Building Blocks for Powerful Ideas: Designing a Programming Language to Teach the Beauty and Joy of Computing

Jens Mönig
jens.moenig@sap.com/ Research Expert SAP, Germany

Recepción: 9-8-2019 / Aceptación: 21-8-2019

Abstract. Snap! is a cloud-native graphical programming environment and an online community. It is the programming language made for UC Berkeley's popular introductory CS course named "The Beauty and Joy of Computing". Snap! is taught in colleges and high schools across the U.S. from Palo Alto to Philadelphia. It has been translated to more than 40 languages and is used around the world—from Göttingen to Beijing—for teaching and research. Snap! has been designed for inclusion. Its low floor welcomes beginners and its multi-media capabilities invite creative thinkers of all ages. At the same time, Snap! offers sophisticated abstractions that make it suitable for an intellectually rigorous introduction to computer science.

Keywords: Snap!, BJC, AP CSP, CS0.

Construyendo bases sólidas para ideas poderosas: diseñando un lenguaje de programación para enseñar la belleza y alegría de la informática

Resumen. Snap! es un entorno de programación gráfica nativo de la nube y una comunidad en línea. Es el lenguaje de programación creado para el popular curso introductorio de CS de UC Berkeley llamado "La belleza y la alegría de la informática". Snap! se imparte en colegios y escuelas secundarias de los EE. UU., desde Palo Alto hasta Filadelfia. Se ha traducido a más de 40 idiomas y se utiliza en todo el mundo, desde Gotinga hasta Beijing, para la enseñanza y la investigación. Snap! ha sido diseñado para su inclusión. El nivel bajo le da la bienvenida a principiantes y sus capacidades multimedia que invitan a pensadores creativos de todas las edades. Al mismo tiempo Snap! ofrece abstracciones sofisticadas que lo hacen adecuado para una introducción intelectualmente rigurosa a la informática.

Palabras clave: Snap!, BJC, AP CSP, CS0.

1. INTRODUCTION

Recent years have seen a thunderous revival of programming education, sparked by a growing demand for computationally skilled workforce and spearheaded by MIT's visual Scratch language. In Scratch's wake, a new class of so-called "blocks-based" programming editors has appeared, and visual coding has since evolved into the de-facto standard for introductory CS activities. Along with Scratch's metaphor of stacking bricks, representing chunks of code into program-"towers" that are executed from top to bottom, a very traditional imperative style of programming has been established as quasi-best practice for introducing children and novices to CS.

At the same time, driven by the asynchronous nature of web programming and massive parallelization on the backend side to cope with "big data", many professional text-based programming languages have been revamped to support functional programming techniques such as proper tail-calls and even lambda, which before were considered too exotic to become mainstream. The gap between what beginners are exposed to in visual blocks-based languages and what is required to express themselves in a professional modern programming language today is more than just syntax. The gap is also conceptual and calls for proficiency in paradigms.

2. A CHALLENGE

Frequently discussed among educators is how to foster the transition from visual to textual programming. Sometimes the proposed solutions suggest bi-modal code editing, being able to switch back and forth between blocks and text. While this might address the lesser issue of coping with textual syntax, it does not help with introducing concepts and paradigms unsupported by any one side, and in the worst case even impoverishes the beginner's programming experience to the least common denominator of two programming languages.

On the other side of the spectrum, efforts are under way to broaden the scope and raise the ceiling of blocks-based programming. I will present one such project: Snap! Build Your Own Blocks. Snap! is a Scratch-like programming language that treats code blocks as first-class citizens instead of confining them to an editing modality. Embracing nested data structures and higher order functions, Snap! lets learners create arbitrary control structures and even custom programming languages with just blocks. Snap! has been developed for UC Berkeley's introductory computer science course named "The Beauty and Joy of Computing".

3. IN THIS TALK

I will share thoughts on the design of Snap! in a live-programmed excursion touching on a selection of powerful ideas from algorithms to artificial intelligence.