Exercise 15: Final Project (Therapist Bot)

For your final coding project, you'll be building a therapist bot. Don't be intimidated – it's easier than it sounds. (Okay, it's not a good therapist by any stretch of the imagination.)

History

Back in 1966, the first therapist bot was created by artificial intelligence professor Joseph Weizenbaum at MIT. There was a theory (a correct one, in my opinion) that the most effective thing a therapist could do is to actively listen, rather than give advice. So Weizenbaum created a program called ELIZA that gave the illusion it was listening to whatever someone typed in. Of course, it had no understanding, but it played a bunch of language tricks and did fairly well, especially for 1960s technology.

Weizenbaum is one of my personal heroes. He was concerned about how technology was interfering with social interaction, and this was long before the invention of the internet. In the 1970s, most artificial intelligence researchers were convinced that creating a thinking machine was possible within 10 years. Weizenbaum was one of the few who realized how special human beings are, and that it would not be so easy. Whenever I hear about how AI is going to take over the world and end the human race, I think of him. I suspect he would shake his head and say, "I've heard it all before."

Initial code

Create a new file in your text editor and paste in this code, which is the basics to get started. It has a long textbox, a submit button, and a divider to store the conversation. There is a <head> element, which makes the paragraph spacing smaller so you can see more conversation on the page. On load, the code will call a function called **initialize**. When the button is clicked, the code will call a function called **submitLine**.

```
<!DOCTYPE HTML>
<html>
<head>
<style>
p {
    line-height: 50%;
}
</style>
</head>
<body onload="initialize()">

<h1>Eliza</h1>

<p>Write here: <input id="textbox" size="150"/></p>
<p><button onclick="submitLine()">Submit</button></p>

<div id="conversation"></div>

<script>
</script>

</body>
</html>
```

Save this as **therapist.html**.

Note: Throughout this exercise, I will not be putting in comments to explain how things work. The reason is that you will do that for the next exercise.

First lines of conversation

In the script section, declare a variable called **therapySession**. You don't need to set it to anything yet. This will hold the conversation.

Next, add a function called **initialize.** This will create the first line of the conversation.

```
function initialize() {
    therapySession = "<p> I am the psychotherapist. What is your problem? </p>";
    conversation.innerHTML = therapySession;
}
```

Save and open the file in the browser. You should see this first line appear under the **Submit** button.

Next, add a function called **submitLine** to handle the button click. It will take the textbox value and add it to the **therapySession** variable. Note that the patient's lines will be italic because of the <em> (emphasis) tag.

```
function submitLine() {
    var patientLine = textbox.value;
    therapySession += "<p> <em>" + patientLine + "</em> </p>";

    conversation.innerHTML = therapySession;
}
```

Save and refresh the browser page. Enter in some text (I always start with "I am sad") and hit the **Submit** button. That line will appear in italics under the therapist's first line.

Generic Responses

The first thing to do is give the therapist the ability to give a generic response. We'll use this when there's nothing better to say. The generic responses will simply encourage the user to write more.

First, add a new constant that's an array of strings with generic responses. Here's mine, but feel free to change or add your own:

```
const genericResponses = [
    "Uh-huh.",
    "Go on.",
    "Why do you say that?",
    "That's very interesting.",
    "Fascinating...",
    "Keep talking."];
```

We're going to want to pick a random one of these. This is something we are going to do several times, so let's create a function that will take care of it. After the **submitLine** function, add a function called **randomElement** that will have a parameter that's an array and it returns a random element from that array.

How do you get random numbers in JavaScript? Well, a Google search on "JavaScript random" shows that there is a function called **Math.random** that returns a random number between 0 and 1. If you multiply that number by an integer, and strip off all the numbers after the decimal point, you'll get a random number between 0 and that integer minus 1. So if do this to the length of an array, we'll get a random index into that array. The function to strip the decimal points off a number and turn it into an integer is **Math.floor**. So our function should look like this:

```
function randomElement(myArray) {
    var index = Math.floor(Math.random() * myArray.length);
    return myArray[index];
}
```

Almost done. We just need to add two more lines before the **conversation.innerHTML** line in the **submitLine** function. This first will create a new variable to hold the therapist's line and set it to the random string in the array we created, using our new function. The second will add that new line to the **therapySession** array.

```
    var therapistLine = randomElement(genericResponses);
    therapySession += "<p>" + therapistLine + "</p>";
```

Save and refresh the browser page. Have a conversation with the therapist.

It gets frustrating really quickly, doesn't it? To me, it sounds just like someone pretending to listen. The worst part is if you ask it a question, it doesn't even acknowledge that a question was asked. Let's see if we can do better.

Having trouble? You can check my solution at: http://sdkbridge.com/prog1/Exercise15aAnswers.pdf.

Handling Questions

There's no easy way to write code to interpret and respond to questions in a meaningful way. So let's use a technique where you deflect the question back to the user. Add a constant with an array of question responses:

```
const questionResponses = [
    "Why do you ask that?",
    "What do <em>you</em> think?",
    "That's an interesting question.",
    "How long have you wanted to know that?",
    "That depends on who you ask."];
```

Feel free to change or add more phrases to the array. Note that since this is HTML, you can do things like put the "you" in italics by using the emphasis <em> tags.

Next you need to figure out if it's a question. Let's assume that if it is, it ends with a question mark. The ability to find the last character of a string is going to be one we will use several times, so let's create a separate function that does just that. If a string is 5 characters long, then the first characters have index 0, 1, 2, 3, and 4. The last character is 4, which is 5 - 1. In other words, the last character has an index of the length of the string minus one.

JavaScript strings have a method called **substring**. If you pass in just one parameter, it returns the string from the index to the end of the string. So you can use that in your function:

```
function lastChar(myString) {
    return myString.substring(myString.length - 1);
}
```

Finally, change the line that sets the **therapistLine** variable to check the last character. If it's a question mark, then get a random question response. Otherwise, get a random generic response. You can do this as an if/else statement:

```
    var therapistLine;
    if (lastChar(patientLine) == "?") {
        therapistLine = randomElement(questionResponses);
    } else {
        therapistLine = randomElement(genericResponses);
    }
```

Save and refresh the browser. Have a conversation now with some questions.

Having trouble? You can check my solution at: http://sdkbridge.com/prog1/Exercise15bAnswers.pdf.

Handling Exclamations

Sometimes the user will end a sentence with an exclamation point. In this case, it works to have the therapist bot respond to the fact that the user is an emotional state. Add some code to check if the last character is an exclamation point, and if it is, then choose a special response. Here's the list of responses I came up with for this situation:

- "Please calm down."
- "No need to get so excited."
- "You sound very passionate about that."
- "Would you care to restate that in a more neutral tone?"

See if you can add code to respond to exclamations much like you just did to respond to questions. There is more than one way to do this, but I would recommend using an if / else if / else statement to decide what the **therapistLine** variable should be set to.

Having trouble? You can check my solution at: http://sdkbridge.com/prog1/Exercise15cAnswers.pdf.

Language Manipulation

So far, our therapy bot is extremely primitive. Now you'll add a little bit of sophistication to it. The idea is that the therapist bot asks questions that actually make it sound like it was listening. For example, when the user says:
"I am sad."
We'd like the therapy bot to respond something like:
"Why do you say that you are sad?"

To transform the statement into this question, several things have to happen:

1. The beginning of the question is generated. In this case, it's "Why do you say that". Like before, you can have an array of these question beginnings and randomly pull one of them.
2. The "I am" is transformed into "you are".
3. The period at the end is removed
4. A question mark is added to the end

So, first thing, create a new function that transforms the statement into a question. Call it **createQuestion**. It will have one parameter called **patientLine**.

Removing the period

Of the steps above, you should already know how to do #1, and #4 is easy. So let's start with #3, which is removing the period at the end, if there is one.

Create an if statement that:

1. Checks to see if the last character is a period. (Use the function we've created for that purpose.)
2. If it is, set **patientLine** to be a "substring" of **patientLine** with all but the last character.

   To do this, you will use the **substring** method of the **patientLine** variable. We've used it before, but unlike in the **lastChar** function where we just had one parameter, this time you will use **substring** with two parameters. The first parameter provides the starting index (which is the beginning of the string, in this case), and the second parameter provides the length of the string (which is going to be just long enough not to include the last character). See if you can figure this out. If you are stuck, trying reading up on the **substring** method of the String class in JavaScript by searching on "JavaScript substring".

Changing Point of View

Now you'll do the hardest part, which is #2 in the list above, where you change "I am" to "you are". This involves changing the point of view from first person (about me) to second person (about you). There's a series of changes that need to be made:

- "I" to "you"
- "me" to "you"
- "am" to "are"
- "my" to "your"
- "mine" to "yours"
- etc.

Now, in theory, it should go the other way, too:

- "you" to "I"
- "you" to "me"
- "are" to "am"
- "your" to "my"
- etc.

Do you see the problem? In English, "you" is the second person equivalent of both "I" and "me", so there's no easy way to tell which it should be. (Without some sophisticated language analysis, that is.) So check if there's a "you" in the patient line, and if there is, then you'll bail out. By bailing out, I mean you'll quit the function and then choose a generic response. The easiest way to do that is to return null, and then you'll handle a null return when you call the function.

First, you need to figure out if the **patientLine** string contains "you". In order not to be case sensitive, we can convert the string to lowercase using the **toLowerCase** method. Then, to check if a string contains another string, you can use the **indexOf** method. This searches a string and finds the first time that another string appears inside of it and returns the index of the starting point. If the string never appears in side of it, it returns -1. So to see if a string contains "you", we just need to make sure that **indexOf("you")** is not -1. Add this code to be the first lines in the **createQuestion** function.

```
if (patientLine.toLowerCase().indexOf("you") != -1) {
  // Can't handle this one. Return null.
  return null;
}
```

Next, create a constant object that contains the mapping between first person and second person phrases. The first person phrase will be the property name and the second person phrase will be the property value. Put this with the other constants.

```
const povSwitches = {
    "I": "you",
    "i": "you",
    "me": "you",
    "myself": "yourself",
    "am": "are",
    "my": "your",
    "My": "your",
    "I'm": "you're",
    "I'd": "you'd",
    "I'll": "you'll",
    "i'm": "you're",
    "i'd": "you'd",
    "i'll": "you'll"
}
```

Note that the keys are in quotes because some of them have apostrophes, which would be confused with single quotes. Also note that I included small "i"s just in case the user is too lazy to capitalize.

Now, you don't want to replace all "i" letters with "you". Otherwise, a word like "happiness" would be converted to "happyouness". We want to only do it if it's a word, which means that there's space on either side. (We should also handle punctuation, but let's not worry about that for now.) This means you need to make sure there's space at the beginning and end of the string so that when you search for " I", an "I" at the start of the string will be found. After the code to remove the period at the end, create a new variable called **modifiedLine** and set it to **patientLine**, but adding a space at the beginning and a space at the end.

Here's the tricky part. You are going to loop through all properties of the **povSwitches** object and replace the key with the key's value using the string's **replace** method. (Well, first you'll add spaces on either side of both the key and the value.) If you look back at how to loop through an object's properties, you'll see it looks like this. Note that I created a new variable called **modifiedProperty** that has spaces on either side so that we are only replacing whole words.

```
for (var property in povSwitches) {
    if (povSwitches.hasOwnProperty(property)) {
        var modifiedProperty = " " + property + " ";
        modifiedLine = modifiedLine.replace(modifiedProperty,
                " " + povSwitches[property] + " ");
    }
}
```

Don't add this code in yet. You should also keep track of whether you actually made any changes. The reason for this is that if the user put in a one-word phrase, you don't want to act like it was a full sentence. In other word, if they type, "yes", you don't want the therapist to say, "Why do you say that yes?" That makes no sense.

So the solution is to have a Boolean variable called **found** that is set to false to start with. If the modified line contains the modified property, then set it to true. Then, at the end of the loop, you will know whether you made any replacements. The code will look like this:

```
var found = false;
for (var property in povSwitches) {
    if (povSwitches.hasOwnProperty(property)) {
        var modifiedProperty = " " + property + " ";
        if (modifiedLine.indexOf(modifiedProperty) != -1) {
            modifiedLine = modifiedLine.replace(modifiedProperty,
                " " + povSwitches[property] + " ");
            found = true;
        }
    }
}
```

Now, you need the beginning of the question. In the area with the other constants, create a constant array called **questionStarts** with strings for question beginnings. Here's the list I came up with, but feel free to add to it:

- Why do you say that
- How is it that
- Can you tell me more about how
- And why is it that
- Can you explain why you say that

Back in the **createQuestion** function, under the loop you just added, add some code that

1. Checks to see if **found** is true.
2. If it is, then remove the space at the end of **modifiedLine** that you added earlier. (Similar to how you removed the period from **patientLine**.)
3. Finally (still if **found** is true), return a string that's a random question beginning, a space, the modified line, and a question mark.

One last thing in this function: if you get beyond this block, then **found** is not true. So return null.

Now, you'll call the function. We need to insert it after checking for questions and exclamations. So, put that in the final else statement, and then at the end, if **therapistLine** is null, then choose a basic response. Replace your existing code with this:

```
    var therapistLine;
    if (lastChar(patientLine) == "?") {
        therapistLine = randomElement(questionResponses);
    } else if (lastChar(patientLine) == "!") {
        therapistLine = randomElement(exclamationResponses);
    } else {
        therapistLine = createQuestion(patientLine);
    }

    // Still no good response, so use a basic response.
    if (therapistLine == null) {
        therapistLine = randomElement(genericResponses);
    }
```

Save and refresh the page. Try sentences that have the **povSwitch** properties to see if it properly converts them. Try putting a period at the end and make sure it's stripped out. Try sentences with "you" or "your" to see if you get the generic responses.

If you get stuck, you can check my solution at: http://sdkbridge.com/prog1/Exercise15dAnswers.pdf.

If You Want to Do More

Are you having fun? There's more you can do to make the therapist bot better. Here are some ideas. Feel free to do them, or to move on to the next exercise.

Better Replacement

The string **replace** method only replaces the first occurrence it finds. You should really replace all of them. This requires a "regular expression" object. See the code below on how to do this, with new code in bold.

Note: The "g" in the code below means "global", which will replace all of the strings it finds. The **new** keyword means creating a new object – it's part of object oriented programming.

```
var found = false;
for (var property in povSwitches) {
    if (povSwitches.hasOwnProperty(property)) {
        var modifiedProperty = " " + property + " ";
        var propRegEx = new RegExp(modifiedProperty,"g");
        if (modifiedLine.indexOf(modifiedProperty) != -1) {
            modifiedLine = modifiedLine.replace(propRegEx,
                " " + povSwitches[property] + " ");
            found = true;
        }
    }
}
```

Respond to More Keywords

Look for particular words and come up with a list of responses for sentences with them. For example, you could look for the word “boss” and ask things like, “How long have you worked for your boss?” Other words might be: husband, wife, boyfriend, girlfriend, mother, father, sister, brother, relationship, death, sex, etc.

Share

Share your therapist bot with your friends and have them try it out. You can always email it to them and have them double-click it to open it in a browser. Better yet, if you have access to a server, put it up there and give them the URL.