Evaluating the Effects on Comprehension in Python Code from Idiomatic Development

JOSHUA A.C. BEHLER, Kent State University, USA

This work outlines an eye-tracking study designed to assess the readability of idiomatic versus non-idiomatic Python code among both novice and expert developers. Idioms are an important part of Python's design and developer community. However, no prior research has rigorously evaluated whether these idiomatic constructs enhance comprehension compared to their non-idiomatic counterparts. This study will address this by measuring the comprehension of idiomatic and non-idiomatic Python code to determine if there is a significant difference.

Advisor: Jonathan I. Maletic (Kent State University)

CCS Concepts: • **Software and its engineering**; • **Human-centered computing → Empirical studies in HCI**;

Additional Key Words and Phrases: eye-tracking python

ACM Reference Format:

Joshua A.C. Behler. 2025. Evaluating the Effects on Comprehension in Python Code from Idiomatic Development. In *2025 Symposium on Eye Tracking Research and Applications (ETRA '25), May 26–29, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3715669.3727335

1 INTRODUCTION

Python is an immensely popular programming language. It consistently tops various language popularity charts, such as the TIOBE Index[1] or the IEEE Spectrum Top Programming Languages list [2]. Due to its popularity, Python has an extremely dedicated developer community. Over the years, the Python developer community has developed a list of idioms - code snippets with a specific semantic purpose - that they consider the recommended and proper way of using Python [Alexandru et al. 2018; Farooq and Zaytsev 2021; Zhang et al. 2022].

The community dubs these idioms as being "Pythonic". Pythonic code is referred to throughout literature and programming guides. In their 2018 work, Alexandru et al. lays the foundation of Python idiom research by compiling commonly used ideas and recommendations from numerous sources [Alexandru et al. 2018].

These idioms provide a great insight into the analysis of how Python developers write and read code. However, there is very little research that looks at how developers read and understand Python. As a result, there is no empirical evidence to support the widespread use of Pythonic standards beyond the general consensus from developers. This is a fundamental problem, as Python education continues to focus on promoting these standards without evidence showing they are better than alternatives. In an attempt to address this issue, I propose an eye-tracking study on Python code to determine whether the use of Pythonic idioms provide tangible benefits to program comprehension.

[1] https://www.tiobe.com/tiobe-index/
[2] https://spectrum.ieee.org/the-top-programming-languages-2024

Author's address: Joshua A.C. Behler, Kent State University, Kent, Ohio, USA, jbehler1@kent.edu.

Manuscript submitted to ACM

2 RELATED WORKS

2.1 Eye-tracking on Python

There have been a couple of studies which utilize eye-tracking on Python stimulus. For instance, Roberto et al. explores how effective Python's official style guide is for maintaining readability in programming novices [Roberto et al. 2024]. Python's style guide, PEP 8 [van Rossum et al. 2013], details style recommendations for the entire language, including spacing, naming, and formatting. In their study, they explore how readable various guidelines are with novices, finding that some guidelines reduce fixation duration and decrease the number of regressions.

Turner et al. compare the comprehension of C++ code to Python code from undergraduate and graduate students [Turner et al. 2014]. Their results show that there is little difference between languages in how accurate participants were in understanding the provided, or in how long it took the participants. There was a difference in how much participants fixated on buggy lines of code, with buggy Python code being slightly easier to identify.

Segedinac et al. perform a similar study by investigating the readability of specific Python statements, and how much they correlate to comprehension [Segedinac et al. 2024]. They collected data from 90 undergraduates and evaluated common statements taught in a programming course. A machine learning model was then created to predict how readable a Python file is based on the statements used. The results from the study were used to create recommendations for future teaching.

2.2 Python Idioms

There is limited use of the Python idioms to evaluate Python code. Notably, Zhang et al. use Python idioms to evaluate whether the use of idioms improves the efficiency of Python code [Zhang et al. 2023]. Their findings show that Python idioms often do not improve performance in real-world settings, and can even decrease performance. This is in contrast to the common perception of idioms improving the performance of code, such as stated in [Alexandru et al. 2018].

It is also commonly believed that Python idioms improve the readability of Python code. However, I am aware of no work that explicitly examines whether Python idioms are more readable than their non-idiomatic counterpart.

3 OVERVIEW OF PROPOSED WORK

To better understand the usefulness of Python idioms within programming, I propose a study which examines how well Python developers comprehend idiomatic code versus non-idiomatic code. Specifically, I plan to examine all levels of Python developers, including both complete beginners (novices) and industry professionals with years of experience working in Python (experts). Using academic and professional connections, I plan to conduct this study on a wide breadth of individuals so ensure the most conclusive results.

This study aims to answer the following research questions:

RQ1 Do the most commonly used Python idioms improve comprehension compared to non-idiomatic forms?
RQ2 Do Python idioms allow for Python novices to comprehend unfamiliar code?
RQ3 Does having pre-existing knowledge of a Python idiom correlate to a larger comprehension gap between the idiomatic and non-idiomatic version of the Python code?

3.1 Study Design/Method

Participants in the study will be recruited in two main categories: novices from my institution and experts from industry in the local area. Processing each participant will take roughly an hour each. Participants will be sat at a

```python
with open("eye_gazes.json",'r') as input_file:
    data = json.loads(input_file.read())

print(data["gazes"])
```

```python
input_file = open("eye_gazes.json",'r')
data = json.loads(input_file.read())
input_file.close()

print(data["gazes"])
```

Fig. 1. An example of an idiomatic (top) and non-idiomatic (bottom) implementation of opening and reading a file. The idiomatic implementation makes use of Python's with statement, and is known as context management.

height-adjustable table that is adjusted to their height in a simple room with little external stimulus. A single monitor with the eye tracker will be on the table, and the participant will be provided a mouse and keyboard. Adjacent to the participant, a researcher will be overviewing the study. The researcher will have their own monitor which displays the Tobii eye-tracker manager, and will ask the participant to adjust themselves if they become unaligned with the eye-tracker.

Each participant will be first given a pre-questionnaire which collects demographic information. Additionally, each participant will be given a Python knowledge quiz. This quiz will ask participants how much experience they have working with Python, how comfortable they are reading Python, and will list various Python idioms and ask if they are familiar with the syntactic feature. This quiz will not preclude anyone from participating - rather, it is used for gathering data related to any pre-existing knowledge of a Python feature.

Afterwards, the participant will be shown a series of Python files, each focusing on some specific idiomatic or non-idiomatic implementation. The participant will be asked to read each file, determine what the function does, and then answer some questions on the shown code. Figure 1 showcases example snippets of idiomatic and non-idiomatic Python code that the participant would look at during the study.

To measure their comprehension, heuristics like the number and duration of fixations, the number of line regressions, and the syntactic distribution of gazed tokens will be used to determine how much cognitive effort was needed to understand the code.

To accomplish this, I will make use of the iTrace Infrastructure. iTrace recently released support for the JetBrains family of IDEs. This means that PyCharm, a very popular IDE which specializes in Python, can be used for conducting eye-tracking research. Additionally, srcML [Collard et al. 2011, 2013], which iTrace uses to support the collection syntactic context information from the source code, recently released beta support for Python. While Python for srcML is still a work in progress, it is more than sufficient for the simple Python programs that will be used in this study.

3.2 Envisioned Contributions

This results of this study will have impactful implications for the general Python development community. As discussed by Alexandru et al., the Python community has a large affinity for writing code the "Pythonic" way. Previous works show that the performance improvements of writing Pythonic code are limited. This work will help to answer if there is any comprehension benefit to Pythonic programming.

Similarly, this work will have major impacts in education. As Python continues to grow in popularity, more and more institutions, coding camps, and online tutorials are adopting Python. The results from this work will help dictate what the best practices are for teaching and writing Python. Industry will also be affected, as this work will provide evidence for what or what not to include in internal coding requirements.

4 SUMMARY

The proposed study aims to provide empirical evidence on the comprehension of common idioms Python developers use versus non-idiomatic equivalents. Results gathered from the study will advance general understanding of how Python developers read programs, and will provide evidence either for or against the general Python developer community's focus on Pythonic programming.

REFERENCES

Carol V Alexandru, José J Merchante, Sebastiano Panichella, Sebastian Proksch, Harald C Gall, and Gregorio Robles. 2018. On the usage of pythonic idioms. In *Proceedings of the 2018 ACM SIGPLAN international symposium on new ideas, new paradigms, and reflections on programming and software*. 1–11.

M. L. Collard, M. J. Decker, and J. I. Maletic. 2011. Lightweight Transformation and Fact Extraction with the srcML Toolkit. In *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*. 173–184. https://doi.org/10.1109/SCAM.2011.19

M. L. Collard, M. J. Decker, and J. I. Maletic. 2013. srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration. In *2013 IEEE International Conference on Software Maintenance*. 516–519. https://doi.org/10.1109/ICSM.2013.85

Aamir Farooq and Vadim Zaytsev. 2021. There is more than one way to zen your python. In *Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering*. 68–82.

Pablo Roberto, Rohit Gheyi, José Aldo Silva da Costa, and Márcio Ribeiro. 2024. Assessing Python Style Guides: An Eye-Tracking Study with Novice Developers. arXiv:2408.14566 [cs.SE] https://arxiv.org/abs/2408.14566

Milan Segedinac, Goran Savić, Ivana Zeljković, Jelena Slivka, and Zora Konjović. 2024. Assessing code readability in Python programming courses using eye-tracking. *Computer Applications in Engineering Education* 32, 1 (2024), e22685.

Rachel Turner, Michael Falcone, Bonita Sharif, and Alina Lazar. 2014. An eye-tracking study assessing the comprehension of c++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (Safety Harbor, Florida) *(ETRA '14)*. Association for Computing Machinery, New York, NY, USA, 231–234. https://doi.org/10.1145/2578153.2578218

Guido van Rossum, Barry Warsaw, and Nick Coghlan. 2013. PEP 8 – Style Guide for Python Code | peps.python.org. https://peps.python.org/pep-0008/#function-and-variable-names

Zejun Zhang, Zhenchang Xing, Xin Xia, Xiwei Xu, and Liming Zhu. 2022. Making python code idiomatic by automatic refactoring non-idiomatic python code with pythonic idioms. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 696–708.

Zejun Zhang, Zhenchang Xing, Xin Xia, Xiwei Xu, Liming Zhu, and Qinghua Lu. 2023. Faster or slower? performance mystery of python idioms unveiled with empirical evidence. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1495–1507.