**Homework 10: Alexa (Total Points: 100)**

Due: Sunday December 2nd, 11:59PM ET

**Problem 1 - Build a Basic Alexa Skill (50 points)**

Build your own Alexa skill using the “Fact Skill” as your starting template.

* Build the skill to return facts about yourself (they do not have to be true facts).
* Include at least 15 facts.

Paste a screenshot of your Fact Skill's JSON Editor. Ensure that the full JSON is visible (you can include multiple screenshots if necessary) (10 points)

|  |
| --- |
|  |

Paste below the contents of the index.js Lambda function source code with your modifications. (10 points)

|  |
| --- |
| /\* eslint-disable func-names \*/ /\* eslint-disable no-console \*/  const Alexa = require('ask-sdk');  const GetNewFactHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'LaunchRequest'  || (request.type === 'IntentRequest'  && request.intent.name === 'GetNewFactIntent');  },  handle(handlerInput) {  const factArr = data;  const factIndex = Math.floor(Math.random() \* factArr.length);  const randomFact = factArr[factIndex];  const speechOutput = GET\_FACT\_MESSAGE + randomFact;   return handlerInput.responseBuilder  .speak(speechOutput)  .withSimpleCard(SKILL\_NAME, randomFact)  .getResponse();  }, };  const HelpHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'IntentRequest'  && request.intent.name === 'AMAZON.HelpIntent';  },  handle(handlerInput) {  return handlerInput.responseBuilder  .speak(HELP\_MESSAGE)  .reprompt(HELP\_REPROMPT)  .getResponse();  }, };  const ExitHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'IntentRequest'  && (request.intent.name === 'AMAZON.CancelIntent'  || request.intent.name === 'AMAZON.StopIntent');  },  handle(handlerInput) {  return handlerInput.responseBuilder  .speak(STOP\_MESSAGE)  .getResponse();  }, };  const SessionEndedRequestHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'SessionEndedRequest';  },  handle(handlerInput) {  console.log(`Session ended with reason: ${handlerInput.requestEnvelope.request.reason}`);   return handlerInput.responseBuilder.getResponse();  }, };  const ErrorHandler = {  canHandle() {  return true;  },  handle(handlerInput, error) {  console.log(`Error handled: ${error.message}`);   return handlerInput.responseBuilder  .speak('Sorry, an error occurred.')  .reprompt('Sorry, an error occurred.')  .getResponse();  }, };  const SKILL\_NAME = 'Okonech Facts'; const GET\_FACT\_MESSAGE = 'Here\'s your fact: '; const HELP\_MESSAGE = 'You can say tell me an okonech fact, or, you can say exit... What can I help you with?'; const HELP\_REPROMPT = 'What can I help you with?'; const STOP\_MESSAGE = 'Goodbye!';  const data = [  'Alex is a competitive powerlifter',  'Alex competitively plays Magic the Gathering',  'Alex is 30 years old',  'Alex has trained guinea pigs',  'Alex once burned toast with an argon laser',  'Alex can run a mile in under 6 minutes',  'Alex plays basketball',  'Alex really likes playing Beat Saber',  'Alex is a night owl',  'Alex enjoys cooking simple food',  'Alex made money in last years crypto craze',  'Alex hates the commute to work',  'Alex prefers indoor sports to outdoor sports',  'Alex likes all-you-can-eat restaurants',  'Alex knows a lot halberds' ];  const skillBuilder = Alexa.SkillBuilders.standard();  exports.handler = skillBuilder  .addRequestHandlers(  GetNewFactHandler,  HelpHandler,  ExitHandler,  SessionEndedRequestHandler  )  .addErrorHandlers(ErrorHandler)  .lambda(); |

Test your Facts Skill using the **Test** tab in the Alexa Developer Console and paste the screenshots of two interactions. Make sure the input and the response are visible for each interaction. (30 points)

|  |
| --- |
|  |

**Problem 2: Build an Alexa Skill using the Decision Tree template (50 Points)**

Paste a screenshot of your Decision Tree Skill's JSON Editor. Ensure that the full JSON is visible (you can include multiple screenshots if necessary) (10 points)

|  |
| --- |
| {  "interactionModel": {  "languageModel": {  "invocationName": "decision tree",  "intents": [  {  "name": "AMAZON.CancelIntent",  "samples": []  },  {  "name": "AMAZON.HelpIntent",  "samples": []  },  {  "name": "AMAZON.StopIntent",  "samples": []  },  {  "name": "CouchPotatoIntent",  "slots": [],  "samples": [  "couch potato",  "lazy"  ]  },  {  "name": "RecommendationIntent",  "slots": [  {  "name": "playerType",  "type": "playerTypeType",  "samples": [  "{I\_Want} to be a {playerType} player",  "{playerType}"  ]  },  {  "name": "strength",  "type": "strengthType",  "samples": [  "{I\_Am} {article} {strength}",  "{strength}"  ]  },  {  "name": "jumpHeight",  "type": "jumpHeightType",  "samples": [  "{I\_Can} {jumpHeight}",  "{jumpHeight}"  ]  },  {  "name": "I\_Can",  "type": "iSubjectType"  },  {  "name": "I\_Want",  "type": "iSubjectType"  },  {  "name": "article",  "type": "articleType"  },  {  "name": "I\_Am",  "type": "IAmType"  }  ],  "samples": [  "recommend a sport",  "{I\_Can} {jumpHeight}",  "{I\_Am} {article} {strength}",  "{I\_Want} to be a {playerType} player",  "to recommend a sport",  "start looking"  ]  }  ],  "types": [  {  "name": "articleType",  "values": [  {  "name": {  "value": "a"  }  },  {  "name": {  "value": "an"  }  }  ]  },  {  "name": "IAmType",  "values": [  {  "name": {  "value": "I am"  }  }  ]  },  {  "name": "iSubjectType",  "values": [  {  "id": "true",  "name": {  "value": "positive",  "synonyms": [  "I want",  "I prefer",  "I really like",  "I love",  "I like"  ]  }  },  {  "id": "false",  "name": {  "value": "negative",  "synonyms": [  "I don't want",  "I really don't like",  "I don't like",  "I hate"  ]  }  }  ]  },  {  "name": "strengthType",  "values": [  {  "name": {  "value": "strong",  "synonyms": [  "muscular",  "explosive",  "burly"  ]  }  },  {  "name": {  "value": "weak",  "synonyms": [  "skinny",  "frail"  ]  }  }  ]  },  {  "name": "jumpHeightType",  "values": [  {  "name": {  "value": "true",  "synonyms": [  "yes",  "yeah",  "high",  "far",  "tall"  ]  }  },  {  "name": {  "value": "false",  "synonyms": [  "no",  "nope",  "short"  ]  }  }  ]  },  {  "name": "playerTypeType",  "values": [  {  "name": {  "value": "team",  "synonyms": [  "play on a team",  "part of a team",  "group oriented",  "team player"  ]  }  },  {  "name": {  "value": "solo",  "synonyms": [  "solo player",  "on my own",  "by myself",  "alone"  ]  }  }  ]  }  ]  },  "dialog": {  "intents": [  {  "name": "RecommendationIntent",  "confirmationRequired": false,  "prompts": {},  "slots": [  {  "name": "playerType",  "type": "playerTypeType",  "confirmationRequired": false,  "elicitationRequired": true,  "prompts": {  "elicitation": "Elicit.Intent-RecommendationIntent.IntentSlot-playerType"  }  },  {  "name": "strength",  "type": "strengthType",  "confirmationRequired": false,  "elicitationRequired": true,  "prompts": {  "elicitation": "Elicit.Intent-RecommendationIntent.IntentSlot-strength"  }  },  {  "name": "jumpHeight",  "type": "jumpHeightType",  "confirmationRequired": false,  "elicitationRequired": true,  "prompts": {  "elicitation": "Elicit.Intent-RecommendationIntent.IntentSlot-jumpHeight"  }  },  {  "name": "I\_Can",  "type": "iSubjectType",  "confirmationRequired": false,  "elicitationRequired": false,  "prompts": {}  },  {  "name": "I\_Want",  "type": "iSubjectType",  "confirmationRequired": false,  "elicitationRequired": false,  "prompts": {}  },  {  "name": "article",  "type": "articleType",  "confirmationRequired": false,  "elicitationRequired": false,  "prompts": {}  },  {  "name": "I\_Am",  "type": "IAmType",  "confirmationRequired": false,  "elicitationRequired": false,  "prompts": {}  }  ]  }  ]  },  "prompts": [  {  "id": "Elicit.Intent-RecommendationIntent.IntentSlot-playerType",  "variations": [  {  "type": "PlainText",  "value": "Which phrase best describes your thoughts about sports, I want to play on a team, or I want to play solo."  }  ]  },  {  "id": "Elicit.Intent-RecommendationIntent.IntentSlot-strength",  "variations": [  {  "type": "PlainText",  "value": "Would you say that you are naturally strong or weak?"  }  ]  },  {  "id": "Elicit.Intent-RecommendationIntent.IntentSlot-jumpHeight",  "variations": [  {  "type": "PlainText",  "value": "Can you jump high?"  }  ]  }  ]  } } |

Paste below the contents of the index.js Lambda function source code with your modifications. (10 points)

|  |
| --- |
| /\*\*  Copyright 2017 Amazon.com, Inc. and its affiliates. All Rights Reserved.  Licensed under the Amazon Software License (the "License").  You may not use this file except in compliance with the License.  A copy of the License is located at  http://aws.amazon.com/asl/  or in the "license" file accompanying this file. This file is distributed  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express  or implied. See the License for the specific language governing  permissions and limitations under the License.  This skill demonstrates how to use Dialog Management to delegate slot  elicitation to Alexa. For more information on Dialog Directives see the  documentation: https://developer.amazon.com/docs/custom-skills/dialog-interface-reference.html  This skill also uses entity resolution to define synonyms. Combined with  dialog management, the skill can ask the user for clarification of a synonym  is mapped to two slot values.  \*\*/  "use strict";  const Alexa = require("alexa-sdk");  // For detailed tutorial on how to make an Alexa skill,  // please visit us at http://alexa.design/build  let handlers = {  "LaunchRequest": function () {  console.log("in LaunchRequest");  this.response.speak("Welcome to Decision Tree. I will recommend a sport for you. Do you want to start looking or be a couch potato?");  this.response.listen("Do you want a sport reccomendation or to be a couch potato?");  this.emit(":responseReady");  },  "CouchPotatoIntent": function () {  this.response.speak("You don't want to play sports? Have fun wasting away on the couch.");  this.emit(":responseReady");  },  "RecommendationIntent": function () {  // delegate to Alexa to collect all the required slots  let filledSlots = delegateSlotCollection.call(this);  if (!filledSlots) {  return;  }  console.log("filled slots: " + JSON.stringify(filledSlots));  // at this point, we know that all required slots are filled.  let slotValues = getSlotValues(filledSlots);  console.log(JSON.stringify(slotValues));  let key = `${slotValues.playerType.resolved}-${slotValues.strength.resolved}-${slotValues.jumpHeight.resolved}`;  let sport = options[slotsToOptionsMap[key]];  console.log("look up key: ", key, "object: ", sport);  let speechOutput = "So you want to be a " + slotValues.playerType.resolved +  " player. You are naturally " + slotValues.strength.resolved +  " and you " + (slotValues.jumpHeight.resolved === "true" ? "can" : "can't" ) +  " jump high " +  ". You should consider being a " + sport.name;  console.log("Speech output: ", speechOutput);  this.response.speak(speechOutput);  this.emit(":responseReady");  },  "SessionEndedRequest": function () {  console.log("Session ended with reason: " + this.event.request.reason);  },  "AMAZON.StopIntent": function () {  this.response.speak("Bye");  this.emit(":responseReady");  },  "AMAZON.HelpIntent": function () {  this.response.speak("This is Decision Tree. I can help you find the perfect sport. " +  "You can say, recommend a sport.").listen("Would you like a sport to play or do you want to be a couch potato?");  this.emit(":responseReady");  },  "AMAZON.CancelIntent": function () {  this.response.speak("Bye");  this.emit(":responseReady");  },  "Unhandled": function () {  this.response.speak("Sorry, I didn't get that. You can try: 'alexa, tell Decision Tree to" +  " recommend a sport.'");  }  };  exports.handler = function (event, context) {  // Each time your lambda function is triggered from your skill,  // the event's JSON will be logged. Check Cloud Watch to see the event.  // You can copy the log from Cloud Watch and use it for testing.  console.log("====================");  console.log("REQUEST: " + JSON.stringify(event));  console.log("====================");  let alexa = Alexa.handler(event, context);  // Part 3: Task 4  // alexa.dynamoDBTableName = 'petMatchTable';  alexa.registerHandlers(handlers);  alexa.execute();  };  const REQUIRED\_SLOTS = [  "jumpHeight",  "strength",  "playerType"  ];  const slotsToOptionsMap = {  "solo-strong-true": 0,  "solo-strong-false": 1,  "solo-weak-true": 2,  "solo-weak-false": 3,  "team-strong-true": 4,  "team-strong-false": 5,  "team-weak-true": 6,  "team-weak-false": 7,  };  const options = [  {"name": "Olympic lifter", "description": ""},  {"name": "Powerlifter", "description": ""},  {"name": "Sprinter", "description": ""},  {"name": "Marathon runner", "description": ""},  {"name": "Football player", "description": ""},  {"name": "Wrestler", "description": ""},  {"name": "Volleyball player", "description": ""},  {"name": "Relay racer", "description": ""}  ]  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // \*\* Helper functions from  // \*\* These should not need to be edited  // \*\* www.github.com/alexa/alexa-cookbook  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // \*\* Dialog Management  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  function getSlotValues(filledSlots) {  //given event.request.intent.slots, a slots values object so you have  //what synonym the person said - .synonym  //what that resolved to - .resolved  //and if it's a word that is in your slot values - .isValidated  let slotValues = {};  console.log("The filled slots: " + JSON.stringify(filledSlots));  Object.keys(filledSlots).forEach(function (item) {  // console.log("item in filledSlots: "+JSON.stringify(filledSlots[item]));  let name = filledSlots[item].name;  //console.log("name: "+name);  if (filledSlots[item] &&  filledSlots[item].resolutions &&  filledSlots[item].resolutions.resolutionsPerAuthority[0] &&  filledSlots[item].resolutions.resolutionsPerAuthority[0].status &&  filledSlots[item].resolutions.resolutionsPerAuthority[0].status.code) {  switch (filledSlots[item].resolutions.resolutionsPerAuthority[0].status.code) {  case "ER\_SUCCESS\_MATCH":  slotValues[name] = {  "synonym": filledSlots[item].value,  "resolved": filledSlots[item].resolutions.resolutionsPerAuthority[0].values[0].value.name,  "isValidated": true  };  break;  case "ER\_SUCCESS\_NO\_MATCH":  slotValues[name] = {  "synonym": filledSlots[item].value,  "resolved": filledSlots[item].value,  "isValidated":false  };  break;  }  } else {  slotValues[name] = {  "synonym": filledSlots[item].value,  "resolved": filledSlots[item].value,  "isValidated": false  };  }  },this);  //console.log("slot values: "+JSON.stringify(slotValues));  return slotValues;  }  // This function delegates multi-turn dialogs to Alexa.  // For more information about dialog directives see the link below.  // https://developer.amazon.com/docs/custom-skills/dialog-interface-reference.html  function delegateSlotCollection() {  console.log("in delegateSlotCollection");  console.log("current dialogState: " + this.event.request.dialogState);  if (this.event.request.dialogState === "STARTED") {  console.log("in STARTED");  console.log(JSON.stringify(this.event));  let updatedIntent = this.event.request.intent;  // optionally pre-fill slots: update the intent object with slot values  // for which you have defaults, then return Dialog.Delegate with this  // updated intent in the updatedIntent property  disambiguateSlot.call(this);  console.log("disambiguated: " + JSON.stringify(this.event));  this.emit(":delegate", updatedIntent);  } else if (this.event.request.dialogState !== "COMPLETED") {  console.log("in not completed");  let updatedIntent = this.event.request.intent;  //console.log(JSON.stringify(this.event));  disambiguateSlot.call(this);  this.emit(":delegate", updatedIntent);  } else {  console.log("in completed");  //console.log("returning: "+ JSON.stringify(this.event.request.intent));  // Dialog is now complete and all required slots should be filled,  // so call your normal intent handler.  return this.event.request.intent.slots;  }  return null;  }  // If the user said a synonym that maps to more than one value, we need to ask  // the user for clarification. Disambiguate slot will loop through all slots and  // elicit confirmation for the first slot it sees that resolves to more than  // one value.  function disambiguateSlot() {  let currentIntent = this.event.request.intent;  let prompt = "";  Object.keys(this.event.request.intent.slots).forEach(function (slotName) {  let currentSlot = currentIntent.slots[slotName];  // let slotValue = slotHasValue(this.event.request, currentSlot.name);  if (currentSlot.confirmationStatus !== "CONFIRMED" &&  currentSlot.resolutions &&  currentSlot.resolutions.resolutionsPerAuthority[0]) {  if (currentSlot.resolutions.resolutionsPerAuthority[0].status.code === "ER\_SUCCESS\_MATCH") {  // if there's more than one value that means we have a synonym that  // mapped to more than one value. So we need to ask the user for  // clarification. For example if the user said "mini dog", and  // "mini" is a synonym for both "small" and "tiny" then ask "Did you  // want a small or tiny dog?" to get the user to tell you  // specifically what type mini dog (small mini or tiny mini).  if (currentSlot.resolutions.resolutionsPerAuthority[0].values.length > 1) {  prompt = "Which would you like";  let size = currentSlot.resolutions.resolutionsPerAuthority[0].values.length;  currentSlot.resolutions.resolutionsPerAuthority[0].values.forEach(function (element, index, arr) {  prompt += ` ${(index === size - 1) ? " or" : " "} ${element.value.name}`;  });  prompt += "?";  let reprompt = prompt;  // In this case we need to disambiguate the value that they  // provided to us because it resolved to more than one thing so  // we build up our prompts and then emit elicitSlot.  this.emit(":elicitSlot", currentSlot.name, prompt, reprompt);  }  } else if (currentSlot.resolutions.resolutionsPerAuthority[0].status.code === "ER\_SUCCESS\_NO\_MATCH") {  // Here is where you'll want to add instrumentation to your code  // so you can capture synonyms that you haven't defined.  console.log("NO MATCH FOR: ", currentSlot.name, " value: ", currentSlot.value);  if (REQUIRED\_SLOTS.indexOf(currentSlot.name) > -1) {  prompt = "What " + currentSlot.name + " are you looking for";  this.emit(":elicitSlot", currentSlot.name, prompt, prompt);  }  }  }  }, this);  }  // Given the request an slot name, slotHasValue returns the slot value if one  // was given for `slotName`. Otherwise returns false.  function slotHasValue(request, slotName) {  let slot = request.intent.slots[slotName];  // uncomment if you want to see the request  // console.log("request = "+JSON.stringify(request));  let slotValue;  // if we have a slot, get the text and store it into speechOutput  if (slot && slot.value) {  // we have a value in the slot  slotValue = slot.value.toLowerCase();  return slotValue;  } else {  // we didn't get a value in the slot.  return false;  }  } |

Test your Facts Skill using the **Test** tab in the Alexa Developer Console and paste the screenshots of two interactions. Make sure the input and the response are visible for each interaction. (30 points)

|  |
| --- |
|  |

**Bonus: (10 points)**

Modify your Fact Skill from problem 1 to retrieve the facts from a remote RESTful service endpoint.

One example is to code your service to retrieve weather information from a random list of locations. See:

<https://dzone.com/articles/4-free-weather-providers-api-to-develop-weather-ap-1>

for information on free APIs that you could use for this.

You can use ANY source you like, as long as your Lambda function invokes a publicly available RESTful API to retrieve the information (e.g. sports statistics, movie data, etc).

Manually invoke the the external RESTful API. Paste a screenshot of the command and response. If you used a program like Postman, provide a screenshot of the invocation. Make sure the input and the response are visible. (5 points)

|  |
| --- |
|  |

Test your modified Fact Skill using the **Test** tab in the Alexa Developer Console and paste the screenshots of two interactions. Make sure the input and the response are visible for each interaction. (5 points)

|  |
| --- |
| /\* eslint-disable func-names \*/  /\* eslint-disable no-console \*/  const Alexa = require('ask-sdk');  const http = require('http');  const GetNewFactHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'LaunchRequest'  || (request.type === 'IntentRequest'  && request.intent.name === 'GetNewFactIntent');  },  handle(handlerInput) {  return httpGet((theResult) => {  console.log('received : ' + theResult);  return handlerInput.responseBuilder  .speak(GET\_FACT\_MESSAGE + theResult)  .withSimpleCard(SKILL\_NAME, theResult)  .getResponse();  });  },  };  const HelpHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'IntentRequest'  && request.intent.name === 'AMAZON.HelpIntent';  },  handle(handlerInput) {  return handlerInput.responseBuilder  .speak(HELP\_MESSAGE)  .reprompt(HELP\_REPROMPT)  .getResponse();  },  };  const ExitHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'IntentRequest'  && (request.intent.name === 'AMAZON.CancelIntent'  || request.intent.name === 'AMAZON.StopIntent');  },  handle(handlerInput) {  return handlerInput.responseBuilder  .speak(STOP\_MESSAGE)  .getResponse();  },  };  const SessionEndedRequestHandler = {  canHandle(handlerInput) {  const request = handlerInput.requestEnvelope.request;  return request.type === 'SessionEndedRequest';  },  handle(handlerInput) {  console.log(`Session ended with reason: ${handlerInput.requestEnvelope.request.reason}`);  return handlerInput.responseBuilder.getResponse();  },  };  const ErrorHandler = {  canHandle() {  return true;  },  handle(handlerInput, error) {  console.log(`Error handled: ${error.message}`);  return handlerInput.responseBuilder  .speak('Sorry, an error occurred.')  .reprompt('Sorry, an error occurred.')  .getResponse();  },  };  function httpGet(callback) {  const options = {  host: 'catfact.ninja',  path: '/fact',  method: 'GET',  };  const req = http.request(options, res => {  res.setEncoding('utf8');  const responseString = '';  // accept incoming data asynchronously  res.on('data', chunk => {  responseString = responseString + chunk;  });  // return the data when streaming is complete  res.on('end', () => {  console.log(responseString);  callback(responseString);  });  });  req.end();  }  const SKILL\_NAME = 'Cat Facts';  const GET\_FACT\_MESSAGE = 'Here\'s your fact: ';  const HELP\_MESSAGE = 'You can say tell me a cat fact, or, you can say exit... What can I help you with?';  const HELP\_REPROMPT = 'What can I help you with?';  const STOP\_MESSAGE = 'Goodbye!';  const skillBuilder = Alexa.SkillBuilders.standard();  exports.handler = skillBuilder  .addRequestHandlers(  GetNewFactHandler,  HelpHandler,  ExitHandler,  SessionEndedRequestHandler  )  .addErrorHandlers(ErrorHandler)  .lambda(); |