

Building User Interfaces

Dialogflow 2

Intermediate Concepts

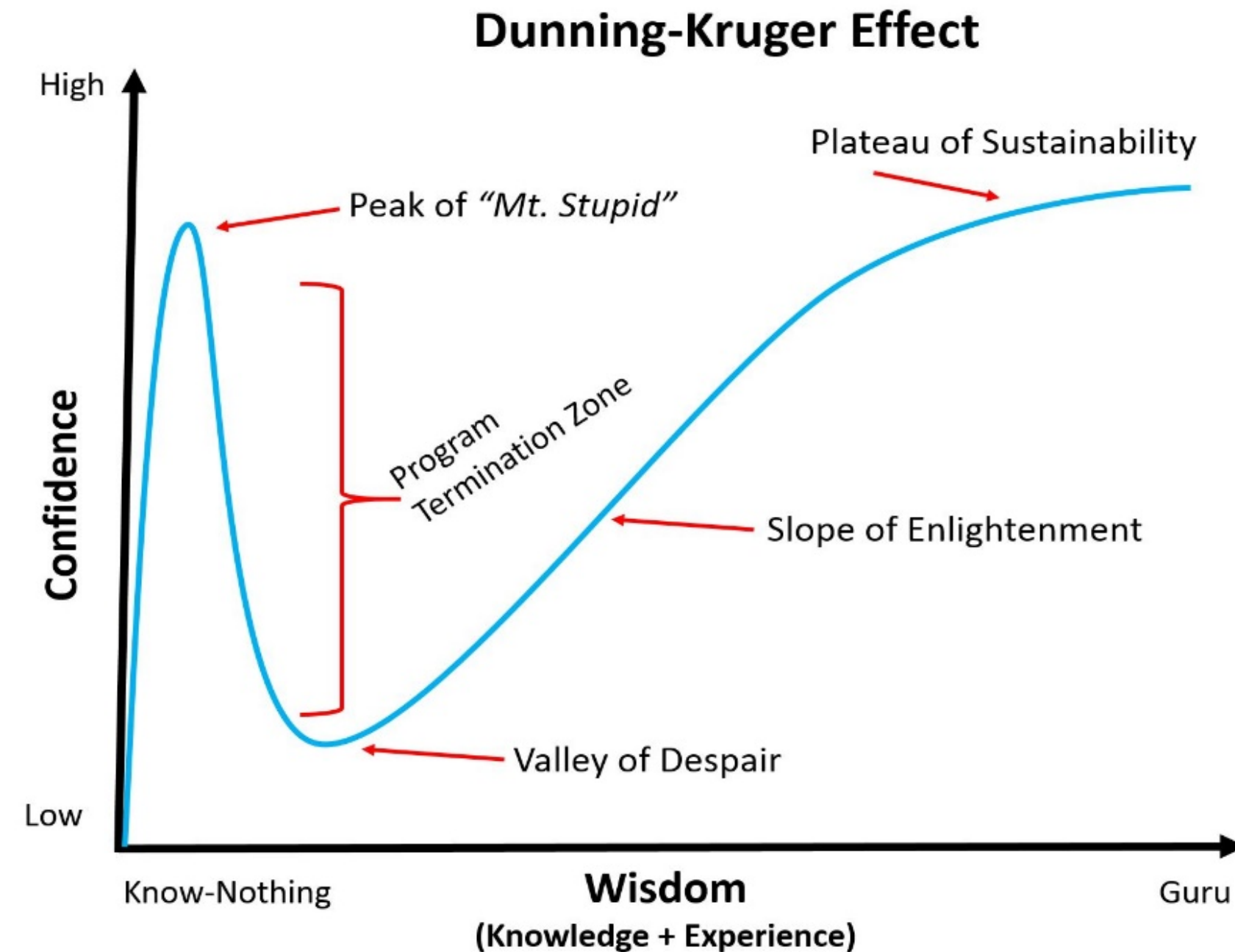
Professor Bilge Mutlu

Announcements

- **This week:** midterm Thursday/Friday
- **Next week:** last week of lectures; lecture on Monday + bonus lecture on Wednesday
- **End of semester:**
 - Extra credit quiz after Thanksgiving
 - All assignments are due on December 8
 - We will offer office hours until December 8

A bit of a pep talk¹

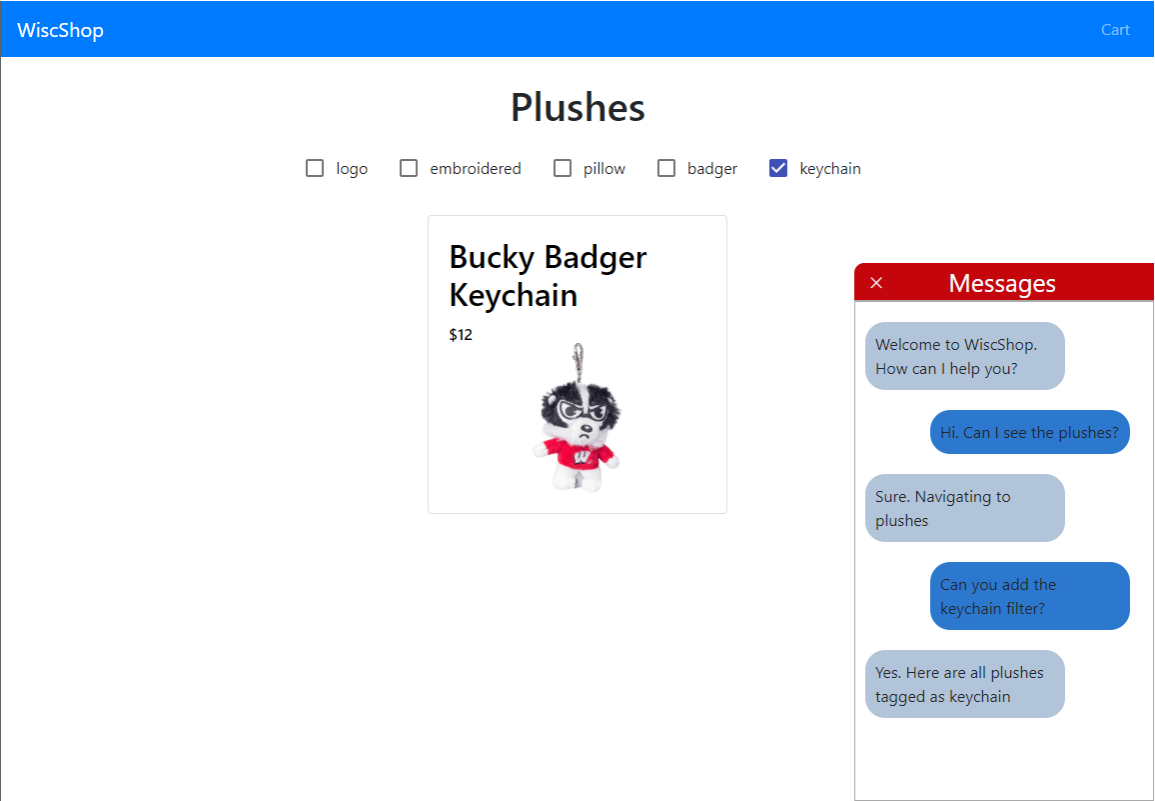
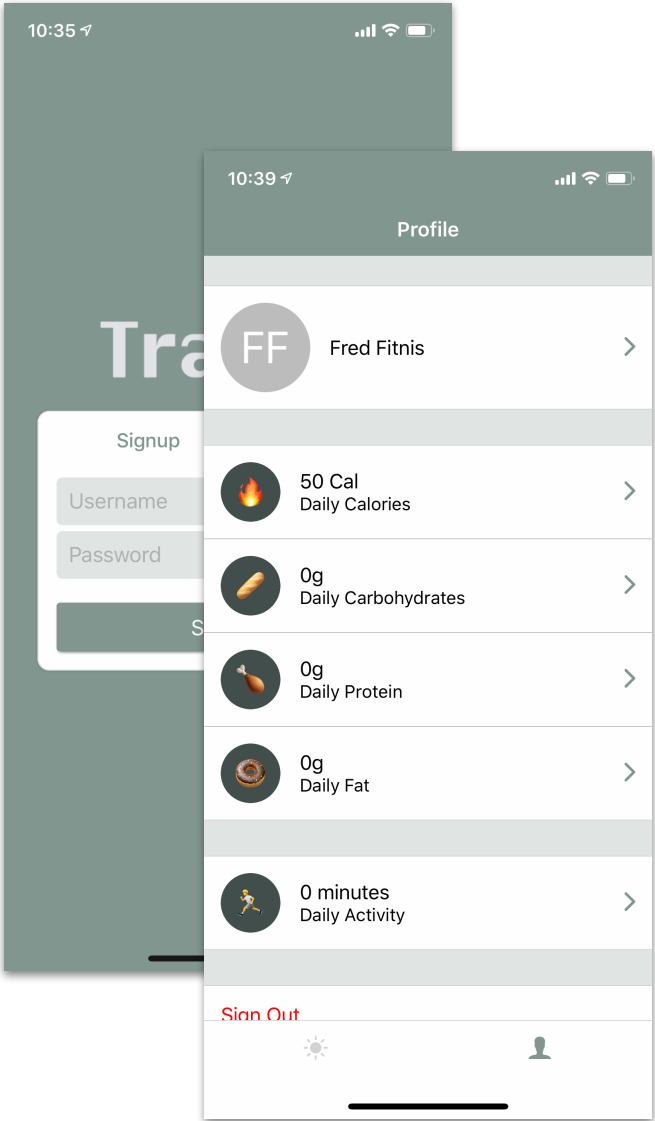
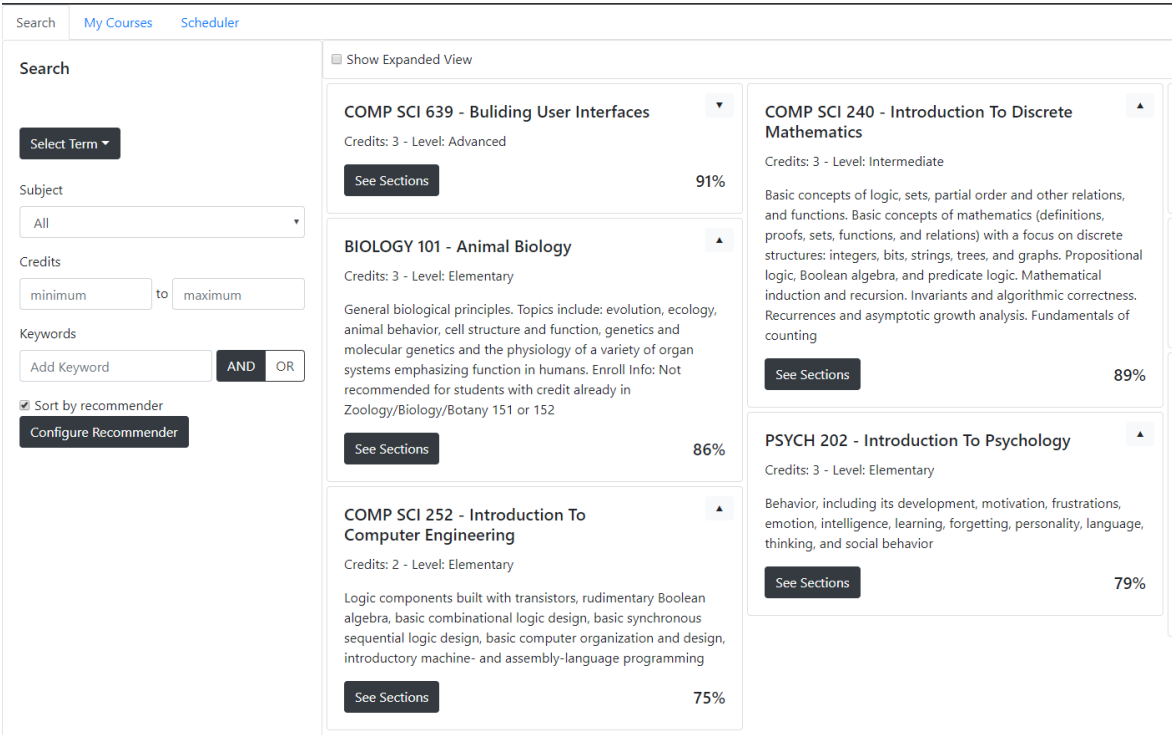
- Why am I feeling terrible?
- That means you are on the path to enlightenment!



¹Kruger & Dunning, 1999, Unskilled and Unaware of It...

Accomplishments

- You now know the basics of UX design *and* UX development.
 - Very few CS or design students can say this.
 - You have a clear path to knowing and learning more.
- You know some cutting-edge stuff.
 - React is now industry standard.
 - No one knows how to build voice assistants.
- You have done enough work for a design portfolio.



Quiz 1

Complete the Canvas quiz.



canvas

What we will learn today?

- Introduction to Dialogflow, Recap
- Dialogflow Building Blocks, Part 2
- Fulfillment

Introduction to Dialogflow, Recap

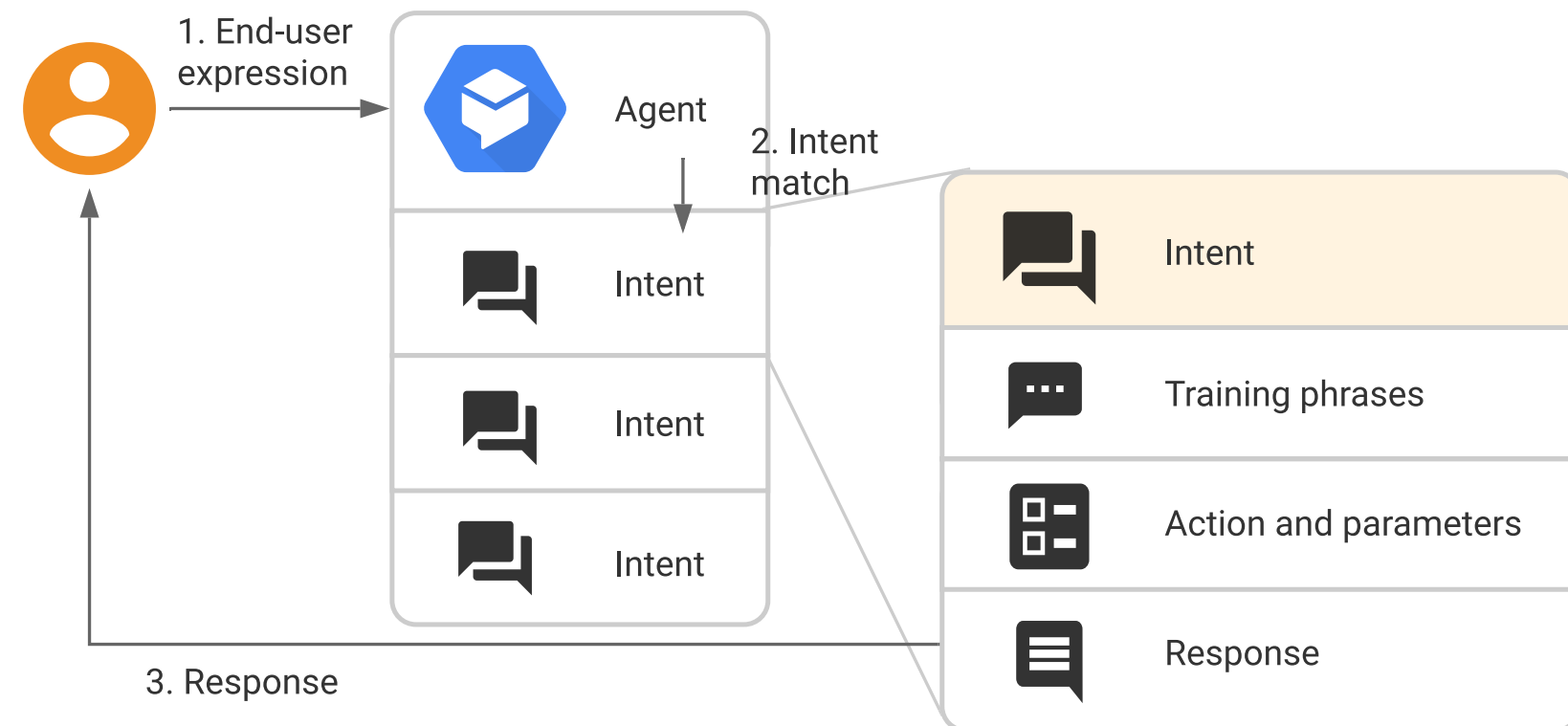
What is Dialogflow?

Dialogflow is an end-to-end, build-once deploy-everywhere development suite for conversational interfaces for websites, mobile applications, and IoT devices (e.g., smart speakers).

How does Dialogflow work?⁸

The process within Dialogflow involves:

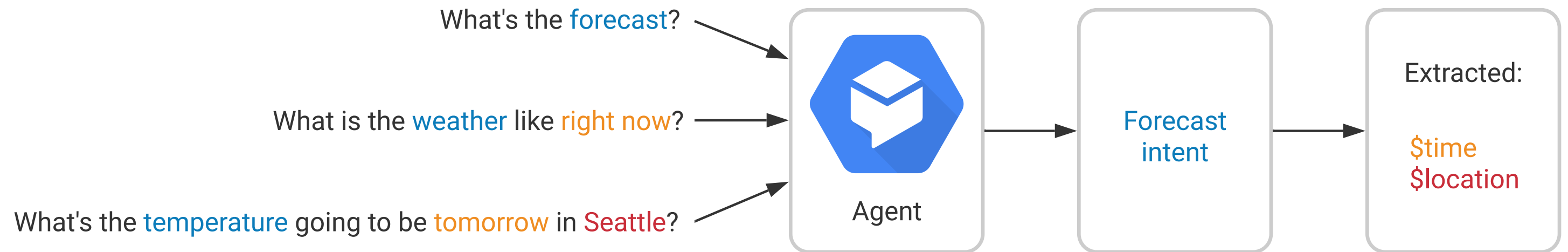
1. User expression
2. Intent matching
3. System response



⁸Image source

What is an *agent*?

Definition: A Dialogflow agent is a virtual agent that handles conversations with users (similar to a human call agent).⁹

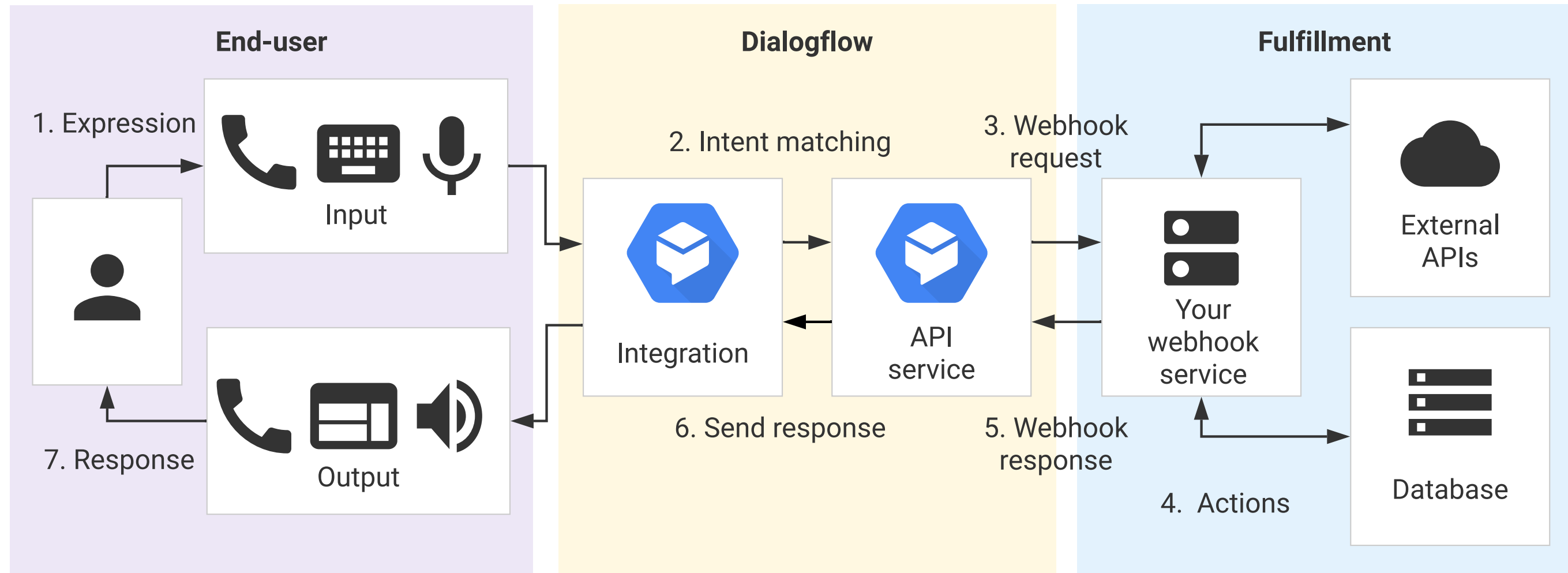


⁹ Image source

Agents are high-level containers for a number of building blocks:

- Agent settings
- Intents
- Entities
- Knowledge
- Integrations
- Fulfillment

The End-to-end Dialogflow Workflow¹⁰



¹⁰Image source

Dialogflow Building Blocks, Part 2

Contexts

What are *contexts*?

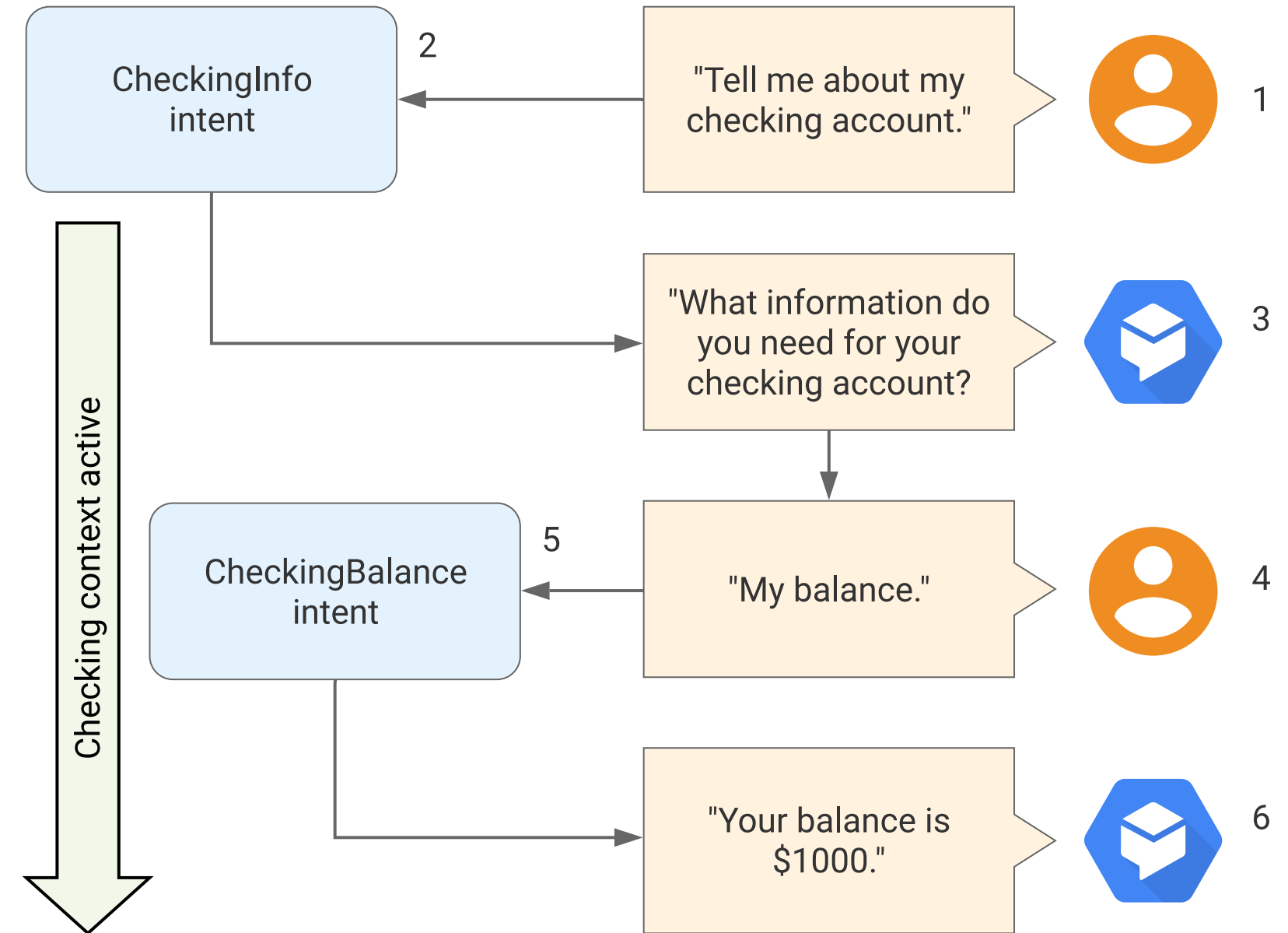
Definition: Context is the *common ground* that communicators have about the conversation that allows them to infer ambiguous speech, such as "pass me that."

In Dialogflow, contexts provide the agent with common ground on the *intent* and respond to ambiguous speech.

Contexts allow us to control the order of intent matching and define different behaviors for intents with the same training phrases.

Let's Look at An Example⁵

A checking context is defined for the CheckingInfo intent and is active, so that the agent is more likely to recognize the CheckingBalance intent.



⁵Image source

Defining Contexts

Contexts for intents are defined as *input context* and *output context*.

Context Example

A: Welcome to WiscShop!

U: Hello!

A: What are you looking for?

U: Hats.

Agent activates the looking-for-hats output context.

A: Great. How can I help?

U: Show me what's on sale.

Agent matches ShowSaleItems, input context looking-for-hats.

Input Context

Input context are added to intents, and they increase the likelihood of an intent being matched when the context is active.

Output Context

Output contexts are added to intents, and, when they are added, they are activated or renewed.

Multiple output contexts can be applied to an intent for finer intent matching.

The lifespan of an output context determines how long the context will be active for.

Context Lifespan

Contexts are designed to expire after some time.

- By default, they expire after 5 requests or 20 minutes once they are activated.
- The lifespan can be specified in terms of number of turns.

Follow-up Intents

Follow-up intents are children to parent intents and automatically set to inherit the context from the parent intent.

Intent name	Training phrase	Input context	Output context	Intent response
Appointment	Hello		appointment-followup	Would you like to make an appointment?
↳ Appointment - yes	Yes	appointment-followup	appointment-yes-followup	Would you like a haircut?
↳↳ Haircut - yes	Yes	appointment-yes-followup		Your appointment is set.
↳↳ Haircut - yes	No	appointment-yes-followup		Goodbye.
↳ Appointment - no	No	appointment-followup		Goodbye.

Events

What Are Events?

Definition: Events trigger intents based on external events rather than user queries.

We can use *platform events* or create our *custom events*.

Platform Events

Definition: Events that are triggered by actions users take on platforms that Dialogflow interacts with, such as Google Assistant, Slack, and Facebook Messenger.

Platform	Event	Description
Multiple	WELCOME	Generic welcome event for any platform.
Actions on Google	GOOGLE_ ASSISTANT_ WELCOME	Triggered when the user starts a conversation with your action.
Slack	SLACK_ WELCOME	Triggered when the user starts a conversation with your Slack bot.
Facebook	FACEBOOK_ WELCOME	Triggered when the user starts a conversation with your Facebook Messenger bot.

Custom Events

Definition: Events defined for communication that cannot be captured through text or voice, such as button clicks, authorization, or timeouts.

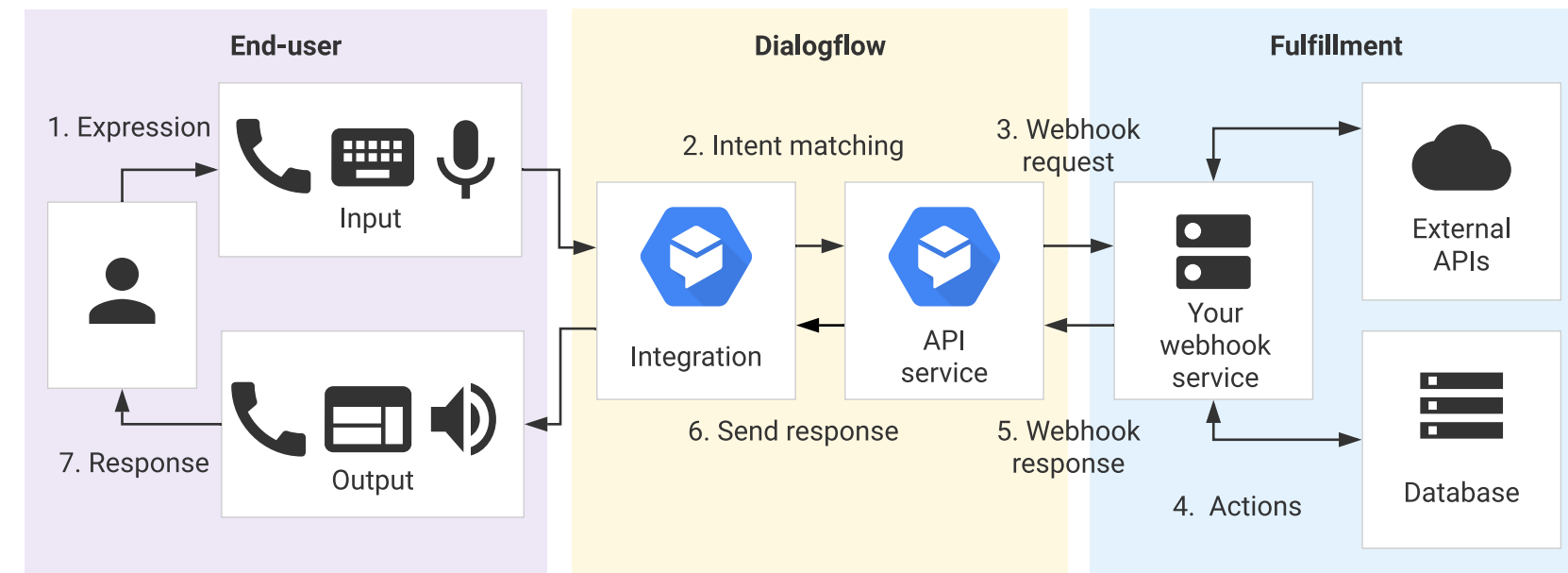
Fulfillment

What Is Fulfillment?⁶

Definition: Fulfillment allows the integration of Dialogflow agents with web services.

Intents are *enabled* for fulfillment.

Once enabled, intents will communicate with external APIs using a *webhook*.



⁶Image source

What Is A Webhook?

Definition: A *webhook* is a web server endpoint that allows applications to interact with the server API.

Webhooks in Dialogflow

Dialogflow fulfillment will follow the following course of action:

1. When an intent is enabled for fulfillment, it will make an HTTP POST request to the associated webhook with a JSON object about the matched event.
2. The webhook is designed to respond to your request appropriately, e.g., by looking up information in a database.
3. The webhook responds back with instructions for what Dialogflow should do next.

Standard POST request format:

POST `https://my-service.com/action`

Headers:

`//user defined headers`

Content-type: `application/json`

POST body: {

`// JSON`

}

You have to design your server to be able to process the JSON that's passed by the POST and to respond in a way that Dialogflow can process.

In the assignment, we did this for you — we are using a webhook and external code to connect your agent with our server.

If you are using Dialogflow to connect with Google services (more common, preferred by Dialogflow), you can use the Inline Editor. (Our webhook works like the Dialogflow Inline Editor.)

We will cover some of the basics next.

Quiz 2

Complete the Canvas quiz.



canvas

Dialogflow Fulfillment

Process for creating a custom fulfillment:

1. Enable fulfillment for the intent
2. Go to the Inline Editor and locate where the request is made
3. Define a function for your intent
4. Connect the intent to the function
5. Deploy

Step 1. Enable fulfillment for the intent

Standard procedure

Inline Editor (Powered by Cloud Functions for Firebase)

ENABLED 

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

index.js package.json



```
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
14   console.log('Dialogflow Request body: ' + JSON.stringify(request.body));
15
16   function welcome(agent) {
17     agent.add(`Welcome to my agent!`);
18   }
19
20   function fallback(agent) {
21     agent.add(`I didn't understand`);
22     agent.add(`I'm sorry, can you try again?`);
23   }
24
```

DEPLOY

Our procedure

Fulfillment ?



 Enable webhook call for this intent

 Enable webhook call for slot filling

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="Enter URL"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/> <input type="text" value="Enter value"/>	
	+ Add header	
SMALL TALK	<input type="text" value="Disable webhook for Smalltalk"/>	

Step 2. Locate where the request is made

Standard procedure

Inline Editor (Powered by Cloud Functions for Firebase)

ENABLED 

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)



```
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
14   console.log('Dialogflow Request body: ' + JSON.stringify(request.body));
15
16   function welcome(agent) {
17     agent.add('Welcome to my agent!');
18   }
19
20   function fallback(agent) {
21     agent.add('I didn\'t understand');
22     agent.add('I\'m sorry, can you try again?');
23   }
24
```

Our procedure

```
app.post('/', express.json(), (req, res) => {
  const agent = new WebhookClient({
    request : req, response: res
  })
  // Do your work here
})
```

Step 3. Define a function for your intent

Define a function for each intent:

```
function welcome() {  
    agent.add('Welcome to our service!')  
}
```

Step 4. Connect the intent to the function

Connect the function and intent through the intent map:

```
let intentMap = new Map()  
intentMap.set('<IntentName>', welcome)
```

Step 5. Deploy

Standard procedure

Inline Editor (Powered by Cloud Functions for Firebase)

ENABLED 

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

index.js package.json 

```
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
14   console.log('Dialogflow Request body: ' + JSON.stringify(request.body));
15
16   function welcome(agent) {
17     agent.add('Welcome to my agent!');
18   }
19
20   function fallback(agent) {
21     agent.add('I didn\'t understand');
22     agent.add('I\'m sorry, can you try again?');
23   }
24
```

DEPLOY

Our procedure

You don't need to do anything!

Now, let's get to the // Do your work part:

- Accessing user input
- Accessing user entities
- Setting and getting context

Everything is done through your agent object.

How to Access User Input

To get the entire user query:

```
agent.query
```

Returns a string with the entire query and `null` if no value.

How to Access User Entities

To get user entities and other information associated with the intent:

`agent.parameters`

Returns a JS object with `key:value` pairs and `null` if no value.

Accessing Specific Parameters

We need to use `$parameter-name` to access a specific parameter. E.g.:

`agent.parameters.color`

Setting and getting context

To set a new outgoing context:

```
agent.setContext(context) // string or object
```

To get a context:

```
agent.getContext(contextName) // string name of context
```

Returns the context object.

An Example

Imagine you are on the product page on WiscShop:

User: I'd like to buy this t-shirt.

intent = purchaseItem; entity = { item: t-shirt, number: null; color: null };

Agent: How many do you want?

User: 3

```
intent = purchaseItem; entity = { item: t-shirt, number: 3; color: null }
```

Agent: What color would you like?

User: Black

```
intent = purchaseItem; entity = { item: t-shirt, number: 3; color: black }
```

What do we do now?

Find which page they are on to find product ID.

```
fetch('URL/application/', <options>) // GET
```

Returns with a page key, e.g., "page": "/username/tshirts/products/14"

Grab 14, use the following three times:

```
fetch('URL/application/products/14', <options>) // POST
```

Formulate your response:

```
agent.add('Ok, I've added ' + agent.parameters.number + ' ' + agent.parameters.item)
```

Add user response and agent response to the messages.

Imagine that the user wanted to hear reviews first before purchasing, once the product was selected, we would want to set the context to that product so that we don't have to ask again.

```
agent.context.set({  
  'name': 'current-product',  
  'lifespan': 5,  
  'parameters': {  
    'id': 14  
  }  
});
```

```
agent.context.get('current-product');  
  
{  
  'name': 'current-product',  
  'lifespan': 5,  
  'parameters': {  
    'id': 14  
  }  
}
```

To Learn More

- Dialogflow documentation
- Online tutorials: e.g., Chatbots Life

Quiz 3

Complete the Canvas quiz.



Quiz 4

Complete the Canvas quiz.



canvas

What did we learn today?

- Introduction to Dialogflow, Recap
- Dialogflow Building Blocks, Part 2
- Fulfillment