Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education

Philip J. Guo
Google, Inc.
Mountain View, CA, USA

ABSTRACT

This paper presents Online Python Tutor, a web-based program visualization tool for Python, which is becoming a popular language for teaching introductory CS courses. Using this tool, teachers and students can write Python programs directly in the web browser (without installing any plugins), step forwards and backwards through execution to view the run-time state of data structures, and share their program visualizations on the web.

In the past three years, over 200,000 people have used Online Python Tutor to visualize their programs. In addition, instructors in a dozen universities such as UC Berkeley, MIT, the University of Washington, and the University of Waterloo have used it in their CS1 courses. Finally, Online Python Tutor visualizations have been embedded within three web-based digital Python textbook projects, which collectively attract around 16,000 viewers per month and are being used in at least 25 universities. Online Python Tutor is free and open source software, available at `pythontutor.com`.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*Computer-assisted instruction*
; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Languages, Human Factors, Experimentation

Keywords

CS1, Python, Program Visualization

1. INTRODUCTION

A core challenge in introductory computer science ("CS1") courses is getting students to understand how a static textual representation (source code) maps to a highly dynamic process (program execution). Instructors typically illustrate program execution using graphical PowerPoint lecture slides, which require a great deal of preparation effort, or by drawing diagrams on a whiteboard, which is tedious and error-prone [11]. In the past few decades, many *program visualization* tools have been created to assist instructors in this task. Chapter 11 of Sorva's dissertation [15] provides a comprehensive overview of 40 such tools, which all look similar at first glance: One GUI pane shows source code with the currently-executing line highlighted, another pane shows a visual representation of run-time state (e.g., stack frame contents, heap objects), and control widgets allow the user to step forwards and backwards through execution points.

This paper presents a free and open-source project called Online Python Tutor, which follows in the long tradition of program visualization tools for CS education. In spite of the rich history of related work in this field, three characteristics make Online Python Tutor modern, unique, and effective:

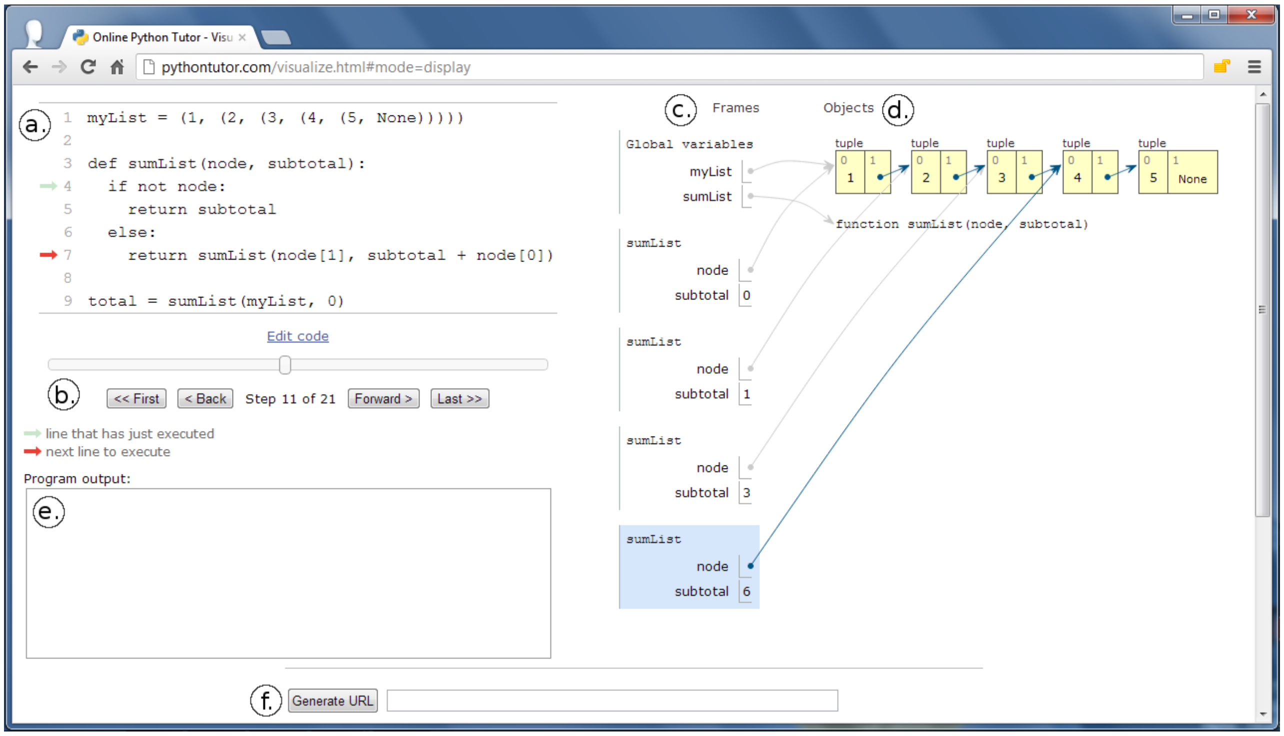
Python: In recent years, Python has been gaining traction as a preferred language for CS1 courses across many universities. For example, two of the largest CS departments—MIT and UC Berkeley—both ported their CS1 curricula from Scheme to Python. Michigan State University switched from C++ to Python and presented empirical evidence that new students were just as prepared for advanced CS courses taught in other languages [7]. Python is also used in free online CS1 courses offered by Udacity, Coursera, and edX, which each enroll tens of thousands of students per term.

Despite the prevalence of Python in teaching, there are almost no program visualization tools for Python. Aside from Online Python Tutor, UUhistle [16] and Jype [9] are the only other Python visualizers[1]. (However, the Jype project seems to be inactive [15], and we cannot find its code online.)

Web-Based: Online Python Tutor is the only Python program visualization tool that runs within a web browser without any required software or plugin installation[1]. UUhistle and Jype are written in Java, so they can be launched from the web on a computer with the proper Java version installed; however, the entire application must first download before launching, which can take a few minutes. In contrast, opening any visualization in Online Python Tutor is as simple and fast as visiting a URL from a web browser.

We believe that this ease of access is a major contributor to adoption: Online Python Tutor has been used in CS1 courses in a dozen universities including UC Berkeley, MIT, the University of Washington, and the University of Water-

[1]to the best of our knowledge



```
1  myList = (1, (2, (3, (4, (5, None)))))
2
3  def sumList(node, subtotal):
4    if not node:
5      return subtotal
6    else:
7      return sumList(node[1], subtotal + node[0])
8
9  total = sumList(myList, 0)
```

Figure 1: Online Python Tutor is a web-based program visualizer where the user can: a.) view the currently-executing line of code, b.) step forwards and backwards through execution using a slider bar and buttons, c.) view stack frames and variables, d.) view heap object contents and pointers, e.) view the program's text console output, and f.) generate a sharable URL of the current visualization at an exact execution point.

loo. Over 30,000 people (unique IP addresses) per month use Online Python Tutor by visiting `pythontutor.com`.

Embeddable: Finally, Online Python Tutor is the only program visualizer that can be seamlessly embedded within webpages[1]. So far, three web-based interactive digital CS textbooks [6, 10, 14] have embedded Online Python Tutor into their lessons (by inserting one line of JavaScript code per visualization). Students can read lessons and interact with code visualizations within the same webpage (Figure 7).

These three digital textbooks attract around 16,000 unique viewers per month and are being used in at least 25 universities around the world. Alvarado et al. analyzed usage log data from a CS1 course that used one such textbook [10] and found that students who interacted more with the embedded visualizations tended to score higher on midterm exams [2].

2. OVERVIEW

Figure 1 shows the Online Python Tutor webpage loaded in Google Chrome. In this screenshot, the user has previously typed in a 9-line Python program for computing the sum of a linked list and is now visualizing execution step 11 of 21, in the midst of the fourth recursive call to `sumList`.

We now describe the GUI components corresponding to each label in Figure 1:

a.) The source code display shows the program that is being visualized. A red arrow in the left margin points to the next line to be executed (line 7 in this example). A light green arrow points to the line that has just executed, which helps users track non-contiguous control flow (e.g., function calls).

b.) A slider bar and text indicate the current execution point being visualized (in this example, step 11 of 21). Each point represents a single executed line. The user can click on or drag the mouse over the slider bar to jump to a particular point or use the VCR-style navigation buttons to step forwards and backwards over executed lines.

c.) The frames pane shows global variables and stack frames at the current execution point, with the stack growing downward. Each frame shows the function name and a list of local variables. Each variable's value is an arrow that points to a heap object. The currently-active (bottommost) frame is highlighted in blue, as are all arrows emanating from it. To reduce visual clutter, arrows emanating from all other frames are rendered in light gray. In this example, the `node` variable in each recursive call to `sumList` each point to successive elements in the linked list. Finally, even though every object in Python is technically on the heap, we chose to render immutable primitive objects (e.g., numbers, strings, bools) inline within each frame for visual simplicity.

d.) The objects pane shows visual representations of Python objects and their pointer references to one another. Online Python Tutor renders all compound data types necessary for teaching CS1, including list, tuple, set, dict, class, object instance, and closures (see Figure 2). This example shows a linked list of integers built from a chain of five tuples.

e.) The program output text box shows the cumulative output of `print` statements issued by the program at the current execution point.

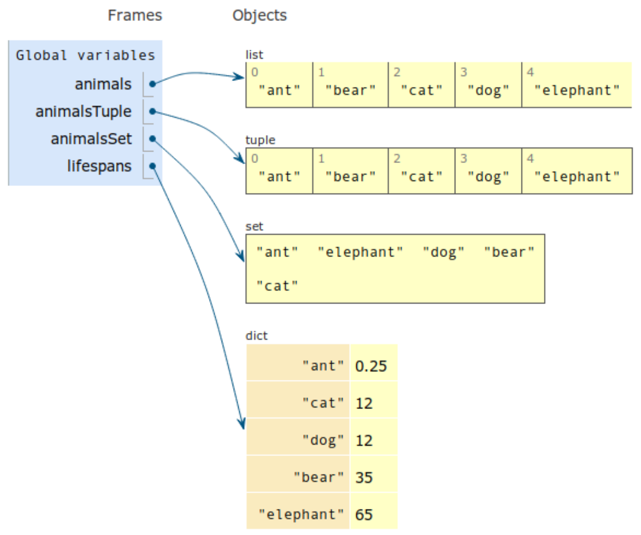


Figure 2: Visualizations of Python's four main compound data types: list, tuple, set, and dict (class, instance, and closure not shown due to space limits)

f.) Clicking the "Generate URL" button creates a URL that uniquely identifies the current program being visualized and also the current execution point. In this example, the URL encodes the complete contents of the 9-line Python program and also includes "step 11 of 21." The user can send that URL in an email or post it to a discussion forum along with a note. When someone clicks on that URL, they are brought to the exact execution point that the sender has specified.

3. DESIGN METHODOLOGY

Online Python Tutor started as a graduate student side project in January 2010, motivated by the creator's experiences with teaching Python by drawing messy stack and heap diagrams on the whiteboard. Around that time, web browsers and technologies were getting sophisticated enough that it became possible to build an educational program visualization tool that could run within the browser.

Our design goal is to make a tool that Python instructors and students prefer to use instead of (or in addition to) traditional whiteboard and PowerPoint diagrams.

We started by taking common features from existing program visualization tools (surveyed in this dissertation [15]) and selectively adapting them to a web browser environment. Over the past three years, the Online Python Tutor user base has grown organically via word-of-mouth publicity. We refined our initial design and implemented new features (e.g., nested function and lambda support) by consulting with our users, which includes a diverse group of CS instructors—teaching assistants, professors at both small and large universities, and an instructor of several free online CS classes with over 200,000 total enrolled students[2].

4. IMPLEMENTATION

Online Python Tutor is a web application with a backend written in Python and a frontend written using standard web technologies: HTML, CSS, and JavaScript.

[2]courses online at `www.ai-class.com` and `www.udacity.com`

4.1 Backend (Python)

The Online Python Tutor backend takes the source code of a Python program as input and produces an *execution trace* as output. The backend executes the input program under supervision of the standard Python debugger module (`bdb`), which stops execution after every executed line and records the program's run-time state. The trace is an ordered list of execution points, where each point contains the state right before a line of code is about to execute, including:

- The line number of the line that is about to execute,
- the instruction type (ordinary single step, exception, function call, or function return),
- a map of global variable names to their current values at this execution point,
- an ordered list of stack frames, where each frame contains a map of local variable names to current values,
- the current state of the heap,
- and the program's output up to this execution point.

After execution terminates, the backend encodes the trace in JSON format [5], serializing Python data types into native JSON types with extra metadata tags. To guard against infinite loops and excessively long traces, the backend halts execution after 300 executed lines. Although this limit is tiny for real programs, it is sufficient for the pedagogical code examples that our users want to visualize. Note that the trace contains a great deal of redundancy, since it stores a complete snapshot of the stack and heap at every execution point. Traces from pedagogical code examples range from 10KB to 200KB in size. We have not found backend running times or trace sizes to be performance bottlenecks in practice and thus have not yet attempted to optimize.

The backend works with both Python 2 and 3, which is important because courses are taught using both of these major language variants.

The backend can be hosted on any webserver capable of running CGI scripts (using Python 2 or 3) or on the Google App Engine platform (Python 2.7). When the user's web browser makes an HTTP GET request to the backend and passes in the Python program to visualize, the backend generates a trace and returns it to the browser in JSON format.

Since the backend is running untrusted Python code from the web, it implements sandboxing to prevent the execution of dangerous constructs such as `eval`, `exec`, file I/O, and most module imports (except for a customizable whitelist of modules such as `math`). In addition, many webservers implement "defense in depth" for additional protections. However, we have not yet conducted a formal security audit.

4.2 Frontend (HTML/CSS/JavaScript)

The frontend is a website located at `pythontutor.com`, which renders in all modern web browsers without requiring the user to install any extensions or plugins. We have tested the website on a wide range of computer monitor sizes and even on tablet devices (see Figure 3).

The user first sees a text box and can type (or paste) a Python program into there. When the user clicks the "Visualize execution" button, the frontend sends that program to the backend; the backend then sends back a JSON execution trace, which the frontend renders in a GUI like Figure 1.

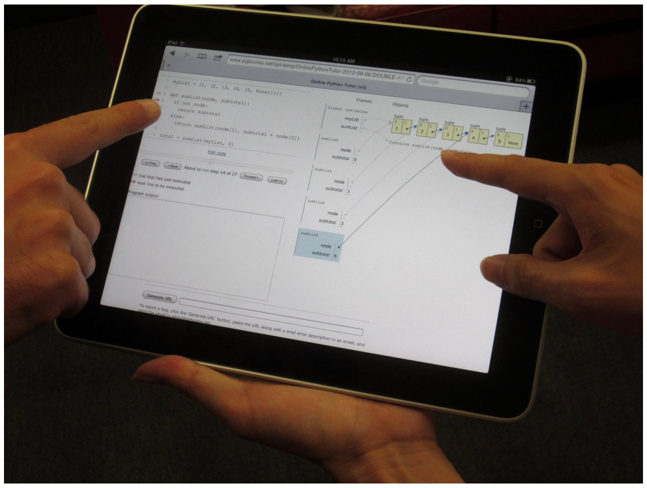


Figure 3: A teacher and student interacting with the Online Python Tutor visualization shown in Figure 1 using the default Safari web browser on an iPad.

From informal testing, the webpage visualizing the code in Figure 1 (and similar code examples) loads in ~ 0.4 seconds on a home broadband Internet connection and in ~ 3 seconds from a shuttle bus with relatively slow Wi-Fi.

After the initial webpage load, all interactions occur in the browser with no additional server calls; thus, stepping forwards and backwards refresh the display instantaneously.

Object layout: The main technical challenge in the frontend is rendering objects in a clear and aesthetically-pleasing way. Since the heap can be an arbitrary graph of objects, a naive rendering scheme could lead to a tangled mess of boxes and pointers. We implemented the following layout heuristics, inspired by looking at diagrams from textbooks and PowerPoint lecture slides from CS1 instructors:

- Each frame variable and heap object is stacked *vertically* in the order in which they were created during the program's execution (see Figure 2). Thus, when the user steps forward through execution, new variables and objects are always appended to the bottom of the display rather than being inserted in the middle.
- Once an object has been rendered, it stays in the same location (relative to other objects) until there are no more pointers to it. This heuristic makes objects not "jiggle" around when the user steps through execution.
- If a new compound object contains pointers to other *new* compound objects, then nest the inner objects inside of the outer one. For example, Figure 4 shows a list of four new Person object instances. Nesting both reduces clutter (instead of rendering more pointers) and also conveys a natural "containment" relationship.
- Since linked lists are common in CS pedagogy, the frontend visualizes them specially by laying out node objects *horizontally* (see Figure 5). The frontend determines that an object is likely a linked list if it contains pointers to identically-structured objects (e.g., a list/tuple of the same size or a dict/instance with the same attribute names). Note that this heuristic fares poorly for tree and graph structures: It simply "flattens" them to one horizontal dimension.

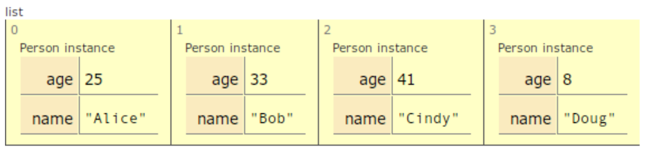


Figure 4: An example visualization of nested data: a list containing four instances of the Person class.

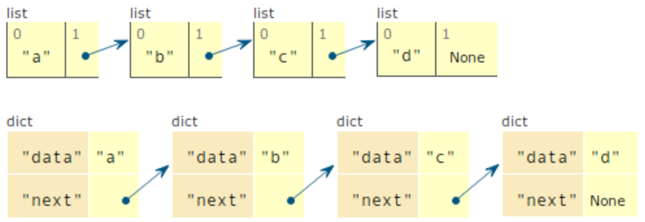


Figure 5: Visualizations of singly-linked lists constructed using Python lists (top) and dicts (bottom).

The frontend uses two main JavaScript libraries: D3 [3] to map execution trace objects to corresponding HTML elements and jsPlumb [13] to draw pointers as arrows.

Embedding: The entire visualizer GUI is encapsulated in a JavaScript `ExecutionVisualizer` class, so embedding it within a webpage takes only one line of code. For example:

```
var v = new ExecutionVisualizer(parentNode, trace);
```

`parentNode` is the HTML DOM element where the visualizer instance should be embedded. The execution `trace` can either be precomputed offline or generated by the web application backend. Multiple visualizer instances with different traces can be simultaneously embedded within one webpage.

5. USE CASES

We conservatively estimate that over 200,000 people have used Online Python Tutor in the past three years since its inception. There are three main categories of use cases:

- Instructors using it as a classroom teaching aid,
- digital textbooks with embedded Python visualizations, which collectively have been deployed in at least 25 universities and have around 16,000 viewers per month,
- students using it either while taking free online CS courses or during self-study for, say, programming interviews (`pythontutor.com` gets around 30,000 visitors per month, estimated by unique IP address hits).

Instructor use cases: Professors, lecturers, and teaching assistants in a dozen universities have used Online Python Tutor in their teaching. Class sizes ranged from 7 students in a summer Python course for non-CS majors at the University of Washington to over 900 students in the Fall 2012 offering of CS1 at UC Berkeley.

Several instructors told us via email that Online Python Tutor helped clarify concepts such as tracing parameters and return values through function calls, control flow, exceptions, scope, recursion, local variable lifetimes, and understanding aliasing when there are multiple identifiers referencing the same object. A professor at UC Davis wrote,

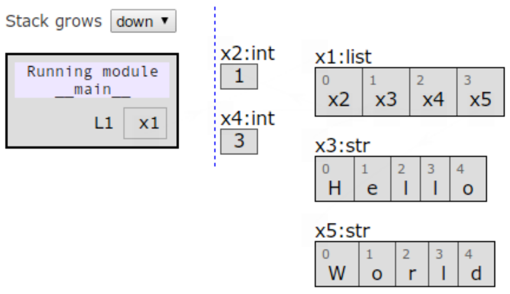


Figure 6: The University of Toronto's customized version of Online Python Tutor, which represents references as labels and strings as character arrays.

"I have students who are currently retaking the class in the summer session, since they dropped out in spring, when they were struggling. I can see from their reaction in class that this tool has literally been an eye-opener for them – it's making a huge difference!"[3] Another wrote, "The diagrams [Online Python Tutor] show[s] are just like the diagrams I'm constantly drawing on the whiteboard. So it matches the model that I try to get the students to see exactly."[3]

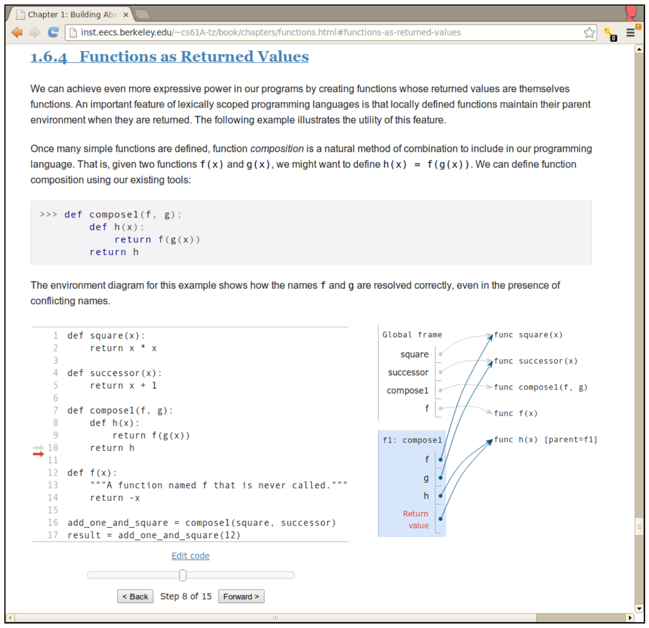
Although most instructors simply visited `pythontutor.com` during lectures and lab sessions to step through code examples, several went further:

- A CS1 teaching assistant at MIT augmented Online Python Tutor with beginner-friendly error messages, which his students found more helpful than Python's default error messages [8]. His error message templates were inspired by common mistakes that students in his class made.
- A team from the University of Toronto used Online Python Tutor as the basis for a CS *peer instruction* system, inspired by those used in science classes [12].
- Another team at the University of Toronto spent the summer of 2012 customizing Online Python Tutor to suit their own pedagogical preferences (see Figure 6). They deployed this customized version in their Fall 2012 CS1 offering, which is being concurrently taught at their university and online through the Coursera platform (with ∼34,000 enrolled students) [4].

Digital textbook use cases: In preparation for teaching the CS1 course at UC Berkeley, John DeNero ported the classic *Structure and Interpretation of Computer Programs* textbook from Scheme to Python and put it online in HTML format [6]. He embedded Online Python Tutor visualizations directly into the HTML so that students can read the lessons and interact with code examples on the same webpage (see Figure 7), as opposed to simply viewing static, manually-drawn diagrams of code examples.

Miller and Ranum embedded Online Python Tutor into their digital textbook, *How to Think Like a Computer Scientist: Interactive Edition* [10], as a component called CodeLens. This web-based e-textbook attracts around 6,000 viewers per month and has been adopted by 25 universities around the world. Alvarado et al. logged student interactions with this resource in a 61-person CS1 course and found that

[3]via personal email communication, August 2012


Chapter 1: Building Abs
inst.eecs.berkeley.edu/~cs61A-tz/book/chapters/functions.html#functions-as-returned-values

1.6.4 Functions as Returned Values

We can achieve even more expressive power in our programs by creating functions whose returned values are themselves functions. An important feature of lexically scoped programming languages is that locally defined functions maintain their parent environment when they are returned. The following example illustrates the utility of this feature.

Once many simple functions are defined, function *composition* is a natural method of combination to include in our programming language. That is, given two functions `f(x)` and `g(x)`, we might want to define `h(x) = f(g(x))`. We can define function composition using our existing tools:

```
>>> def compose1(f, g):
        def h(x):
            return f(g(x))
        return h
```

The environment diagram for this example shows how the names `f` and `g` are resolved correctly, even in the presence of conflicting names.

```
def square(x):
    return x * x

def successor(x):
    return x + 1

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

def f(x):
    """A function named f that is never called."""
    return -x

add_one_and_square = compose1(square, successor)
result = add_one_and_square(12)
```

Figure 7: Online Python Tutor visualization of a code example embedded within DeNero's digital textbook (used for UC Berkeley's CS1 course) [6].

students who interacted more with CodeLens visualizations outside of class hours scored higher on midterm exams [2].

Lastly, the Computer Science Circles project [14], a free online introductory programming course and textbook serving 10,000 visitors per month, has integrated Online Python Tutor as its main visualization and debugging tool. According to its creators, an average of 600 code snippets per day are visualized by their embedded Online Python Tutor[3].

Digital textbook authors often ran our Python backend offline to generate traces for individual code examples, which they then embedded directly as JSON objects in JavaScript code. Thus, the textbooks can work offline without needing to make server calls, which Miller found useful for deploying in regions with slow and unreliable Internet connections[3].

Student use cases: Over 30,000 unique IP addresses visit `pythontutor.com` each month, but to our knowledge, instructors in only a dozen universities are using it in their courses. Thus, we suspect that the majority of users are self-directed learners using it without instructor supervision.

Although we do not have detailed demographic information on users, we know at least one major source of users: students taking free online CS courses offered by Udacity [1], such as CS101: *Introduction to Computer Science* (186,000 total students over two offerings) and CS212: *Design of Computer Programs* (almost 40,000 students).

Online Python Tutor was not an official component of these courses, but some students found the tool and shared their use cases in discussion forums. One student posted, "This tool is particularly helpful when debugging recursive functions. I used it for the recursive unit at cs101 and it was a lifesaver." Other students wrote that it was "exactly what I need to solve knotty python questions without cheating" and "exactly what I need to untangle lists and arrays." [1]

Udacity's CS101 students, many of whom are computer novices, liked the web-based nature of Online Python Tutor because they did not need to install or configure software. They also posted custom URLs to accompany questions that they asked in the forums like, “Click on this link and please explain why this program does X at execution point Y.”

6. LIMITATIONS AND ONGOING WORK

We are delighted by the number of people who have benefited from Online Python Tutor so far, but at the same time, we acknowledge that numbers and anecdotes are no substitutes for a rigorous evaluation of efficacy. We would like to run some formal studies with academic partners to understand students’ attitudes toward and usage of Online Python Tutor, how it correlates with their performance in tracing, explaining, and writing programs, and impacts on their attitudes toward programming in general.

Study results might determine whether we should add new features to encourage active engagement with program visualizations, which prior work has shown to be effective [15], rather than passive viewing. This effort could involve embedding question prompts, mini-quizzes, textual annotations, and discussion threads directly within visualization elements.

Another Online Python Tutor limitation is that it steps through execution one line at a time like a debugger and does not show the details of expression and subexpression evaluation. In contrast, UUhistle [16] visualizes expression evaluation for a subset of Python constructs. Some users have requested such finer-grained stepping functionality.

A limitation we might not immediately address is scalability: Online Python Tutor can visualize only several hundred execution steps of programs with relatively small data structures. It is not meant to be used for debugging production-sized software systems. However, we may investigate how to cache and visually summarize large data structures to scale up to larger pedagogical code examples.

Currently the visualization works well for linked lists, but not for more sophisticated data types. Thus, we are planning to add custom visualizations for trees, graphs, numerical matrices, and other data structures so that Online Python Tutor can be useful in more advanced CS courses.

Another area of ongoing work is determining whether to add animations and smooth transitions between visualization states. The D3 library [3] we already use makes it easy to add animations, but the main design challenge will be determining which animations are pedagogically effective without becoming distractions.

Finally, since the Online Python Tutor frontend can technically visualize programs written in any mainstream language, we are now working with students to write backends for Ruby, JavaScript, and Racket (a Scheme dialect). Also, since the backend generates an execution trace that is not tied to any particular visualization, some users have suggested parsing the trace to render an audio “reading” of execution states to help teach visually-impaired students.

In closing, we have presented Online Python Tutor: a free, open-source, web-based tool following in the long tradition of program visualizations for CS education. Although the high-level ideas embodied by Online Python Tutor are decades-old, our main contribution is adapting and redesigning this sort of tool for a modern web-based environment, where it has been able to reach over 200,000 users thus far.

Acknowledgments: Thanks to Peter Norvig, Andrew Petersen, and David Pritchard for reading drafts of this paper.

7. REFERENCES

[1] Udacity - 21st Century University. `http://www.udacity.com/`.

[2] C. Alvarado, B. Morrison, B. Ericson, M. Guzdial, B. Miller, and D. A. Ranum. Performance and use evaluation of an electronic book for introductory Python programming. Technical Report GT-IC-12-02, Georgia Institute of Technology, 2012.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[4] J. Campbell and P. Gries. Learning to Program: The Fundamentals. `https://www.coursera.org/course/programming1`.

[5] D. Crockford. JSON (JavaScript Object Notation). `http://json.org/`.

[6] J. DeNero. CS61A: Structure and Interpretation of Computer Programs. `http://www-inst.eecs.berkeley.edu/~cs61a/`.

[7] R. J. Enbody, W. F. Punch, and M. McCullen. Python CS1 as preparation for C++ CS2. In *Proceedings of the 40th ACM technical symposium on Computer Science Education*, SIGCSE ’09, pages 116–120, New York, NY, USA, 2009. ACM.

[8] A. J. Hartz. CAT-SOOP: A tool for automatic collection and assessment of homework exercises. Master’s thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 2012.

[9] J. Helminen and L. Malmi. Jype - a program visualization and programming exercise tool for Python. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS ’10, pages 153–162, New York, NY, USA, 2010. ACM.

[10] B. Miller and D. Ranum. Beyond PDF and ePub: toward an interactive textbook. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, ITiCSE ’12, pages 150–155, New York, NY, USA, 2012. ACM.

[11] M. C. Orsega, B. T. Vander Zanden, and C. H. Skinner. Experiments with algorithm visualization tool development. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE ’12, pages 559–564, 2012.

[12] A. Petersen, D. Zingaro, Y. Cherenkova, and O. Karpova. Facilitating Code-Writing in PI Classes. In *Proceedings of the ACM technical symposium on Computer Science Education*, SIGCSE ’13, 2013.

[13] S. Porritt. The jsPlumb JavaScript library. `http://www.jsplumb.org/`.

[14] D. Pritchard and T. Vasiga. CS Circles: An In-Browser Python Course for Beginners. In *Proceedings of the ACM technical symposium on Computer Science Education*, SIGCSE ’13, 2013.

[15] J. Sorva. *Visual Program Simulation in Introductory Programming Education*. Ph.D. dissertation, Aalto University, 2012.

[16] J. Sorva and T. Sirkiä. UUhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling ’10, pages 49–54, New York, NY, USA, 2010. ACM.