PyCodeViz: An Interactive Visualization Tool to Facilitate Program Comprehension

Anush Mangal*
A. Eashaan Rao*
Sridhar Chimalakonda
Research in Intelligent Software & Human Analytics (RISHA) Lab
Department of Computer Science & Engineering
Indian Institute of Technology Tirupati, India
{cs20b054,cs21d002,ch}@iittp.ac.in

ABSTRACT

Novice developers often find it challenging to comprehend the execution of a program. Understanding the concepts of programming constructs and figuring out the changes in a variable during runtime requires program comprehension. Many tools are proposed in the literature to aid program comprehension. One such prominent tool is Python Tutor which offers visualization for step-by-step execution for code snippets in multiple programming languages. However, the tool do not showcase the minute changes for various programming constructs. Thus, in this paper, we propose *PyCodeViz*, with enhanced visualization exhibiting granular changes in constructs such as iterative, conditional, and recursive statements for Python. The main aim of *PyCodeViz* is to facilitate better program comprehension to novice developers. The tool has been evaluated with university students, and the results of the evaluation were positive with 85% of participants are willing to recommend the tool to their peers. The artifacts related to *PyCodeViz* are available at https://anushmangal01.github.io/PyCodeVizDoc/ and demonstration of the tool can be found at https://youtu.be/5mOQt8hvtxY.

CCS CONCEPTS

• **Software and its engineering** → Software notations and tools; • **Social and professional topics** → *Computing education*; • **Human-centered computing** → **Visualization systems and tools**.

KEYWORDS

program comprehension, python tutor, code visualization

ACM Reference Format:
Anush Mangal, A. Eashaan Rao, and Sridhar Chimalakonda. 2022. PyCodeViz: An Interactive Visualization Tool to Facilitate Program Comprehension. In *ICPC '22:IEEE/ACM International Conference on Software Engineering, May 16–17, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

*Equal contribution

1 INTRODUCTION

Program comprehension among novice developers has always been a subject of discussion and research. One of the primary focus in this area is to aid the novice developers to understand the basic programming constructs and comprehend source code at the lowest level of detail [14]. However, during the initial stages, novice developers find it challenging to grasp the inner functioning and execution of basic constructs on their own [3, 12]. These constructs generally comprise basic data structures, conditional and iterative statements. Having a solid foundational base on these programming constructs will help them in writing a better code during the software development [14].

In the literature, numerous tools are developed to facilitate the comprehension aspect of the novice developers [5, 6]. These tools try to provide interactive environments by imparting either step-by-step execution of the code [5, 13] or improving the code error messages [11]. Especially in visualizing code execution, tools such as ViLLE [13], Visualgo [6], Jype [8], and Jeliot 3 [1] aim to visualize code execution and events that happen at the runtime. Thus, designing a programming environment that fosters an understanding of code execution facilitates program comprehension among the novice developers [9, 11]. For instance, novice developers often unable to keep track of the variables and their changing values in recursive and iterative statements. To aid such instances, this paper proposes *PyCodeViz*, a tool to comprehend the code written in the Python at a more granular level.

With Python gaining popularity as one of the preferred choices of programming language among novice developers [16], tools and environments are proposed in the literature, which help developers to comprehend Python programs [2, 5, 8]. For instance, *Pythy* [2] is a web-based Python programming environment which supports live-interacting code sessions, tutorials, assignments and grading. Python Tutor is one of the most widely known and used tool that aids in program comprehension [5]. Since its emergence, in over a decade, more than ten million users from 180 countries have used it as a learning tool to improve their understanding of Python and other programming languages [4].

The common premise in the tools mentioned above is the use of different kinds of visualization to enhance the comprehension of program execution [4]. For instance, Python Tutor brings simplicity in comprehending code by supporting the step-by-step visualization of its execution. However, it does not focus on visualizing basic constructs such as iterative and conditional statements in a detailed

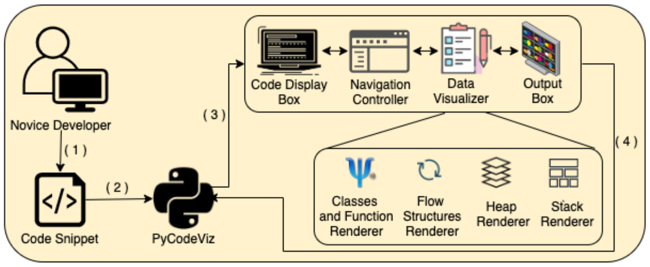


Figure 1: Design of *PyCodeViz*.

manner, which novice developers extensively use [14]. In addition, transitions happening at the code lines or on the values of different types of variables due to program logic needs to be visualize to facilitate better comprehension.

Along with the program comprehension at its core, we firmly believe that there should be a continuous improvement in the visualization aspect. Visualizing code execution in real-time engages the learner and helps them to learn the finer execution details of programming constructs in a faster way [15]. Therefore, main focus of *PyCodeViz* would be to facilitate the program comprehension on Python programs with the help of enhanced visualization.

Thus, we propose **Py**thon **Code Vi**suali**z**er (***PyCodeViz***), to address some of the above challenges of Python Tutor. Its features include visualizing iterative and conditional statements, providing details where the conditions are '*true*' or '*false*'. For iteration, to help novice developers trace the changes in each step, associated changes in the variables are shown explicitly. Much emphasis is given to improving the visualization of recursive functions, which is considered as a hard-to-understand programming construct [7]. It also offers a vibrant user interface with different color-coding for different data types and code lines, which results in better comprehension, user experience [10], and easy segregation of different types of statements. It displays the code frequency of code lines, i.e., dark code line number indicates high recurrence of those code lines in the execution. Hence, *PyCodeViz* tries to support novice developers by visualizing the execution steps for better comprehension.

2 PYCODEVIZ

In this section, we discuss the design, functioning and implementation of *PyCodeViz*.

The tool intends to target novice programmers. Therefore, the user interface is modified to keep it simple and easily navigable. The overall structure is based on the original Python Tutor. *PyCodeViz* has an Execution Visualiser at the core front-end of the tool. The Execution Visualiser handles the front-end's four main sections: the Navigation Controller, the Data Visualiser, the Code Display Box, and the Output Box. Figure 1 shows the different components developed in *PyCodeViz*, which are described in detail below:
Navigation Controller is used to move back and forth the execution steps to see the functioning.
The **Data Visualiser** section has four components:

- **Classes and Functions Renderer**: This column shows all the classes and their attributes and the functions in the code. In the function attributes, it shows whether the function is recursive or not.
- **Stack Renderer**: This column is titled "Frames." It has a global frame section that shows all the values of the global variables. It also shows the previous value of the variables if they change so that the user can keep track of changes. The variables have been color-coded based on their data types. The color-coding will help the novice programmers to understand and visualize type casting: both implicit and explicit. The color-coding legend is present below the Code Display Box. This column also has a stack that shows different function calls, the parameters' values, and the variables associated with them. In the case of recursive functions, it shows the iteration number of the recursive call in the same stack.
- **Heap Renderer**: This column is titled as "Objects". It shows the complex data types like lists, dictionaries, maps, and instances using JsPlumb Connector Lines. These lines originate from the frames section and point to the respective object.
- **Flow Structures Renderer**: This column is titled "Loops and If." It shows FOR and WHILE loops and their number of iterations. It also shows IF conditions and ELIF conditions and whether they are true or false.

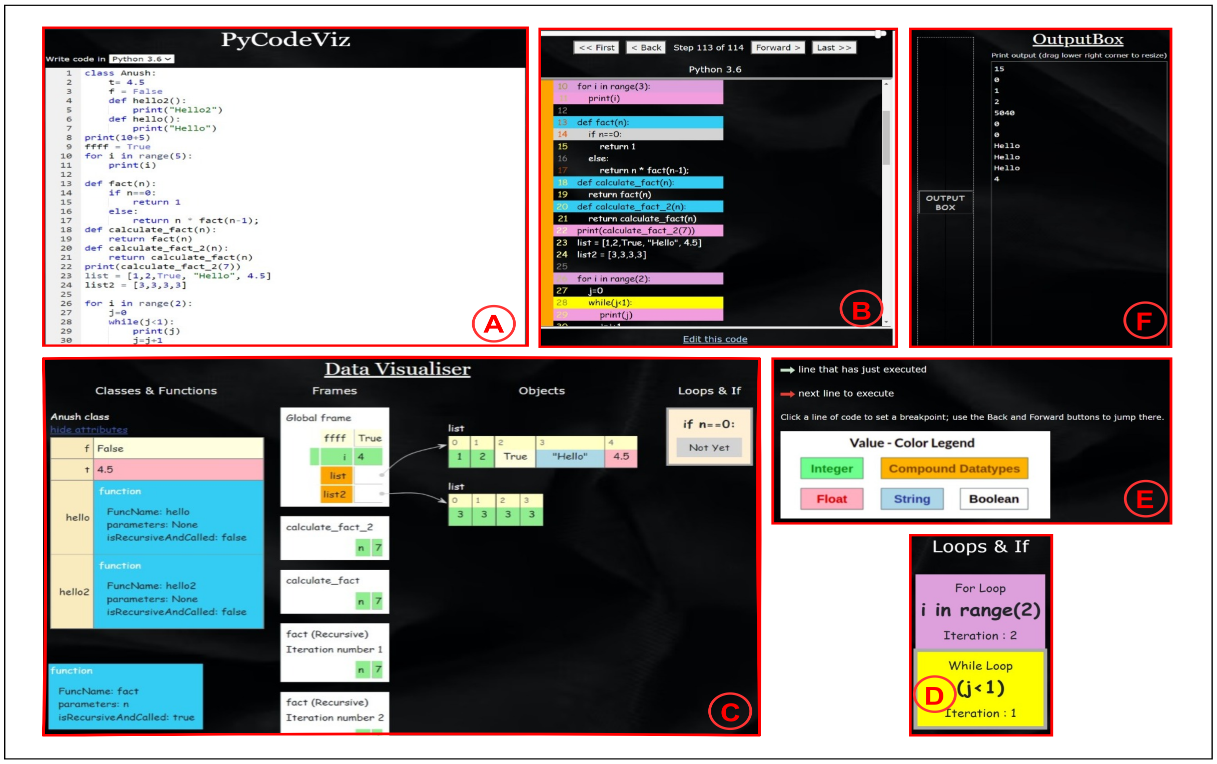
Code Display Box: We have made the user interface fairly vibrant to engage the users' attention by color-coded the code lines in the Code Display box. It depicts the execution frequency of a particular line of code.

Output Box: The output box displays the program's output given by print statements in the code. The user can hide it or show it according to their convenience.

The complete front-end script is mainly written in Javascript in the visualize.bundle.js file in v5-unity/bundle/ folder. We have added a separate function that renders the loops and if statements. It looks for all the active parent loops and if conditions by checking their indentation. It then stores them in a list named scope stack and displays them. Other functions render primitive data types, complex data types, functions, and classes.

The complete back-end of *PyCodeViz* is in Python. One file encodes all the constructs into objects. It adds type labels such as primitive, dictionary, map, lists and tuples to different variables. In addition, it also labels objects and classes. The tool uses standard python debugger framework bdb in the back-end. It generates the complete execution trace of the program in JSON format. The visualize.bundle.js file then uses this JSON trace to visualize the information. The trace contains practically all the information needed. The trace fields are:

- *line*: It stores the line number of the code line.
- *event*: it tells whether the line is a standard step line, a return statement or a function call.
- *func_name*: It tells the function inside which it is working. If it is the global scope, then the function name is "<modulo>".
- *globals*: It stores the variables inside the function and their values. It also stores any complex data types.
- *ordered_globals*: it stores only the variable names.



```
class Anush:
    t= 4.5
    f = False
    def hello2():
        print("Hello2")
    def hello():
        print("Hello")
print(10+5)
ffff = True
for i in range(5):
    print(i)

def fact(n):
    if n==0:
        return 1
    else:
        return n * fact(n-1);
def calculate_fact(n):
    return fact(n)
def calculate_fact_2(n):
    return calculate_fact(n)
print(calculate_fact_2(7))
list = [1,2,True, "Hello", 4.5]
list2 = [3,3,3,3]

for i in range(2):
    j=0
    while(j<1):
        print(j)
        j=j+1
```

```
15
0
1
2
5040
0
0
Hello
Hello
Hello
4
```

Figure 2: Snapshot of *PyCodeViz* tool. In [A], the user enters the code for visualization. [B] shows the code snippet in different colour coding and [E] shows legends used in the visualization. After analysing the code snippet, [C] i.e., Data Visualizer displays different programming constructs separately, such as variables, data structure, and recursive stack, [D] displays visualization for iterative and conditional statements and [F] shows the output of the code snippet in a separate output box.

- *stack_to_render*: It stores the function call stack, if any.
- *heap*: It stores complex data types such as lists, maps, dictionaries and tuples. It also contains all the functions and classes to show.
- *stdout*: Everything to show in the output box.

In the case of recursive functions, we explicitly add one more attribute in the trace element "iteration." It stores the number of recursive iterations of the recursive function.

3 USAGE SCENARIO

Consider *Veda*, a novice developer who wants to learn to program in Python. Since she is a beginner, she is more prone to make mistakes. She understands some of the basic programming constructs but does not comprehend how these constructs run at runtime. Therefore, she wants to understand the step-by-step execution of the code snippet. She gets to know about *PyCodeViz* which can visualize the program execution for a Python code snippet and starts using it to understand the Python program.

She types the program she wants to understand in the Code Input Box (as shown in Figure 2[A]) and clicks on “Visualize Execution” bottom of the box (not shown in the image). She then sees the Execution Visualizer and its different sections: the Code Display box and the Navigation Controller (Figure 2[B]), the Legend (Figure 2[E]), and the Output box (Figure 2[F]). She uses the navigation buttons provided to move back and forth the execution steps at her own pace. This color codes the code lines in the Code Display Box and displays the visualization in Data Visualizer (Figure 2[C]).

After spending some time on *PyCodeViz*, Veda can understand the program. She is now able to understand the concept of classes and their attributes and behavior. She can appreciate the recursive functions and recursive calls written in the program and how it is run in the background. She can see the loops’ execution and their iteration numbers, trace the changes that happened in variables during the iteration, and comprehend the output generated based on the logic written in the program.

4 EVALUATION

We followed a similar approach of in-user survey used by Zhang et al. in [17] to evaluate *PyCodeViz* with 11 volunteers in the age group of 18 to 20 years. All of them were boys and undergraduate computer science students who were in their third semester and had around four months of programming experience.

4.1 Evaluation Criteria

All volunteers were first requested to set up *PyCodeViz* from Github on their systems. The repository’s readme file specifies the steps to be followed to install and run it on the local system. A demonstration has been given to them on using the tool, and all features

No.	Question
Q1	How did you like the visualization of the *PyCodeViz* interface?
Q2	According to you, *PyCodeViz* has shown detailed information related to each execution step satisfactorily.
Q3	*PyCodeViz* has helped me to comprehend the code in a detailed manner.
Q4	*PyCodeViz* has helped me to understand the execution of programming constructs such as iterative, conditional statements and recursive functions, which helped me in getting better insights about the logic of the program.
Q5	Does *PyCodeViz* show the explicit changes in the values of different variables?
Q6	I will recommend *PyCodeViz* to my peers.
Q7	Please suggest how we can improve the *PyCodeViz*?

Table 1: Questions for the user-study and Results for Q1-Q6

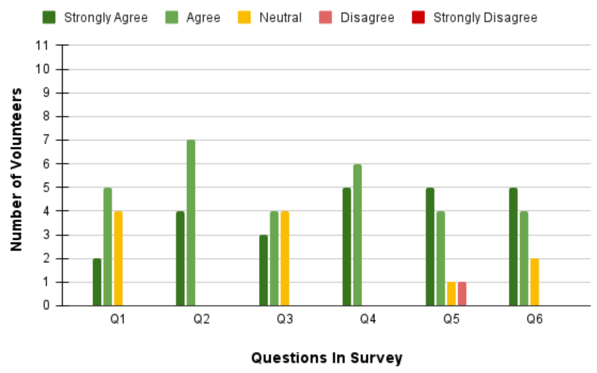


Figure 3: Results of User Survey

are thoroughly explained. After the demonstration, all participants received different code snippets, which they were needed to analyze using the tool and explain the objective achieved by the code snippet. If they understood the code snippet, the participants were requested to navigate the user survey link. The questionnaire as mentioned in Table 1 consists of seven questions asking the volunteers to describe their experience and usability of *PyCodeViz*. Six questions could be answered using the 5-point Likert scale and the seventh question was open-ended.

4.2 Results

The tool received good response where all participants agreed that *PyCodeViz* visualization helps them to understand the logic behind the code snippets in a better way, with 85% of them willing to recommend the tool to their peers. We also used one open-ended question to understand the views of participants on *PyCodeViz* and a few interesting responses include:

- "*The UI can be improved a bit. Recursive extension is good. Will recommend to peers.*"
- "*The iterative steps, if beyond a limit should be compressed so its easier to navigate and understand.*"
- "*To further improve user's experience, file upload option can be provided where user can directly upload his code and need not copy paste the code.*"
- "*Try to add advance python functionalities.*"
- "*The UI format can be more descriptive of what each section is trying to show.*"

These responses suggest that the participants validated the idea of step-by-step representation of data-flow within the program and also welcomed the interface provided for better understanding of the summary. Results of the survey can be found here[1].

5 DISCUSSION AND FUTURE WORK

PyCodeViz is a program comprehension tool aiming to facilitate the visualization and showcase the finer details of program execution to aid novice developers. There is a lot of scope for improvement in *PyCodeViz* which needs to be highlighted. All the features from Python Tutor have been retained, because of which it cannot visualize more than a thousand execution steps. In subsequent works, we would like to increase the execution steps limit. At the current stage, we believe the limit is enough for most novice developers who want to visualize a particular code snippet and the logic behind it, rather than the complete program. Another aspect which needs to be enhanced is the inclusion of more python in-built modules that are used in many Python projects. Our aim is to make *PyCodeViz* as a general purpose program comprehension tool, i.e., complex code written in Python projects can also be visualised, which may facilitate the program comprehension among the developers.

To visualize recursive functions, the attribute named "recursive and called" is added for the functions that gives correct output only for the primitive recursive functions and not the mutually recursive functions, i.e., functions that call another functions which in turn call the first function. *PyCodeViz* will not show a function as recursive if it is not called during the execution. In the subsequent versions, this feature will be added along with separate highlight message for the dead code.

If there are multiple nested loops and if statements, which include function calls, and the called function itself contains loops and if statements, the visualization of such scenarios are not properly rendered by *PyCodeViz*. Hence, we plan to include different visualization techniques, where complex relationships in the code can be properly depicted. *PyCodeViz* currently supports novice programmers and helps them understand flow structures and object references. Therefore, developers cannot use it in software development due to its inability to support multiple modules.

6 CONCLUSION

To aid novice developers in understanding low-level execution details and the inner working of a program, we proposed *PyCodeViz*. The tool visualizes the flow structures and recursive functions and focuses on a rich user experience by engaging the user through a vibrant user interface and motivating them to learn better. We have evaluated *PyCodeViz* with 11 university students, using a 5 point Likert scale-based questionnaire, through an in-user survey. All participants reported positive experience with the tool and agreed to recommend *PyCodeViz* to their peers.

[1]https://github.com/rishalab/PyCodeViz/tree/main/Results

REFERENCES

[1] Sanja Maravić Čisar, Robert Pinter, and Dragica Radosav. 2011. Effectiveness of program visualization in learning java: a case study with jeliot 3. *International Journal of Computers Communications & Control* 6, 4 (2011), 668–680.

[2] Stephen H Edwards, Daniel S Tilden, and Anthony Allevato. 2014. Pythy: Improving the introductory python programming experience. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 641–646.

[3] Sarah Fakhoury, Devjeet Roy, Adnan Hassan, and Vernera Arnaoudova. 2019. Improving source code readability: Theory and practice. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2–12.

[4] Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 1235–1251.

[5] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 579–584.

[6] Steven Halim. 2015. Visualgo–visualising data structures and algorithms through animation. *Olympiads in Informatics* (2015), 243.

[7] Sally Hamouda, Stephen H Edwards, Hicham G Elmongui, Jeremy V Ernst, and Clifford A Shaffer. 2018. Recurtutor: An interactive tutorial for learning recursion. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–25.

[8] Juha Helminen and Lauri Malmi. 2010. Jype-a program visualization and programming exercise tool for Python. In *Proceedings of the 5th international symposium on Software visualization*. 153–162.

[9] Ahmad Jbara, Or Shacham, Bar Ben Michael, and Omer Tavor. 2020. Simply-Hover: Improving Comprehension of else Statements. In *Proceedings of the 28th International Conference on Program Comprehension*. 456–460.

[10] Hyeonsu Kang and Philip J Guo. 2017. Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 737–745.

[11] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1054–1060.

[12] Greg L Nelson, Benjamin Xie, and Andrew J Ko. 2017. Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 2–11.

[13] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. 2008. Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education* 7 (2008).

[14] Amal A Shargabi, Syed Ahmad Aljunid, Muthukkaruppan Annamalai, and Abdullah Mohd Zin. 2020. Performing tasks can improve program comprehension mental model of novice developers: An empirical approach. In *Proceedings of the 28th International Conference on Program Comprehension*. 263–273.

[15] J Ángel Velázquez-Iturbide, Isidoro Hernán-Losada, and Maximiliano Paredes-Velasco. 2017. Evaluating the effect of program visualization on student motivation. *IEEE Transactions on Education* 60, 3 (2017), 238–245.

[16] Jacques Wainer and Eduardo C Xavier. 2018. A controlled experiment on Python vs C for an introductory programming course: Students' outcomes. *ACM Transactions on Computing Education (TOCE)* 18, 3 (2018), 1–16.

[17] Tianyi Zhang, Di Yang, Crista Lopes, and Miryung Kim. 2019. Analyzing and supporting adaptation of online code examples. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 316–327.