Scaffolding to Support Liberal Arts Students Learning to Program on Photographs

Mark Guzdial
mjguz@umich.edu
University of Michigan
Program in Computing for the Arts and Sciences
Ann Arbor, MI, USA

ABSTRACT

Digital photographs are part of liberal arts students' classes (e.g., art, history, and production classes in film and television) and their daily smartphone-based life, in apps like Instagram and Snapchat. Building image filters can be a relevant and engaging context into using computing for humanities students. We have designed a new course for introducing computing in terms of creative expression. We use a *scaffolded sequence of programming languages and activities* to explore computing on photographs: (a) a teaspoon language for generating image filters, (b) a set of custom Snap blocks for even more sophisticated image filters, and (c) an ebook activity for mapping from Snap to Python. Each stage takes less than 10 minutes to introduce, with a wide variety of possible student activities (for in-class active learning or for later homework). While the tools build on each other, the earliest stage (the teaspoon language) could be used within a single class session in other liberal arts courses.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

liberal arts and sciences, computational literacy, CS for All, computational thinking, digital humanities, critical computing, digital photography, image filters

ACM Reference Format:
Mark Guzdial. 2023. Scaffolding to Support Liberal Arts Students Learning to Program on Photographs. In *Proceedings of the 28th ACM Conference on Innovation and Technology in Computer Science Education Vol 2 (ITiCSE 2023), July 10–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXX

Students often manipulate digital photographs with their smart phones, which can make it a useful and relevant programming context for non-CS student. In a new course being developed at the University of Michigan, liberal arts students learn programming in a unit focused on digital image manipulation. Our goal is to engage and interest students in exploring computing further, to develop *conversational programming* skills [1], and to avoid the decrease in self-efficacy that is common in introductory programming classes [4].

In the image unit, we use a *scaffolded sequence of programming languages and activities* to develop knowledge and skills around computing on photographs. Each stage takes less than 10 minutes to introduce in class (and all three can be demonstrated in less than 15 minutes in a conference), yet each stage is flexible enough to support a variety of student activities. While the tools build on each other, the earliest stage (the teaspoon language) could be used within a single class session in other liberal arts courses, such as film or television production class when explaining techniques like chromakey ("green screen" effects).

1 TEASPOON LANGUAGE FOR PIXEL MANIPULATION

Students start programming with language using a *teaspoon language* [5, 6] for pixel manipulation (Figure 1). A teaspoon language is a very small programming language for a specific task. The Pixel Equations teaspoon language can only be used to create image filters by specififfying logical expressions for selecting pixels and arithmetic expressions for setting colors.

After choosing a digital image to manipulate, students specify their image filter by (a) writing a logical expression describing the pixels that they want to manipulate and (b) writing equations for how to compute the red, green, and blue channels for those pixels. In Figure 1, all those pixels on the right half of the picture ($x > 0$, because $(0, 0)$ is the center of the picture) will have their red channels set to 200 (a high value, because each channel is only a single byte, with values from 0 to 255). The equation for specifying the channel change can also reference the previous values of the channels, using the variables *red, green, blue, rojo, verde,* or *azul.*

A variety of image filters can be created with this simple model. For example, we can posterize to reduce the range of red values in a picture to only two (Figure 2). We can also negate an image, to create the negative of each color (Figure 3). An important computer science idea to explore is to change the input picture without changing the filter specification, in order to see how an algorithm works the same on different inputs.

Student activities: The earliest activities are to change the constants in the given examples. For example, students change the thresholds for posterizing filters, or change the target values. They can posterize on green or blue instead of red, or posterize two channels. They get a different effect if they use a constant other than 255 in the negation filter. We might challenge students to recreate a given filtered image, or to figure out which specification generated the given image.

ITiCSE 2023, July 10–12, 2023, Turku, Finland

ACM ISBN YYY.
https://doi.org/XXX

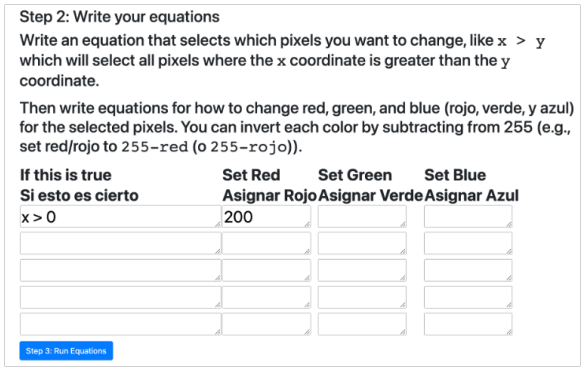

Figure 1: Specifying an image filter in the Pixel Equations teaspoon language

Figure 2: Specifying a posterizing image filter in the Pixel Equations teaspoon language

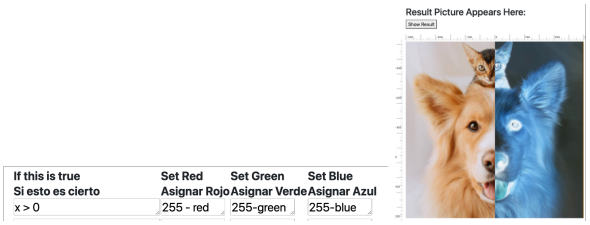

Figure 3: Specifying a negation image filter in the Pixel Equations teaspoon language

2 SNAP CUSTOM BLOCKS FOR PIXEL MANIPULATION

Students are given Snap [3] blocks that can do equivalent manipulation of pixels. Figure 4 negates an image, as in Figure 3. We provide students with a special loop construct "*for each pixel in pixels of image*." We provide blocks for setting each channel or reading each channel. Using an *if* block, we can process only certain pixels.

We provide a custom Snap block for accessing another pixel in another (or the same picture) at a given (x, y) position. With this block, we can inset one image inside another (Figure 5), chromakey (where a background color is replaced with another image, commonly used in movie production), or mirror an image.

Student activities: We ask students to create two image filters, then use each image filter to create modified forms of two of their own digital pictures. Students are then asked to export the images from Snap and create a collage of these images using a slideshow

Figure 4: Negating an image

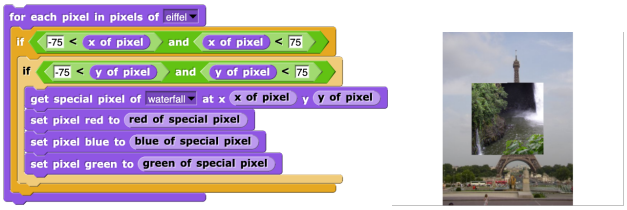

Figure 5: Inserting a portion of one image inside of another

tool (like Powerpoint, Keynote, or Google Slides). Students are encouraged to be creative, and perhaps create a story with their filters and images.

3 SUPPORTING TRANSFER TO PYTHON

At the end of the pixel unit, students engage with a purpose-built Runestone ebook [2]. On each page of the ebook, students see a Snap program that they used in class and a Python program that implements the same filter. Students then answer multiple-choice questions about the Python program. The goal is to encourage transfer of knowledge from their Snap programming into more traditional textual programming [7]. The ebook activities are informed by *purpose-first programming* [1] to develop conversational programming skills and encourage a sense of self-efficacy and authenticity.

REFERENCES

[1] Kathryn Cunningham, Barbara J. Ericson, Rahul Agrawal Bejarano, and Mark Guzdial. 2021. *Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code's Purpose.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445571

[2] Barbara J. Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. 2016. Identifying Design Principles for CS Teacher Ebooks Through Design-Based Research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. ACM, New York, NY, USA, 191–200. https://doi.org/10.1145/2960310.2960335

[3] Dan Garcia, Michael Ball, and Yuan Garcia. 2022. Snap! 7 - Microworlds, Scenes, and Extensions!. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1179. https://doi.org/10.1145/3478432.3499266

[4] Jamie Gorson and Eleanor O'Rourke. 2020. Why do CS1 Students Think They're Bad at Programming? Investigating Self-efficacy and Self-assessments at Three Universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 170–181.

[5] Mark Guzdial. 2022. Creating New Programming Experiences Inspired by Boxer to Develop Computationally Literate Society. In *Companion Proceedings of the 6th International Conference on the Art, Science, and Engineering of Programming* (Porto, Portugal) *(Programming '22)*. Association for Computing Machinery, New York, NY, USA, 67–69. https://doi.org/10.1145/3532512.3539663

[6] Mark Guzdial. 2022. Teaspoon Languages for Integrating Programming into Social Studies, Language Arts, and Mathematics Secondary Courses. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1027. https://doi.org/10.1145/3478432.3499240

[7] Ethel Tshukudu and Quintin Cutts. 2020. Semantic Transfer in Programming Languages: Exploratory Study of Relative Novices. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 307–313.