**Activity: Implement your own Eliza!**

Since the early days of computing, humans wondered if computers could be made "intelligent". The father of modern day computinh Alan Turing proposed “[the Turing test](https://en.wikipedia.org/wiki/Turing_test)”, or the argument if a computer could communicate like a human, then it possessed human intelligence. To demonstrate this philosophical approach, a [computer program named ELIZA](http://psych.fullerton.edu/mbirnbaum/psych101/Eliza.htm) was created to emulate a [Rogerian psychological therapist](https://www.goodtherapy.org/learn-about-therapy/types/person-centered) to a person by responding to queries. In essence, [Eliza](https://en.wikipedia.org/wiki/ELIZA) was Siri (or Alexa if you choose) before Siri was! For this activity I want you to implement your own Eliza!

**Starting point:**

On your class website, there is a file that has HTML markup with embedded Javascript code (inside the <script> tags). Navigate to the [TryitEditor on the w3schools website](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_basic) and copy and paste the contents of the file in the left-hand browser pane, removing what is already there. Click the green “Run” button and see the results on the right-side. Now talk to Eliza!

**What is in the HTML code on the left side?**

*First, let’s look at what is in the HTML (ignore what is in the <script> tag at the bottom). Notice:*

* Each markup element has a begin tag in angle brackets, some content, and a closing tag in angle brackets with a slash. Fr example <head>My Eliza</head> at the top is the HTML document “header”.
* Notice many tags are nested; for example:

<div id="name">

<form onsubmit="showChat(); return false;">

Hi. What is your name? <br>

<input id="username" type="text" name="uname" size=50 /><br>

</form>

</div>

* Note the nesting is *well-formed*; that is 1) each begin tag has a closing tag, and 2) a tag is not closed “out of order”.

*(but wait, what about the <br> tag on the right???)*

* In web programming, this well-formed structure is parsed by a browser to create an internal representation that looks like an outline. Outlines like this can be drawn as a tree. This data structure is common in computer programming of all kinds. In your web browser, this tree is called the Document Object Model (DOM). See the last page.
* Look at the HTML markup on the left and the display on the right. You should be able to pick out which elements (tags) go with which things you see on the screen. Do all of the elements show visually on the screen?
* You may notice some things in the content, like “Hi. What is your name?”, no longer appear on the screen. Why is that?

**What is in the Javascript code on the left side?**

*Wait! Before we can understand the code, we have to think about the process, or algorithm, used to make Eliza work.* An algorithm is defined as “a step-by-step process to solve a problem in a finite number of steps”. Jot down what steps you see when you run the Eliza program. There are no right-or-wrong answers, but when you are ready you can sneak a peek at my breakdown on the next page.

*Now, look at the code inside the <script> tag at the bottom of the left pane. This may look confusing at first but let’s break it down:*

* At the top you see “var dict = *a whole long set of stuff*”. This is a *variable* named *dict* that stores our dictionary in an *array*. The array has in it *objects*, comprised of “key” and “phrase” arrays. An array is a fancy term for a *list*, and an *object* is a fancy term here for a structure that can hold multiple things, like a bag of stuff. This variable is our dictionary of keywords and pseudo-intelligent phrases (steps 3a and 3b).
* After that section, you see some more variables (ignore for now) and then some *functions*. You are used to math functions of the form *f(x) 🡪 y*, or to state plainly, a mapping of input values to a single output value. Javascript functions are somewhat similar; they take zero or more inputs and produce an output. The input(s), if they exist, are in () after the function name (see “processReply”) and the outputs are in *return* statements. But Javascript functions are unlike math functions in that they can also get input by using variables (those “var” statements, like *var dict*), and they don’t always have to return something.
* *There are a lot of functions, do I just read them sequentially?* In some types of programs, yes you can do that. But in most programs, you need to read them according to the steps of the algorithm. If you haven’t yet, review the algorithm page.
  + *What are you looking for?* In the steps, you are looking for *events* – OK, so what are those?
    - *Start*: after the browser loads the HTML, it is ready to start executing code.
    - *User input*. Eliza asks for the user’s name (step 1) and asks the user to talk to her (Step 2)
    - *Passing of time*: Eliza displays an alert after a while (step 4)
    - *Sequencing*: The completion of something, in this case Eliza completing a response (Step 5)
  + In programming, you do something when an event happens; namely, you execute one or more functions!
    - *User input (step 1)*: The event is onsubmit and the function is showChat().
    - *User input (step 2)*: The event again is onsubmit and the function is msgFromUser().
    - *Passing of time (step 4)*: Find timer = setTimeout(timeOut, 20000); in the code. This line is like an alarm clock; it tells your browser “hey, in 20 seconds wake up and invoke function timeOut”.
    - *Sequencing (multiple steps)*: Typically by having one function invoke another. For example, *processReply* is invoked from *msgFromUser*.
* *How do I change what is displayed on the screen*? See the discussion above on the DOM, and the image on the last page. This structure is held in the computer’s memory, and in the Javascript the special variable *document* is automatically assigned to it.
  + You can find parts of the content using document, such as the *document.getElementById* lines.
  + You can manipulate elements once you find them, and the browser automatically re-displays them. Elements are *objects*; they have properties you can assign. *innerHTML* is the property representing the content, you can assign it to whatever you want!

**Activity 1: Modify the HTML**

Let’s personalize your Eliza! Try some of the following:

1. Change the greeting when the page first runs.
2. Add a new paragraph using a <p>…</p> tag to have Eliza give a longer introduction to herself. Consider using [various formatting mechanisms](https://www.w3schools.com/html/html_formatting.asp) like you would in Google Docs.
3. What does Eliza look like? Find a picture on the Internet and insert it in your Eliza welcome message using the [image tag](https://www.w3schools.com/html/html_images.asp) <img>.

**Activity 2: Expand Eliza’s vocabulary and responses**

Modify the *dict* variable. You will need to maintain the proper structure o pay attention to what you are typing!

1. For each existing object in the dictionary, add more keywords and more phrases
2. Add more objects - additional keyword and phrase arrays (pairings)
3. Add more alert messages to the timeout (*timeoutMessages* in the *timeOut* function)

Another software engineering principle is *incremental problem solving*; each of the numbered items under each Activity are independent, so “code a little test a little” by getting each one done and saving it (the save icon is in the middle). In this case saving will give you a bookmarkable URL that you can save in your browser bookmarks. Then tackle the next step!

**EXTRA CREDIT Activity (for the ambitious): Add new features to Eliza!**

See if you can do the following:

1. If the user inputs just the word “quit” when talking to Eliza, *reset the program* to the beginning (without re-loading the webpage).
2. *Change to an* ASU SUCCESS *theme*! Eliza has some pretty boring (and negative) categories. Make an ASU-inspired theme by changing Eliza to (these are just examples):
   1. Vecna – answers all things about the Upside Down</LI>
   2. Your Dad – seems to be all-knowing about a significant number of things</LI>
   3. Your Mom – expresses a lot of empathy for anything you say</LI>
   4. Your Dog - just what is your puppy thinking?</LI>
   5. A favorite movie character - Mine is Kevin from Despicable Me!</LI>

Or whatever you want! Put images, phrasings, or anything in-character (be nice!). You can even make it a secret camper and we will see if others can guess who it is!

1. *Make Eliza more intelligent*. Can you think of ways to make Eliza more intelligent? Yes Activity 2 does that in some ways, but what else can you think of? Keep in mind the Turing test – the idea is to trick someone into believing your computer program is actually a human! What kind of human traits can you inject that might trick the user into thinking Eliza is ALIVE!

**Debrief:**

This activity includes some very real things about computer programming. Eliza is a lab exercise I assign senior-level college students at ASU. Some concepts included here (they may not make sense but talk with someone you know in tech and see what they say):

* *Single-page Application (SPA)*: Eliza is an example of a SPA. In an SPA, you load one web file and interact with it. This is different than other types of webpages where you click on hyperlinks and go to a new page (you see the address in the location bar change). In industry, there are specialized technologies to assist with creating SPAs. Some mobile apps are really SPAs! Note that we do not have a server per se in this exercise.
* *The HTML/Javascript/CSS stack*. Computer programs have grown in complexity in many ways. Nowadays it is not unusual to have one “program” written in many languages and technologies (the “stack”). In this exercise you have used HTML (markup for page structure and content) and Javascript (to exhibit *behaviors*). CSS, which we did not cover, focuses on formatting rendering to provide aesthetically pleasing (pretty) and responsive (the same web file can be displayed on different devices) web pages.
* *Event*-*driven programming*. Traditionally computer programs are learned as a sequential set of instructions, where one line of code is executed then the next until the sequence of instructions is done. While there is a little bit of that here (like inside functions), you are using an event-driven model where code is executed in reaction to something else that has happened. A real-world example of this might be a Smart Home; when a smart home sensor like a motion detector detects motion, it can control other things like opening door locks, turning on lights, or adjusting the room temperature for comfort (check out IFTTT).
* *Functional* *and* *object-oriented* *programming*. There are lots and lots of programming languages out there, with some good reasons for that (and some not so good). Programming languages are usually categorized by paradigm: *functional*, *object-oriented, declarative*, etc. Javascript is a bit of a hybrid; it is both object-oriented and functional, as you saw by working with certain object structures, like the DOM (document) and by having functions manipulate the DOM and handle events. This glosses over a fair amount of the distinction (ASU has 2 courses on this topic CS majors take), but you have been exposed a little to both.
* *Artificial Intelligence (AI)*: Much of AI nowadays is data-driven; using powerful mathematical algorithms to identify patterns in data and make decisions based on them. Eliza is what is called GOFAI (Good old-fashioned AI), where models of human thinking are directly coded into programs. Further, Eliza and the Turing test started a still-ongoing philosophical and ethical debate about what constitutes intelligence; whether machines can become intelligent (or *conscious*, or just *autonomous*). You see this played in popular media with movies like The Terminator, The Matrix, and A.I. amongst others.

**ALGORITHM**

**Steps to Eliza:**

1. Greet the user and ask for her/his name.
2. After the user types in a name, prompt the user to talk to Eliza.
3. When the user types in a question, respond in an intelligent way.
4. If the user does not interact with the program after a while, display an alert with the user’s name calling attention to Eliza.
5. After each completed interaction with the user, copy it on the page.

OK, simple enough. But steps 3 and 4 are not that precise. How do I make Eliza “respond in an intelligent way”? And what does “after a while” mean exactly. For a computer, we have to state our instructions precisely and completely, otherwise we are leaving it open to interpretation by the programmer (like if your teacher gives incomplete or confusing instructions, you have to decide what to do). This is an area of computer science / software engineering called *software requirements specifications*.So let’s state these more precisely:

Revised step 3 (decomposition):

1. When the user types in a question, respond in an intelligent way
   1. Eliza picks out specific important words that a user types in.
   2. Eliza uses the word(s) to determine a pseudo-intelligent response from a set of pre-defined responses.
   3. Eliza "remembers" previous answers the user has presented and applies some randomness to avoid deterministic and/or repeated responses.

Revised step 4 (adds precision):

1. If the user has not typed in a response in 20 seconds, display an alert calling attention back to Eliza. Provide at least 3 different alert messages and randomly select one each time the alert is given.

*When you start with a problem statement and design an algorithm, this is called forward engineering. When you start with a solution and figure out what it does so you can fix/enhance/evolve the solution, that is called reverse engineering.*

Now that we have reverse engineered Eliza’s process, we can inspect the code to understand how it does each of these steps. Return to the previous page for the explanation.

|  |  |
| --- | --- |
|  |  |