WORDPLAY ROBOT

by Stevan Živadinović

For this project you will need: **a computer**—a tablet or phone will do in a pinch, but keyboards are best for coding—**and fearlessness in the face of code.**

No new software installation is required, we will be coding in the browser using **JavaScript** language and help from two libraries: **p5*js** (p5js.org) and **RiTa** (rednoise.org/rita). The first library will provide structure, and the second natural language manipulation controls.

I have a basic boilerplate ready for you to modify at **bit.ly/NasherRobot**

To see your code sketch in action, hit the big **PLAY** button. Click on the colorful square with the generated text on the right to bake new phrases.

To play around in the editor you don't need an account with p5js.org, but to **save and share** your wordplay robot you will need to create one. Account creation is in the top right corner of the editor interface. It can be done with one click, by linking to an existing Google account.

Any text that follows `//` is a comment. I have left notes throughout the code about what each bit does. The action parts of the code, the stuff that we'll be modifying looks like this:

```
rules = {
  ...
};
```

Inside it, we will put the grammar rules that the rest of the code will convert into phrases and display on the screen with each click of the madlib square. Let's start by emptying the existing rules and rebuilding them from the ground up.

Our wordplay will start with the `start` key corresponding to our first string of rules RiTa will process:

```
rules = {
  "start": "beep | bop | boop"
};
```

This simple grammar rule will return *"beep", "bop"* or *"boop"* as our random phrase. The vertical line character `|` is known as the pipe delimiter and lives above the enter/return button.

Ok, so far so cool, but how do we expand on it and madlib it up? We can define special variable keywords that will stand in for different words or phrases:

```
rules = {
  "start": "$bobity $bob",
  "bobity": "beepity | bopity | boopity",
  "bob": "beep | bop | boop"
};
```

Note the `$` before the `$bobity` and `$bob` in our new `start` phrase. Those variables will be replaced with one of the `bobity` and `bob` options. **Mind where the quotes and commas and colons and pipe delimiters go!** If they are

missing or in a weird place, the console will pop up with some confusing lingo about syntaxErrors which can sometimes misdiagnose what the problem is and where the problem lies. Just carefully look at your punctuation first.

So far so good, but there is no reason we cannot add a bit of delightful chaos into our madlibs. Let's try this:

```
rules = {
  "start": "$bobity $bob",
  "bobity": "beepity | bopity | boopity | $start",
  "bob": "beep | bop | boop | $start"
};
```

Now a quarter of our generated phrases should double up on the start phrase as a **bobity**. Let's make it more chaotic by putting the finger on the scales of our potentially infinite return to start:

```
rules = {
  "start": "$bobity $bob",
  "bobity": "beepity | bopity | boopity | $start",
  "bob": "beep | bop | boop | $start [4]"
};
```

Now our loop is **[4]** times more likely to come up on any random roll than the other **bob**s. Don't get too carried away with large numbers on that infinite recursion as you can cause your browser to use up all of your computer's working memory and grind to a halt. There is still a chance that will happen with a **[4]** but it is a distant one, and what fun is a program without built-in crash potential?

Let's do one more thing. Let's get RiTa to automatically pluralize stuff for us:

```
rules = {
  "start": "$bobity $bob | $bobity.pluralize()",
  "bobity": "beepity | bopity | boopity | $start",
  "bob": "beep | bop | boop | $start [4]"
};
```

Continuing our ethos of playful chaos, this code might do wacky loops, pluralizing things that have already been pluralized, creating funky multiple pluralseses!

While it is delightful to take this scat game further with more random syllables and new inception-y patterns, this little sketch can be used to generate whatever robotic wordplay your heart desires. If this project has piqued your interest, check out the cheat sheet I left in the code comments for more ways to interact with RiTa. RiTa website linked above contains more examples of fun things to do with text generation.