Introduction:

Programming by Example is a powerful approach to programming that allows code to be generated or repaired by manipulating input/output examples. Live Programming is a method of programming that allows programmers to see a response to their code in real time. When combined, Live Programming by Example (LPbE) allows a user to modify input/output examples, and see the code change in real time. In practice, this can be used for many purposes, but perhaps the most practical is in assisting in education. With Programming by Example, a user with limited programming experience can create functions by simply demonstrating the desired behavior. In a live environment, that same user can see how the code is built so that they can repeat it again in the future. Our goal is to create an editor that uses LPbE as a programming aid for novice and intermediate programmers, with an easy to use interface that provides them with the feedback they need to complete their code.

Programming by Example has been applied with great success in applications like Flash Fill for Microsoft Excel\(^1\). However, here the goal is not to replace programming knowledge, but to supplement and teach it. It is estimated that half of the time a programmer spends is spent understanding how legacy and other previously written

code works alone. By allowing modifications to the output to change the source code, programmers can better see and understand how the code they are working on works, and learn for the future. This can be applied to teaching people new syntax, as well as assisting locate changes in major variables, and aid modification of helper functions. Instead of telling them how to be coding, we show them how the code works, using examples and being hands on.

Design:

Currently, we are developing a plug-in for the Atom IDE that allows for users to use LPbE with JavaScript. The main issue with this implementation is that it is relatively inaccessible to novice programmers. A solution to this would be to improve the front-end of the editor to make it more interactive and have a friendlier user interface. Currently, the interface uses two columns of text editors. Programmers write code in the left text editor, and write examples in the right text editor. Writing examples in this manner Instead, we can add CSS and JavaScript elements to better guide the programmer in using the platform.

For example: Programming by Example relies on desired outputs to produce the needed code. LPbE accomplishes this by taking the existing code, and creating informative examples of outputs that help describe the function. The way it currently works, it takes a function on the first editor, and displays sample outputs on the second editor. With either editor has its contents changed, the other will adapt with it. Users can

---

also append their own outputs to the second editor to add changes to the code. If a user might want one output to not change at all once the code is changed, they would need to lock some of those outputs from dynamically changing. To solve this, we could use checkboxes to mark a dynamic versus a locked output as seen in the image below.

Outputs (Check to Lock):
- $x + y = 6$
- $x * y = 18$
- $x \% 2 = 0$

New output: [Submit]

The interface will need to be fleshed out to allow for more simple user interaction like this.

Given that Atom is an IDE meant to be “hacked” and is built in Electron, a JavaScript based platform\(^3\), it should be easy to create a functional frontend. The goal is to make the users comfortable using it so that it is not a learning curve to start programming by example, but instead a way to enhance their programming abilities.

**User Study:**

To supplement the editor, we will conduct a user study on a variety of different users with differing experience with coding and JavaScript to gauge the effectiveness of LPbE for educational usage. We will interview them for their feedback and experience to see the viability and effectiveness of the product. Figuring out the ideal use case of this

technology is key to where LPbE is useful to introduce and accelerate programing abilities.

We want to learn how people interact and use the software against a traditional workflow and see if this is something that people would actually use and see a benefit from using. It is important to answer questions such as the editor providing the desired output when used, or if any manual editing is needed. Also figuring out how often people refer back to examples when coding manually will help us better understand their workflow. To do this, we will be using A/B testing on users to see how well a traditional environment works as a control against LPbE.

In order to receive unbiased information, we must ensure that users are evaluated before and after being exposed to LPbE. Everybody has a different scale of how they define their experience with programming, so we must judge the difference in ability, not the pure ability to reproduce. Asking for their prior experience with the language is important to know where LPbE may be useful, as the syntax is often the biggest hurdle in an experienced programmer learning a new language. Ideally, we will take people with some basic programming knowledge, but not in one the languages that we offer (currently JavaScript and Haskell), so that we can test this theory.

Before they start programming, we will give them a quick tutorial of the platform so they can quickly understand how it works and how to use it. This can be either in person or by a video. It should explain to them how to use examples and what programming by example does.
For the experiment, we will randomly determine which of two different tasks they will be solving, and which environment between traditional and LPbE they will use first. While programming, we will frequently log their code and examples. Notably, when an example as added, removed or modified, and when the code is changed to modify an output dynamically, otherwise every few seconds. This will allow us to review their behavior and interaction with LPbE to look for patterns and habits.

After the experiment, we will have a quick interview to see their response to using it. On a numeric scale, we would like them to rate their experience using the editor: how helpful was programming by example, and how helpful were live updating examples. We would also ask them for a short explanation as to why they gave the ratings they did, and what would make the program and live updating examples more helpful or accessible for them.

The important distinction to understand is where LPbE might be useful and have a place in the ecosystem. Is it best for novice programmers to learn, experienced developers, to debug legacy code or some other area? With this information, we can also judge where to improve and expand the software based on what people want to be able to do with it. Is it a seamless addition to the programming workflow, or is there a learning curve that makes them less efficient for a time? What can we add to LPbE to make it as easy to introduce as possible?
Deliverables:

1. A functional and elegant front end design for the Atom IDE extension that will allow easy interaction with the user.
   a. Examples of this include an output locking mechanism, additions, and deletions.

2. A specification for an overall interface design that will detail how the backend should interact with the created frontend.
   a. This should include standards for how the code interface and the output interface should react with given changes.

3. A case study of multiple different users experience with the plugin that will answer:
   a. How users interact with LPbE.
   b. How helpful they find LPbE to be for their workflow.
   c. Which improvements might help LPbE be more accessible.
   d. Ideal environments in which LPbE is most useful.

4. A report summarizing said case study to supplement any academic papers on the research that may be written.