Python Programming Education with Semantics-oriented Screen Reading for K-12 Students with Vision Impairments

God'salvation F. Oguibe
The University of Texas at San Antonio
USA
Godsalvation.Oguibe@my.utsa.edu

Lauryn M. Castro
The University of Texas at San Antonio
USA
Lauryn.Castro@my.utsa.edu

Katherine Cantrell
The University of Texas at San Antonio
USA
Katherine.Cantrell@utsa.edu

Kathy B. Ewoldt
The University of Texas at San Antonio
USA
Kathy.Ewoldt@utsa.edu

Leslie Cockerill Neely
The University of Texas at San Antonio
USA
Leslie.Neely@utsa.edu

Wei Wang
The University of Texas at San Antonio
USA
Wei.Wang@utsa.edu

ABSTRACT

Because of the high pay and high flexibility, computer science careers can be highly viable for people with blindness or vision impairments (BVI). However, in our programming education, we observed that existing screen readers used by students with BVI usually cannot properly handle computer programs, which mix English letters, digits, and punctuation marks. When applied to computer programs, current screen readers either ignore the punctuation marks, or mix English words, digits, and punctuation marks, making the screen reading either incorrect or hard to understand. The resulting difficulty in understanding program statements significantly hinders students' ability to locate incorrect code and independent coding.

To address these limitations in current screen reading, we are developing a new semantic-oriented screen reader, called *JupyterVox*, which reads Python statements by their meanings to speedup code navigation, and thus improve independent coding skills. This new screen reader is based on compiler's lexical and syntax analyses to parse, understand, and read program statements. This poster presents the initial implementation and results of *JupyterVox*.

CCS CONCEPTS

• **Human-centered computing** → **Accessibility technologies**; • **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → **Compilers**.

KEYWORDS

Special Education, Computer Programming Education, Blindness and Vision Impairment

ACM Reference Format:
God'salvation F. Oguibe, Lauryn M. Castro, Katherine Cantrell, Kathy B. Ewoldt, Leslie Cockerill Neely, and Wei Wang. 2024. Python Programming Education with Semantics-oriented Screen Reading for K-12 Students with Vision Impairments. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635494

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635494

1 INTRODUCTION

Software engineers are among the highest-paid occupations in the US. In addition to the high pay, software engineering jobs are also flexible in working modality. A significant number of software engineering positions are remote and only require a desk, a computer, and the Internet. These combined benefits make software engineering a perfect position for people with disabilities, including people with blindness or vision impairments.

In the past years, the authors have been teaching simple Python programming and machine learning to middle and high school students with BVI. A major difficulty we and our students have encountered in our teaching is the confusing readings from existing screen readers. Screen readers are typically designed to read English paragraphs instead of computer program statements, which mix letters, digits, and punctuation marks. The computer statement readings from current screen readers are typically difficult to understand – the readings can be either complex and/or misleading. A direct result of this understanding difficulty is the slow code reading and code navigation, which directly leads to slow programming speed and low independent coding abilities. Some students would completely avoid using screen readers when coding and chose to use the more expensive Braille displays. More specifically, there are at least two issues with current screen readers,

(1) First, by default, most screen readers will ignore (some of) the punctuation marks when reading. For example, as shown in Figure 1, for the dictionary definition statement {"$t1$" : a, "$t2$" : b}, Windows Narratoronly reads "T one A two twelve." Mac VoiceOverreads braces, but not quote marks. These readings completely misrepresented the statement.
(2) Second, when the screen readers are configured to read punctuation marks, the readings become long and confusing. Figure 1 also shows the screen readings of Windows Narrator and Mac VoiceOver when they are configured to read all

	Standard Screen Reader			*JupyterVox*		
Type of Statement	Statement	Time to Understand	Understanding Correctness	Statement	Time To Understand	Understanding Correctness
Expression	a*b+c	10.24s	correct	a*(b+c)	9.67s	correct
Assignment	z=y%3//5	21.71s	incorrect	b=a//2 % 3//5	11.84s	correct
Aug Assignment	x +=5	6.82s	correct	b -= a	7.48s	incorrect
Assign w. Expr.	c=3-x*y	34.42s	incorrect	c = y/z+3	6.80s	correct
Complex Assign.	a=(x*y)+(z/5-b)	10.37s	incorrect	c = (a*b) - (3*z/y)	43.7s	correct
if statement	if x<=10: return 9	7.12s	correct	if x>=3: return 4	8.87s	correct
String Variable	test_string="Hello World"	7.19s	incorrect	day_string="Sunday"	10.80s	correct

Table 1: Initial evaluation results.

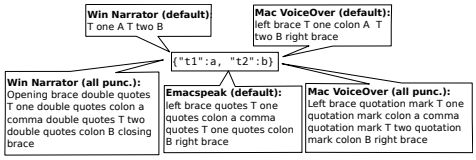

Figure 1: An example of how the handling of punctuation marks affect screen readings for three screen readers under the default and read-all-puncutations (all punc.) settings.

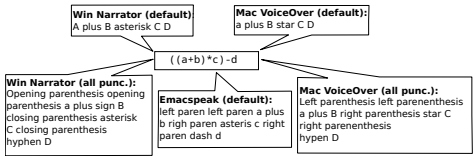

Figure 2: Second example of how the handling of punctuation marks affect screen readings, especially for statements with nested parentheses.

punctuation marks. Additionally, Figure 1 includes the reading of Emacspeak [4], which is a screen reader that understands programs and reads punctuation marks by default. Figure 1 shows that all these readings mixed punctuation marks, digits, and English words, and require considerable effort to parse in one's mind. This problem is even more complicated when nested parentheses are involved (e.g., Figure 2) – determining the matching pairs of parentheses while they are read is extremely difficult for beginners.

Two other popular screen readers, Jaws [2] and NVDA [1], also have the above limitations. There is also work on using audio-assisted debugging [6, 7] and more intuitive languages [3, 5, 8]. However, these studies cannot be directly applied to Python, which is extensively used in Data Science.

2 METHODOLOGY

To address these limitations, our hypothesis is that: *Instead of reading literally, the understanding of program statements can be significantly simplified if the statements can be read based on their meaning (semantics), which, in turn, accelerates code navigation and improves independent coding skills for programming education.*

As lexical and syntax analyses in compilers already understand the semantics of program statements, it is then possible to employ compiler techniques to generate semantics-oriented readings. More specifically, we employ Python AST module to generate an abstract syntax tree (AST) given a statement, then convert that AST tree to an English sentence representing the meaning of that statement. This conversion closely resembles the coding generation processing

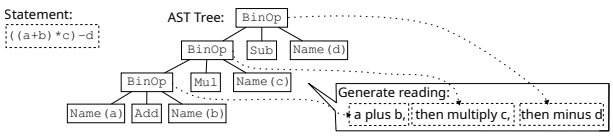

Figure 3: An example of reading generation using AST tree in *JupyterVox* using the same statement from Figure 2.

using an AST tree. Figure 3 gives an example of how speeches are generated based on an AST tree for the same statement in Figure 2. The reading is "a plus b, then multiply c, then minus d."

3 EVALUATION

We finished an initial implementation of the screen reader, called *JupyterVox*. As an early evaluation of *JupyterVox*, we conducted a short experiment with one human subject who listened to seven statements read by the an existing screen reader and *JupyterVox*. Table 1 gives the results of this screen reader. The subject was considered as having understood the reading once he could restate the statement correctly. As Table 1 shows, *JupyterVox* could significantly increase the correctness of statement understanding, from only three statements to six statements.

ACKNOWLEDGEMENT

This work is supported by NSF grants, 2202632, 2221843, 2155096, and 2215359. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of NSF.

REFERENCES

[1] NV Access. 2023. About NVDA. https://www.nvaccess.org/about-nvda/. (2023).
[2] Freedom Scientific Inc. 2023. JAWS. https://www.freedomscientific.com/products/software/jaws/. (2023).
[3] Aboubakar Mountapmbeme and Stephanie Ludi. 2020. Investigating Challenges Faced by Learners with Visual Impairments Using Block-Based Programming/Hybrid Environments. In *Int'l ACM SIGACCESS Conference on Computers and Accessibility.*
[4] T.V. Raman. 1997. Emacspeak-an Audio Desktop. In *IEEE COMPCON.*
[5] Jaime Sánchez and Fernando Aguayo. 2006. APL: Audio Programming Language for Blind Learners". In *Computers Helping People with Special Needs*, Klaus Miesenberger, Joachim Klaus, Wolfgang L. Zagler, and Arthur I. Karshmer (Eds.).
[6] Andreas Stefik, Roger Alexander, Robert Patterson, and Jonathan Brown. 2007. WAD: A Feasibility study using the Wicked Audio Debugger. In *IEEE Int'l Conf. on Program Comprehension.*
[7] Andreas Stefik, Andrew Haywood, Shahzada Mansoor, Brock Dunda, and Daniel Garcia. 2009. SODBeans. In *IEEE Int'l Conf. on Program Comprehension.*
[8] Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. 2011. On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science. In *ACM Technical Symp. on Computer Science Education.*