My core research question is: *How can we help a more diverse range of young people become fluent with computational technologies?*

This question revolves around two core ideas: *fluency* and *diversity.*

- **Fluency.** I emphasize fluency to highlight the importance of learning to design, create, and express yourself with computational technologies. Learning to express yourself with computational technologies is valuable in itself, providing new ways to communicate, present, and share ideas. At the same time, it also provides a particularly effective way to learn important computational concepts, since the concepts are learned in a meaningful context. This emphasis on fluency leads naturally to a focus on programming. Just as learning to write is a central component of traditional fluency, learning to code is a central component of computational fluency.

- **Diversity.** I emphasize diversity partly to highlight the importance of opening opportunities for a broader range of people to become involved in computing as a profession— but, even more important, because computational fluency (just like traditional fluency) is becoming essential for anyone who wants to participate fully and actively in today’s society, regardless of their profession. That means that we need to make pathways towards computational fluency accessible and appealing to people from all different backgrounds, genders, interests, and learning styles.

To address these issues, my research group at the MIT Media Lab has been developing and studying the Scratch programming environment and online community (http://scratch.mit.edu). Since its public launch in 2007, Scratch has been used by millions of young people around the world, mostly ages 8 and up. Scratch builds on a long tradition of programming languages (such as Logo, Etoys, AgentSheets, and Alice) designed especially for children and teens. Guided by the dual goals of fluency and diversity, we have designed Scratch to support activities and experiences that are more tinkerable, more meaningful, and more social.
More Tinkerable
For many years, my research group has worked closely with the LEGO Company, helping develop LEGO Mindstorms and other robotics kits. We have always been intrigued and inspired by the way children play and build with Lego bricks. Given a box full of them, children will immediately start tinkering. They’ll snap together a few bricks, and the emerging structure will give them new ideas. As they play and build with LEGO bricks, plans and goals evolve organically, along with the structures and stories.

We wanted the process of programming in Scratch to have a similar feel. The Scratch grammar is based on a collection of graphical “programming blocks” that children snap together to create programs. As with LEGO bricks, connectors on the blocks suggest how they should be put together. Children can start by tinkering with the bricks, snapping them together in different sequences and combinations to see what happens. Scratch blocks are shaped to fit together only in ways that make syntactic sense. There is none of the obscure syntax of traditional programming languages.

The name “Scratch” itself highlights the idea of tinkering, as it comes from the scratching technique used by hip-hop disc jockeys, who tinker with music by spinning vinyl records back and forth with their hands, mixing music clips together in creative ways. In Scratch programming, the activity is similar, mixing graphics, animations, photos, music, and sound.

The scripting area in the Scratch interface is intended to be used like a physical desktop. You can even leave extra blocks or stacks lying around in case you need them later. The implied message is that it’s OK to be a little messy and experimental. Most programming languages (and computer science courses) privilege top-down planning over bottom-up tinkering. With Scratch, we want tinkerers to feel just as comfortable as planners.
More Meaningful
We know that people learn best, and enjoy most, when they are working on personally meaningful projects. So in developing Scratch, we put a high priority on connecting with youth interests – supporting a diversity of project genres (stories, games, animations, simulations), so that people with widely varying interests can all work on projects they care about. We also put a high priority on personalization – making it easy for people to personalize their Scratch projects by importing photos and music clips, recording voices, and creating graphics.

We have been amazed by the diversity of projects that appear on the Scratch website, reflecting the wide-ranging interests of community members. As expected, there are lots of games on the site, ranging from painstakingly recreated versions of favorite video games (such as Donkey Kong) to totally original games. But there are many other genres, too. Some Scratch projects document life experiences (such as a family vacation in Florida), while others document imaginary wished-for experiences (such as a trip to meet other Scratchers). Some Scratch projects (such as birthday cards and messages of appreciation) are intended to cultivate relationships. Others are designed to raise awareness on social issues (such as global warming and animal abuse).

As Scratchers work on personally meaningful projects, we find they are ready and eager to learn important mathematical and computational concepts related to their projects. Consider Rafael, a 13-year-old boy who used Scratch to program an interactive game in his after-school center. He created the graphics and basic actions for the game but didn’t know how to keep score. When I visited the center, Rafael asked me for help. I showed Rafael how to create a variable in Scratch,
and Rafael immediately saw how he could use it for keeping score. He began playing with the blocks for incrementing variables, then reached out and shook my hand, saying “Thank you, thank you, thank you.” I wondered: How many eighth-grade algebra teachers get thanked by their students for teaching them about variables?

**More Social**

Our development of the Scratch programming language has been tightly coupled with development of the Scratch website and online community. To support our goals of fluency and diversity, we feel that the language needs to be linked to a community where people can explore, support, discuss, and build upon one another’s work.

Since the launch of Scratch in 2007, about 4 million projects have been shared on the Scratch website. For many Scratchers, the opportunity to put their projects in front of a large audience – and receive feedback and advice from other Scratchers – is strong motivation. The large library of projects on the website also serves as inspiration. By exploring projects there, Scratchers get ideas for new projects and learn new programming techniques. Marvin Minsky once noted that the programming language Logo had a great grammar but not much literature. Whereas young writers are often inspired by reading great works of literature, there was no analogous library of great Logo projects to inspire young programmers. The Scratch website is the beginning of a “literature” for Scratch.

The Scratch website is fertile ground for collaboration. Community members are constantly borrowing, adapting, and building on one another’s ideas, images, and programs. Roughly 30% of the projects on the site are remixes of other projects on the site. For example, there are dozens of versions of the game Tetris, as Scratchers continue to add new features and try to improve gameplay – and, in the process, learn important computational concepts and strategies.
Looking Forward
Scratch has been successful in attracting a more diverse range of young people than traditional computer-programming languages and courses – and supporting those young people on interest-driven trajectories towards computational fluency. But Scratch is still reaching only a small slice of the overall population. As we look ahead, we need to develop new design strategies, new pedagogical approaches, and new social frameworks to enable a wider range of young people to find and follow interest-driven trajectories into computational fluency.