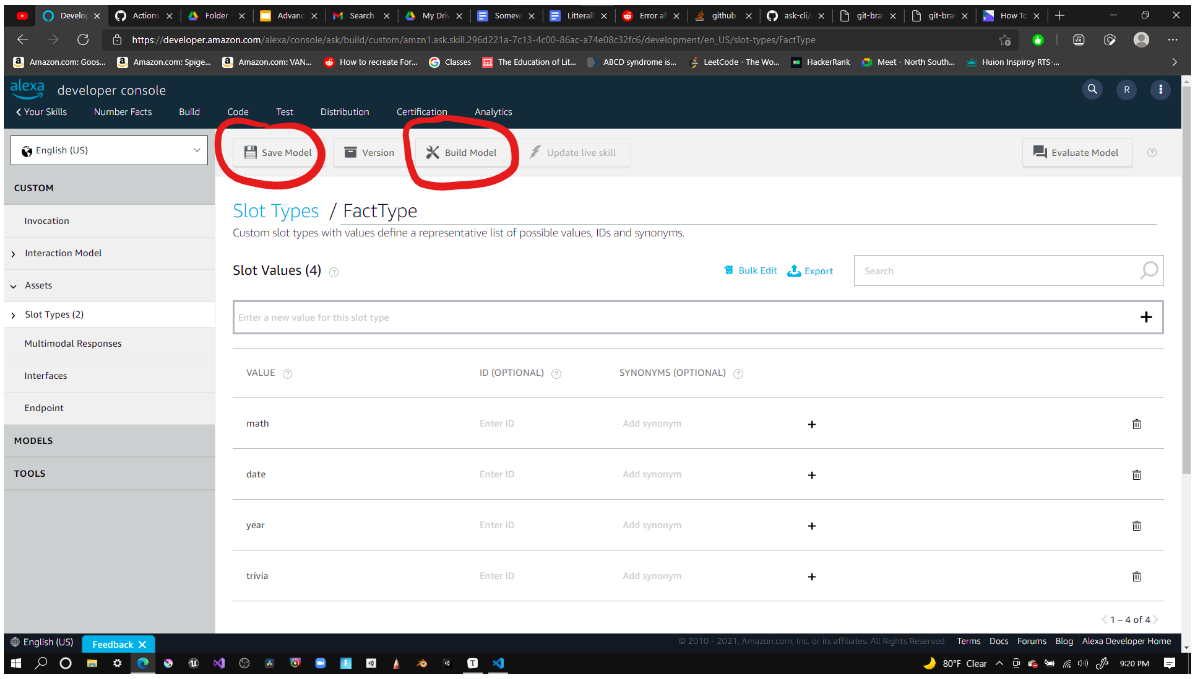
Custom slot types with values define a representative list of possible values, IDs and synonyms.

Slot Values (4)

VALUE	ID (OPTIONAL)	SYNONYMS (OPTIONAL)
math	Enter ID	Add synonym
date	Enter ID	Add synonym
year	Enter ID	Add synonym
trivia	Enter ID	Add synonym

1 – 4 of 4

Make sure to save and build!!!



Slot Types / FactType

Custom slot types with values define a representative list of possible values, IDs and synonyms.

Slot Values (4)

Enter a new value for this slot type

VALUE	ID (OPTIONAL)	SYNONYMS (OPTIONAL)
math	Enter ID	Add synonym
date	Enter ID	Add synonym
year	Enter ID	Add synonym
trivia	Enter ID	Add synonym

1 – 4 of 4

Code

Time to start doing the fun stuff!

Add requests module to `requirements.txt`

Add the following line to your requirements.txt:

```
requests==2.25.1
```

Tip: Make sure to save

Import requests module

As we talked about in the slides we have to import the requests module to get the api's messages.

```
import requests
```

Change the speak_output of the `LaunchRequestHandler`

We need to update the speak output of the launch handler to match with our skill.

```
speak_output = "Welcome to number facts, you can ask for a trivia, year, date, or math fact of a random or specific number."
```

Done!

This is what the `LaunchRequestHandler` should look like:

```
class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)
    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Welcome to number facts, you can ask for a trivia, year, date, or math fact of a random or specific number."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )
```

Create RandomFactIntentHandler Class

We will need a new intent handler called "**RandomFactIntentHandler**". This handler will take care of the **RandomFactIntent**.

*Tip: Copy and Paste the **HelloWorldIntent** and rename it. It is not illegal (even though it may feel like it).*

```
class RandomFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Hello, World"

        return (
            handler_input.response_builder
                .speak(speak_output)
                # .ask("add a reprompt if you want to keep the session open for the user to respond")
                .response
        )
```

Now, we need to get the fact slot (*like we did in the previous math memer example*).

Note: This code is editing the previous code. So, replace this handle function with the previous one.

```
def handle(self, handler_input):
    #type: (HandlerInput) -> Response
    slots = handler_input.request_envelope.request.intent.slots
    fact = slots['fact'].value
```

The next step is to actually use the Number API to get a fun fact. To do this, we are going to use the requests module.

We do it like this:

```
def handle(self, handler_input):
    #type: (HandlerInput) -> Response
    slots = handler_input.request_envelope.request.intent.slots
    fact = slots['fact'].value

    speak_output = requests.get(f"http://numbersapi.com/random/{fact}").text
```

Then, we just make Alexa speak the **speak_output**. Easy.

```
def handle(self, handler_input):
    # type: (HandlerInput) -> Response
    slots = handler_input.request_envelope.request.intent.slots
    fact = slots['fact'].value

    speak_output = requests.get(f"http://numbersapi.com/random/{fact}").text

    return (
        handler_input.response_builder
        .speak(speak_output)
        # .ask("add a reprompt if you want to keep the session open for the user to respond")
        .response
    )
```

We are now done with this class!

This is the final class:

```
class RandomFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        slots = handler_input.request_envelope.request.intent.slots
        fact = slots['fact'].value

        speak_output = requests.get(f"http://numbersapi.com/random/{fact}").text

        return (
            handler_input.response_builder
            .speak(speak_output)
            # .ask("add a reprompt if you want to keep the session open for the user to respond")
            .response
        )
```

Create NumberFactIntentHandler Class

We will need a new intent handler called "**NumberFactIntentHandler**". This handler will handle the **NumberFactIntent**. You may want to copy and paste this from the **RandomFactIntentHandler**.

```python
class NumberFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        slots = handler_input.request_envelope.request.intent.slots
        fact = slots['fact'].value

        speak_output = requests.get(f"http://numbersapi.com/random/{fact}").text

        return (
            handler_input.response_builder
            .speak(speak_output)
            # .ask("add a reprompt if you want to keep the session open for the user to respond")
            .response
        )
```

Now, we need another slot for this intent. If you remember we created a number slot for this intent. Getting it is as easy as this:

```python
num = slots['number'].value
```

Now, we feed the number to the API:

```python
speak_output = reqeusts.get(f"http://numbersapi.com/{num}/{fact}").text
```

Now, we are done with this Handler!

Our finished class looks like:

```python
class NumberFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        slots = handler_input.request_envelope.request.intent.slots
        fact = slots['fact'].value
        num = slots['number'].value

        speak_output = requests.get(f"http://numbersapi.com/{number}/{fact}").text

        return (
            handler_input.response_builder
            .speak(speak_output)
            # .ask("add a reprompt if you want to keep the session open for the user to respond")
            .response
        )
```

Deleting out old Handler class and Adding our new Handler classes to the `SkillBuilder ( )`

First off, Removing the old HelloWorldIntentHandler. Delete the `HelloWorldIntentHandler()` class and then, scroll all the way to the bottom and look for `sb.add_request_handler(HelloWorldIntentHandler())`, then, delete it.

Next, adding our two new classes to the SB. Easy enough, add these lines to after the **HelloWorldIntentHandler** that you just deleted:

```
sb.add_request_handler(RandomFactIntentHandler())
sb.add_request_handler(NumberFactIntentHandler())
```

We are now done!

Our complete code:

```
# -*- coding: utf-8 -*-

# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.
# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,
# session persistence, api calls, and more.
# This sample is built using the handler classes approach in skill builder.
import logging
import ask_sdk_core.utils as ask_utils

from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput

import requests

from ask_sdk_model import Response

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)



class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Welcome to number facts, you can ask for a trivia, year, date, or math fact of a random or specific
number."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
            .response
        )


class RandomFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        slots = handler_input.request_envelope.request.intent.slots
        fact = slots['fact'].value

        speak_output = requests.get(f"http://numbersapi.com/random/{fact}").text

        return (
            handler_input.response_builder
                .speak(speak_output)
                # .ask("add a reprompt if you want to keep the session open for the user to respond")
                .response
        )


class NumberFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("NumberFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        slots = handler_input.request_envelope.request.intent.slots
        fact = slots['fact'].value
        num = slots['number'].value

        speak_output = requests.get(f"http://numbersapi.com/{num}/{fact}").text
        # speak_output = num

        return (
            handler_input.response_builder
                .speak(speak_output)
                # .ask("add a reprompt if you want to keep the session open for the user to respond")
                .response
        )


class HelpIntentHandler(AbstractRequestHandler):
    """Handler for Help Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("AMAZON.HelpIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "You can say hello to me! How can I help?"
        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )


class CancelOrStopIntentHandler(AbstractRequestHandler):
    """Single handler for Cancel and Stop Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return (ask_utils.is_intent_name("AMAZON.CancelIntent")(handler_input) or
                ask_utils.is_intent_name("AMAZON.StopIntent")(handler_input))

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Goodbye!"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .response
        )

class FallbackIntentHandler(AbstractRequestHandler):
    """Single handler for Fallback Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("AMAZON.FallbackIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        logger.info("In FallbackIntentHandler")
        speech = "Hmm, I'm not sure. You can say Hello or Help. What would you like to do?"
        reprompt = "I didn't catch that. What can I help you with?"

        return handler_input.response_builder.speak(speech).ask(reprompt).response

class SessionEndedRequestHandler(AbstractRequestHandler):
    """Handler for Session End."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_request_type("SessionEndedRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        # Any cleanup logic goes here.

        return handler_input.response_builder.response


class IntentReflectorHandler(AbstractRequestHandler):
    """The intent reflector is used for interaction model testing and debugging.
    It will simply repeat the intent the user said. You can create custom handlers
    for your intents by defining them above, then also adding them to the request
    handler chain below.
    """
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_request_type("IntentRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        intent_name = ask_utils.get_intent_name(handler_input)
        speak_output = "You just triggered " + intent_name + "."

        return (
            handler_input.response_builder
                .speak(speak_output)
                # .ask("add a reprompt if you want to keep the session open for the user to respond")
                .response
        )


class CatchAllExceptionHandler(AbstractExceptionHandler):
    """Generic error handling to capture any syntax or routing errors. If you receive an error
    stating the request handler chain is not found, you have not implemented a handler for
    the intent being invoked or included it in the skill builder below.
    """
    def can_handle(self, handler_input, exception):
        # type: (HandlerInput, Exception) -> bool
        return True

    def handle(self, handler_input, exception):
        # type: (HandlerInput, Exception) -> Response
        logger.error(exception, exc_info=True)

        speak_output = "Sorry, I had trouble doing what you asked. Please try again."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )

# The SkillBuilder object acts as the entry point for your skill, routing all request and response
# payloads to the handlers above. Make sure any new handlers or interceptors you've
# defined are included below. The order matters - they're processed top to bottom.


sb = SkillBuilder()

sb.add_request_handler(LaunchRequestHandler())
sb.add_request_handler(RandomFactIntentHandler())
sb.add_request_handler(NumberFactIntentHandler())
sb.add_request_handler(HelpIntentHandler())
sb.add_request_handler(CancelOrStopIntentHandler())
sb.add_request_handler(FallbackIntentHandler())
sb.add_request_handler(SessionEndedRequestHandler())
sb.add_request_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doesn't override your
custom intent handlers

sb.add_exception_handler(CatchAllExceptionHandler())
```

```
lambda_handler = sb.lambda_handler()
```

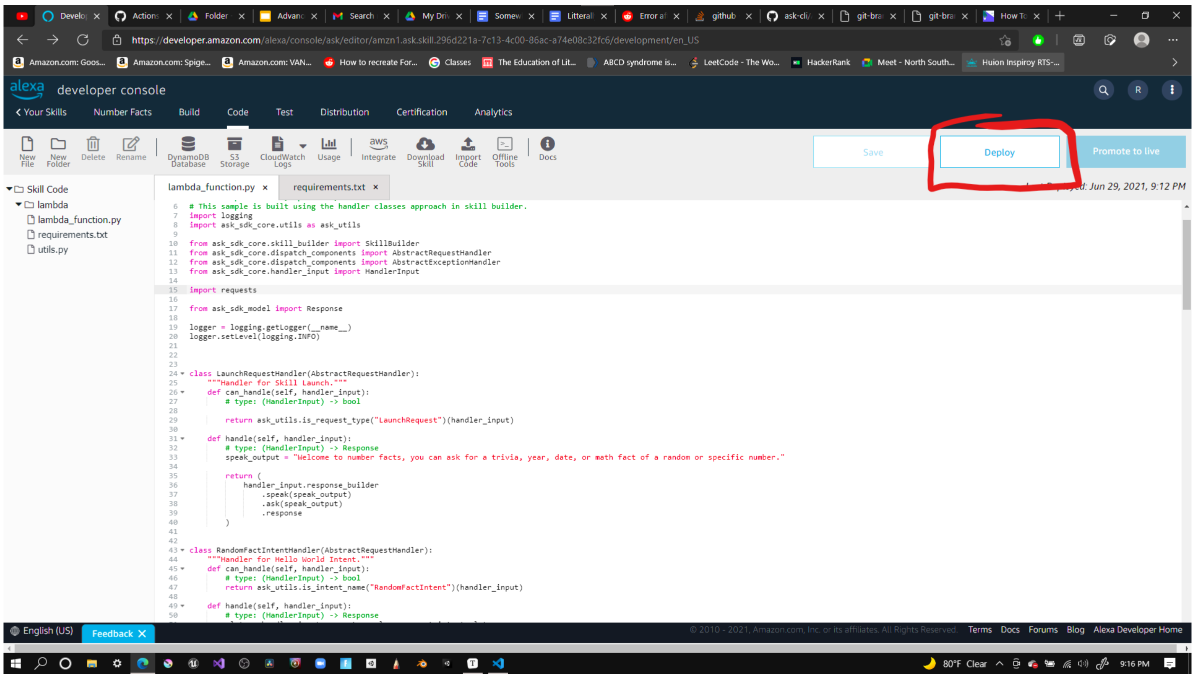
Tip: Make sure to save!

Testing

Tip: Before you start testing, make sure to save all the files that you changed!

Deploy

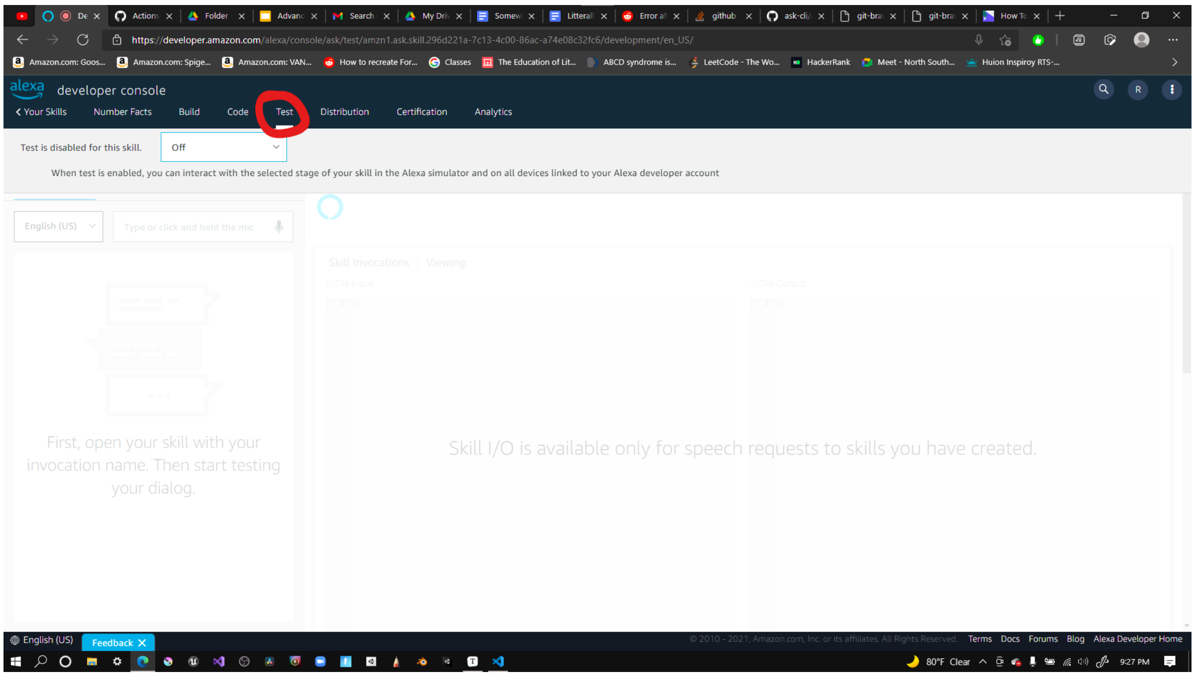
The first thing to do before going to the test tab is to deploy the code that you saved.



```
# This sample is built using the handler classes approach in skill builder.
import logging
import ask_sdk_core.utils as ask_utils

from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput

import requests

from ask_sdk_model import Response

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)



class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Welcome to number facts, you can ask for a trivia, year, date, or math fact of a random or specific number."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )


class RandomFactIntentHandler(AbstractRequestHandler):
    """Handler for Hello World Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("RandomFactIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
```

Testing

First go to the **Test** tab.



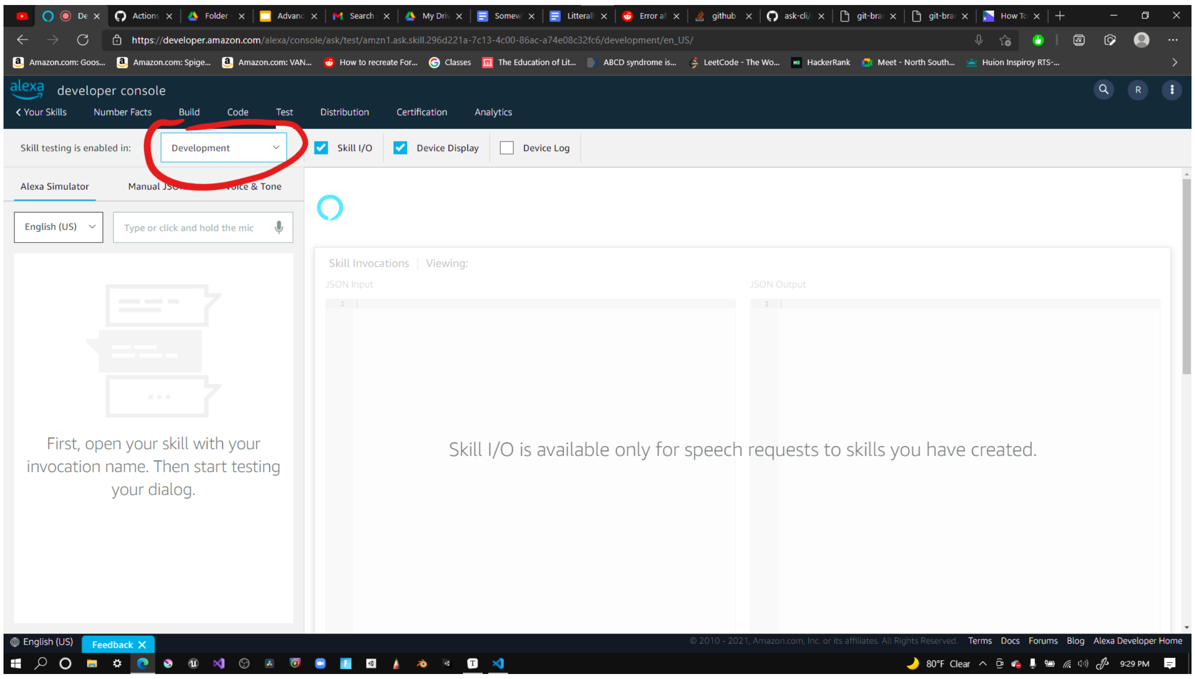
Test is disabled for this skill. Off

When test is enabled, you can interact with the selected stage of your skill in the Alexa simulator and on all devices linked to your Alexa developer account

First, open your skill with your invocation name. Then start testing your dialog.

Skill I/O is available only for speech requests to skills you have created.

Then, enable the testing to "**Development**".



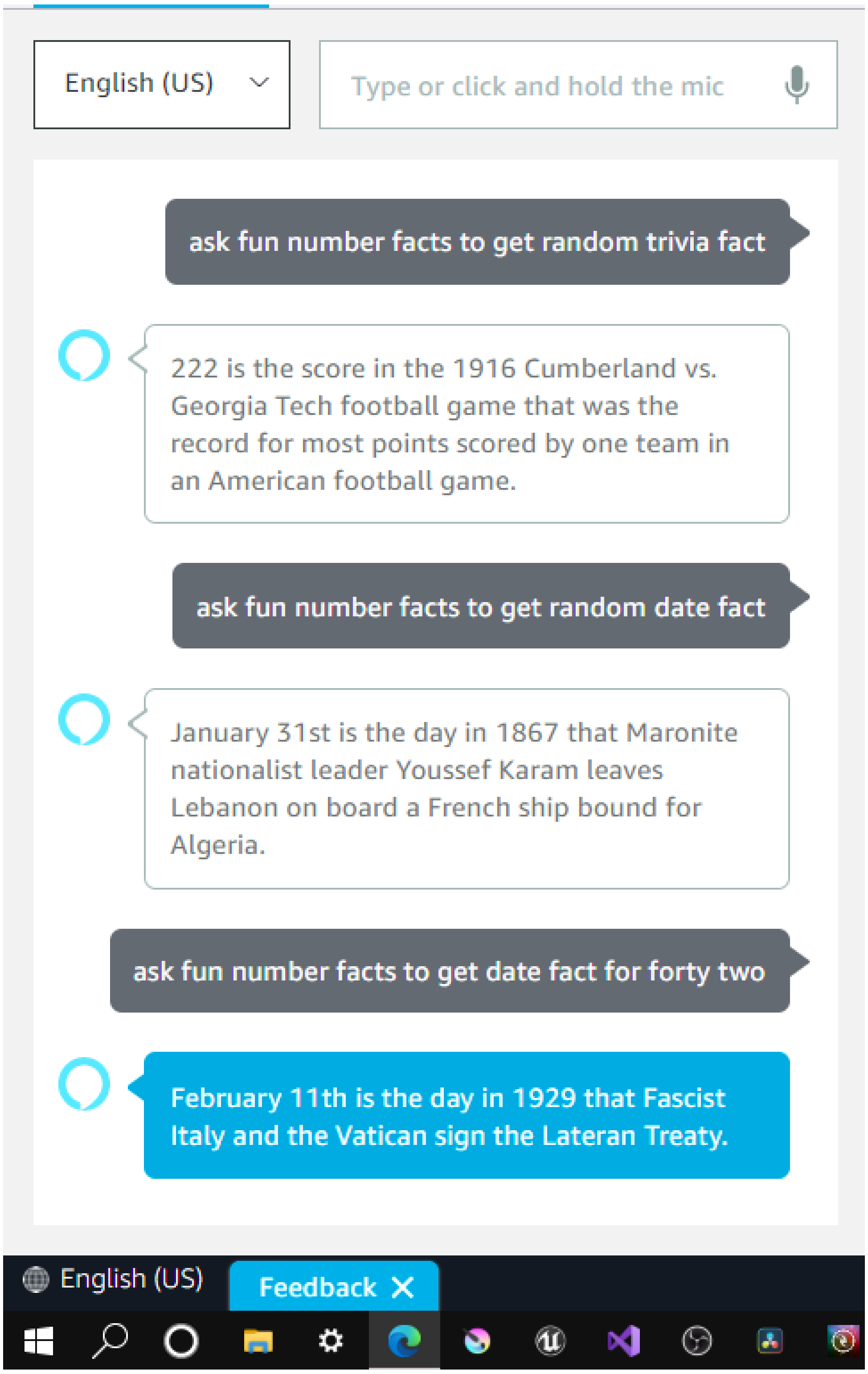
Skill testing is enabled in: Development

Skill I/O ✓ Device Display ✓ Device Log

First, open your skill with your invocation name. Then start testing your dialog.

Skill I/O is available only for speech requests to skills you have created.

This is what a test run of mine looked like:



English (US)

Type or click and hold the mic

ask fun number facts to get random trivia fact

222 is the score in the 1916 Cumberland vs. Georgia Tech football game that was the record for most points scored by one team in an American football game.

ask fun number facts to get random date fact

January 31st is the day in 1867 that Maronite nationalist leader Youssef Karam leaves Lebanon on board a French ship bound for Algeria.

ask fun number facts to get date fact for forty two

February 11th is the day in 1929 that Fascist Italy and the Vatican sign the Lateran Treaty.

Congratulations! You have managed to create an Alexa skill that spews out facts about numbers.