Scaffolding to Support Humanities Students Programming in a Human Language Context

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

ABSTRACT

Language is a key topic of interest for students in the humanities – language is the way in which humans express themselves, communicate, and make art. Computing on language (e.g., recognizing language, generating language, building bots) can be a pathway into using computing for humanities contexts. At the University of Michigan, we are developing a new program to support students in liberal arts and sciences to learn about computing, explicitly including programming. We have designed two courses for introducing computing (1) in terms of creative expression and (2) around the implications of computing on justice. In both classes, we use a *scaffolded sequence of programming languages and activities* to explore computing on language: (a) a teaspoon language for sentence generation and recognition, (b) a set of custom Snap blocks for sentence generation and recognition, (c) a set of custom Snap blocks for building Chatbots, and (d) an ebook activity for mapping from Snap to Python. Each language 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 linguistics, communications, or 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

ACM Reference Format:
Mark Guzdial. 2023. Scaffolding to Support Humanities Students Programming in a Human Language Context. 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

Much of the humanities is centered on language which can make it a useful and relevant programming context for humanities students [5]. Programming on language might include recognizing parts of speech in a sentence, generating new sentences from a language model, and building bots for chatting or Twitter. In two new courses being developed at the University of Michigan, liberal arts students learn programming contextualized around language. Our goal is to engage and interest students in exploring computing further, to develop *conversational programming* skills [1], and to avoid decreases in self-efficacy that is so common in introductory programming classes [4].

In both classes, we use a *scaffolded sequence of programming languages and activities* to explore computing on language. Each stage takes less than 10 minutes to introduce in class (and all four stages can be demonstrated in less than 15 minutes at a conference!), yet 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 linguistics, communications, or other liberal arts courses.

1 TEASPOON LANGUAGE FOR SENTENCE RECOGNITION AND GENERATION

Students start programming with language using a *teaspoon language* [6, 7] for sentence generation and recognition (Figure 1). A teaspoon language is a very small programming language for a specific task. The sentence model is specified the same way for recognition or generation, using just the five words: **noun**, **verb**, **adjective**, **adverb**, and **article**.

The sentence recognizer progresses through each word in the **model** looking for a match in the **input sentence**, based on the provided lexicon. If a match fails, the rest of the model is ignored. While simple, there are still opportunities to have expectations fail and to learn debugging. For example, the model "noun verb noun" will match to"The lazy dog runs to the student," it will fail to match on the second noun in "The lazy dog runs to the house," until the word "house" is added to the default lexicon.

The sentence generator uses the same model to randomly select words from the lexicon to generate sentences. Each execution of the model generates 10 sentences, not all of which will make sense. In class, we can talk about what features lead to a higher rate of reasonable sentences: shorter or longer language models, more words in the lexicon, or more carefully chosen words in the lexicon.

Student activities: Given equivalent sentences in Standard English, Spanglish, African-American Vernacular English, and Hawaiian Creole, students are challenged to create language models and lexicons that will recognize each, all, or even more than one of the sentences. Students discuss how AI assistants like Alexa or Siri might handle the problem of recognizing different forms of English.

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

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

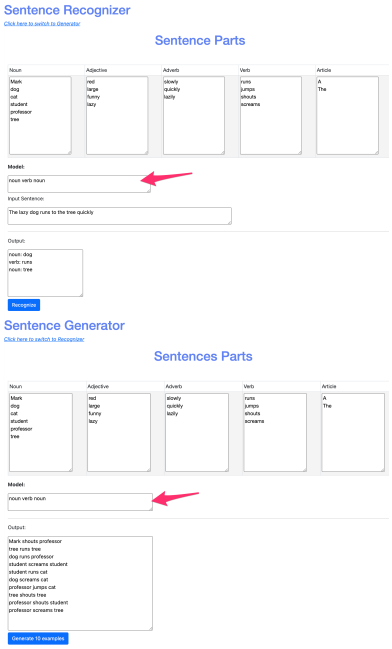

Figure 1: Sentence Recognition and Generation teaspoon language

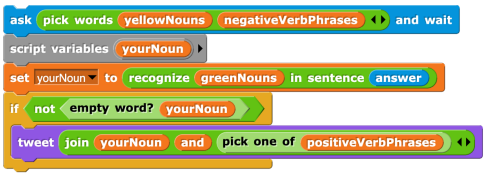

Figure 2: Creating a politically-biased bot

2 SNAP CUSTOM BLOCKS FOR SENTENCE RECOGNITION AND GENERATION

Students are given Snap [3] blocks that can do equivalent sentence recognition and generation, such as identifying parts of speech from a lexicon given an input sentence, or generating sentences from the parts of speech. These are sufficient to create an algorithm for how a bot might be created to saying something negative about the "yellow party," wait for a response that mentions a noun related to the opposing "green party," then retort with a positive statement about the same "green" noun (Figure 2).

Student activities: Students can create programs to generate random Dr. Seuss-like text such as "Cat on hat. Cat on mat." By creating new lexicon categories (e.g., "verbs that end in -er" or "nouns that end in -at"), students can generate random rhyming text. A common first programming assignment is to generate random haiku using lexicon categories built around the number of syllables in the noun, verb, or other language part.

Figure 3: Two rules in a simple Eliza-like chatbot

3 SNAP CUSTOM BLOCKS FOR CHATBOTS

Students are given examples of chatbots built in Snap with a common structure: Prompt the user for a response, then test the response against a set of rules that change the prompt. To simplify language parsing, students are given custom blocks that *match all* words in the input or *match any* words in the input (Figure 3).

Student challenges: Students develop chatbots to represent fictional or historical characters, or develop chatbots with different political or philosophical perspectives.

4 SUPPORTING TRANSFER TO PYTHON

At the end of the text 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 does the same thing. 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 [8]. 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. 2010. Does contextualized computing education help? *ACM Inroads* 1, 4 (2010), 4–6.

[6] 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

[7] 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

[8] 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.