Web and AI

The Web is the primary way we use computers, communicate, and access new forms of Artificial Intelligence (AI). You will build a modern version of a classic AI web application, and consider what it means for a computer to be “intelligent”.

Table of Contents

Overview

In our previous module, we introduced network communication in two forms - USB and WiFi. WiFi (wireless fidelity) is accepted shorthand for wireless networking, which you commonly do nowadays on your personal computers, tablets, and phones - but there are also landline connections, like those you see with the white cables in our classroom.

Our interest in this module is more at a higher level of networking, or specifically HyperText Transfer Protocol (http), the network application protocol of the Web. When you use a web browser (and almost all apps you use on your phones), your device speaks to other computers over http. So what does http do and why is it important? We will cover that in this module.

We will also discuss how web browsers (and again, many of your phone apps) work - meaning how do they get, render (show), and interact with web-based content? The most common format here is called HTML (HyperText Markup Language), which really is not all that different from how this document is stored.

Almost since the start of computer programming, we have wondered if computers would ever become as intelligent as humans. One of the earliest computer programs ever written (and traditionally the 1st one you write in any new programming language), is "Hello World!" - which suggests a computer program is intelligent and speaking with you. Today in Generative AI programs like ChatGPT or Google Gemini, we have incredibly powerful and seemingly highly intelligent computer programs. But are these programs "intelligent", or merely give the appearance of intelligence? How do we even answer this question?

In 1950, a famous computer scientist Alan Turing (movie: "The Imitation Game") posed "the Turing Test" - where a person is asked to interact with a computer program and a real person through a computer interface, and if the person cannot distinguish which responses come from the computer and which come from the human, then the computer program is "intelligent". This ability to imitate a human is the origin of the movie title!

In this module, we will iteratively build a modern version of the classic Eliza program, which was one of the first attempts at passing the Turing Test.

Prerequisites: A web browser like Firefox, Chrome, Safari, or Edge. NodeJS and npm. Any code editor will work fine.

Iteration 1: a browser-based Eliza application

In this first version, we will focus on building a version of Eliza in your web browser without using networking (http). The code for this iteration is in the “given” folder.

Before jumping into the technical details, let’s discuss how Eliza is “intelligent”. Eliza acts like a therapist - it attempts to get you talking more than it talks, and by prompting you to keep taling and drive the conversation, you soon thinking you are talking with a real, intelligent person.

In this first client-only version of Eliza, we use HTML, Javascript, and CSS (what is called the client-side "stack") to directly create an application in a Web browser. This example shows the basics of how a browser uses this stack to present, style, and manipulate content without a web server.

Step 1: Run the elizabot.html in your browser

Using a web browser, go to File menu → Open (or type “Ctrl-O”), and navigate to the file Web → given → client → elizabot.html. You should see something like the following:

Play with this program a bit by typing in short phrases. How does Eliza respond? What aspects of Eliza’s responses seem human? How would you improve it?

Embedded in the code are several comments explaining how it works, and some challenges for you to enhance the application. But first let’s talk at a high level about how this application works:

1. You loaded an HTML page named elizabot.html
2. Web browsers understand (“parse”) HTML and know how to do 3 things:
    a. Display markup (the things in tags like <h3>...</h3> visually on the screen
    b. Apply any customizations to that visual display via CSS (more below)
    c. Use code, in a language called Javascript, to facilitate interaction with this web page.
3. Each time you type something in, the code (function processReply) sees if any of the words you typed in match what is in vocab.js. If one does, then a resulting phrase and follow-up question are displayed. If not, a generic response is shown.
4. The display is updated by adding what you typed and the responses to the page in the addMessage function.

So how can we make this application better and more intelligent. Embedded in the comments in the code are some challenges. You can use the “Find” (Ctrl-F) feature in Notepad++ and search for “CHALLENGE” to find them, but they are also listed here:

CHALLENGE 1: (line 7 of elizabot.html) - make the program fancier in its display by uncommenting line 8 that includes eliza.css. Then reload elizabot.html in your browser.

CHALLENGE 2: (line 10 of elizabot.html) - CSS stands for “Cascading StyleSheets”, with the key word here being “style”. Open eliza.css in your editor. It looks kinda strange, here is an example:

```
#chat-container {
   width: 400px;
   background: white;
   border-radius: 12px;
   box-shadow: 0 4px 10px rgba(0,0,0,0.1);
   overflow: hidden;
 }
```

What does this mean and how can you modify it? Let’s break it down:

- #chat-container is a *selector* that matches a tag with a id chat-container in the html file (see line 26 of elizabot.html). So the stuff inside {..} applies to that.
- You can kind of figure out what each line is by looking at the label and playing around with it. For example, *width* is the width of the chatbox, currently set at 400 pixels. Change it - set it to 600 pixels (or whatever you like) and reload the page.

CHALLENGES 3-5: How intelligent Eliza seems is based on the size of its vocabulary. This is not all that unusual, you have spent your school years learning more complex vocabulary in your English classes, and if you have taken a foreign language, you spend a significant time learning words in the new language to build your vocabulary.

Challenge 3, in elizabot.html, asks you to extend what Eliza might say if it does not understand what the user types in. Right now it can only choose between 2 responses (line 87), and gives the same follow-up question each time. Well, if we have a computer program that gives the same responses every time it does not understand something, then we’d probably figure out real quick that it is not intelligent. Challenge 3 asks you to expand the set of responses Eliza gives when it reads something it does not understand.

How do we know what Eliza does and does not understand? Our Eliza's vocabulary is in vocab.js, and consists of 10 groups of entries. Here is the 1st one:

```
{
       key: ["love", "romance", "heart"],
       phrase: ["Love is a curious algorithm, isn't it?", "Even AIs need a
little affection... sometimes."],
       question: [
           "What does love mean to you?",
           "Can machines understand romance?",
           "Is love more logic or emotion?",
           "Do you believe in soulmates?",
           "What makes someone fall in love?"
       ]
   }
```

What Eliza does is see if the user typed in one of the keywords (love, romance, or heart), and if so will randomly return one of the phrases and one of the follow-up questions. By randomly returning these responses to a keyword, Eliza gives the appearance of intelligence.

In the given code, there are 30 total keywords across 10 entries. If the user fails to type in one of those words, then Eliza does not understand (Challenge 3 above). This is a very small vocabulary, so both Challenge 4 and 5 ask you to grow Eliza's vocabulary.

Challenge 4 asks you to extend the vocabulary by directly editing vocab.js. You can add keywords, phrases, and/or questions to the existing 10 entries. You can also add entirely new entries. In either case, pay close attention to the format - this is computer code, so you have to follow the pattern exactly, otherwise you may get an error.

How many entries and how many keywords do you think you would need before Eliza seems intelligent? Probably a lot right? A typical 12 year-old has about 50,000 nouns in her/his vocabulary. We are a long way from that, and manually creating that vocabulary would take a long long time (though some projects, like WordNet, have tried!). Challenge 5 asks you to use ChatGPT (or Gemini, or any Generative AI) to create a large vocabulary. What prompt do you need to give to make this happen?

Iteration 1 reflection:

1. What are the parts of the Web stack and what do they do?
2. What does it mean for a computer program to be intelligent? What aspects of a computer program can be made to appear intelligent?

Iteration 2: a client-server Eliza application

In iteration 1, we just loaded an HTML file directly into a browser using File → Open. That isn't what you usually do - you usually type in a URL (Universal Resource Locator) into an address bar on your browser, something like "https://www.asu.edu". In this iteration we will discuss why you do this, how it works, and convert our Eliza to use this way of doing things. This will help us make our Eliza appear more intelligent.

In this second version of Eliza, we start as before trying to use only HTML, Javascript, and CSS, but find we run into security limitations working directly with local files. To get around this, we introduce a server. A server "serves" content and features to "clients", in this case web browsers. A client and a server are separate programs, so to talk to each other, we use a network, just like we did with the Arduino WiFi.

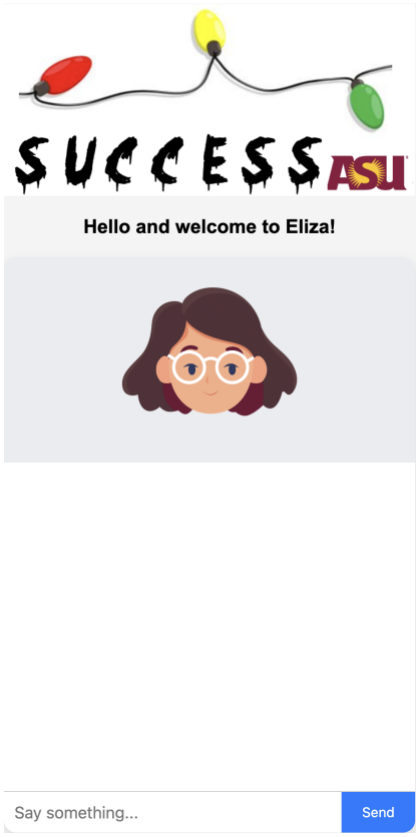
When we talk as humans, we use natural language, like English or French. When we do so, there are rules we must follow so others can understand us. For example, we structure our sentences properly, ("grammar") and the way we do this depends on the language we use. For example, in English I might say "I like that red dress", but in French I'd say "J'aime cette robe rouge". What is different? In French the adjective "rouge" (red) comes after "robe" (dress), the opposite of English. These rules, which you learned in school and use every day, are called a "protocol".

Computers use protocols to set the rules for their communication as well. You've probably typed it in to your computer or phone before - something that starts with "http" (or "https"). The "p" stands for "protocol", so "http" (HyperText Transfer Protocol) is the set of rules for a web browser to communicate with a web server. In this example we will introduce a web server that serves HTML and other content to your web browser, and also allows us to send information to the server.

We also extend our basic Eliza to give a more convincing presentation of intelligence. In this case, we will add a talking avatar, and speech, to Eliza to further try and trick our human user into thinking Eliza is intelligent.

To start, do what you did before: File→Open→Web→given2→client→ elizabot.html. You will see what you saw at the end of Iteration 1. But… this code tries to load an avatar (see lines 14, 26, 39-49), but there is no avatar on the screen - why? There is a security error - open your Developer Tools, and you will see a Cross-Origin Request Blocked (CORS) error. What is this? It is a fancy way for the browser to tell you it does not want to load the avatar as it thinks it is not secure. All modern browsers do this to protect you from hackers trying to insert dangerous content into the web pages you use. But we aren't trying to do anything dangerous, so how do we solve this problem?

To run this example, you will need to open a terminal window and cd to the server subdirectory and run "node server.js". Then you can go to your web browser and enter "http://localhost:8008/elizabot.html" to get started. This is the same file you tried to use File→Open on before, but now you are using a network connection (the “http” part) to load it, and the browser will trust what you do now, so the avatar gets loaded.



Let’s talk about the AI part. Again, what Eliza is trying to do is give the appearance of being an intelligent human. One way we did this before was to expand the vocabulary of Eliza. In this iteration, we will enhance the intelligent appearance by inserting an animated avatar who appears to be talking, and to apply the new Speech API (Application Programming Interface) in web browsers to have our program talk!

The avatars we will use were created by the lottie platform. We have downloaded 10 free lottie avatars and put them in the lotties subfolder. If you have an account or want to create one, you can download additional ones or even create your own! The initial avatar, shown in the picture, is girl.json. You can change this to any of the others (see line 50 in server.js) and restart the server (to do so, hit Ctrl-C in the terminal where you started node.js, and simply start it again).

There are a set of challenges that involves you adding the avatar, animating an avatar, and adding speech. Solutions are provided if needed.

CHALLENGE 1: (server.js) Make it so you can change the lottie avatar you use. There is a challeng 1b where code is given to randomly select the avatar
CHALLENGE 2: (elizabot2.html) Start and stop the avatar talking animation
CHALLENGE 3: (elizabot3.html) Improve the AI when the user types in something that does not match.
CHALLENGE 4: (elizabot4.html) Use the speech API to make Eliza talk!

NOTE: if “node server.js” gives you an error please call one of us over to help you!

Iteration 2 reflection:

1. What is a network protocol and why is it important?
2. Does humanizing a computer program make it appear intelligent? In what ways do hackers humanize their approaches to try and trick you?

Iteration 3: Making Eliza truly intelligent!

So far we have been focused mostly on making Eliza appear intelligent, instead of “being intelligent”. We will try and change this by using Generative AI to help Eliza - but is this intelligence or merely a more complex attempt at making her appear intelligent?

In our 3rd and final version of Eliza, we will use Generative AI (ChatGPT) to generate Eliza's responses instead of our vocab.js. Our vocabulary is large, but it is finite, meaning that if the user does not type in one of our keywords, then Eliza will not understand what he user says. Further, the approach can get repetitive if the same keyword is entered many times, and being keyword-based, sometimes the responses do not seem related to what the user typed in. Can ChatGPT do a better job?

In this example, our server will call ChatGPT's API (Application Programming Interface) to present the user's input as a "prompt" to ChatGPT, and get it's response and follow-up question.

Do you think this version will be better or worse (more or less intelligent) than our vocabulary version? As our last challenge, we will have our server randomly choose between the vocabulary and ChatGPT for responses, and the human user has to figure out which one gave the response!