How To Zen Your Python

Aamir Farooq
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.a.farooq@student.utwente.nl

ABSTRACT

Although the popularity of Python is frequently attributed to its concept of *pythonicity*, Alexandru et al. claim that until recently few have attempted to formally define it. They contend that they are the first, and to do so, they interviewed various experienced developers, conducted a literature review to discover *pythonic idioms*, and deduced usage statistics for the idioms in popular Python projects through automated detection. Despite Python being one of the most popular programming languages right now, there is a lack of empirical evidence to explain the phenomenon of pythonicity, and while Alexandru et al. appropriately defined this notion, their work is incomplete. This research paper brings the work that Alexandru et al. set out to do closer to completion by providing an extended list of pythonic idioms, as well as statistics on how pythonic idiom usage has evolved over time.

Keywords

Pythonic, Python, idioms, conventions, community, programming

1. INTRODUCTION

1.1 Background

A programming language is not just its syntax and its vocabulary, but also a set of known effective ways to solve actual problems with it. There exists a well-studied category of the conventions and idioms in programming languages such as Java [2, 10, 29], which can take the form of implementation patterns, formatting rules, calling conventions, naming conventions, etc. Such conventions are referred to as *idioms* in the software language field, and Alexandru et al. formally define this term as a language feature or "reusable abstraction" that can improve the quality of code [1].

Much like with other languages, the same concept exists in the Python community, and Python developers call code *pythonic* when such idioms are used. The *pythonicity* of a piece of code stipulates how concise, easily readable, and in general terms, "good" the code is.

While the concept of conventions and idiom usage exists in other languages, it is especially pronounced in the Python community. There is a general "feeling" among the community that it goes beyond a set of practices, rather it is a philosophy that the community strives to uphold. Python developers are in the constant pursuit of upholding the so-called *Zen of Python* rules, such as "*There should be one — and preferably only one — obvious way to do it.*", and "*Beautiful is better than ugly. [...] Simple is better than complex.*" [17].

Given a piece of code, any experienced Python programmer can easily tell whether it is pythonic or not. Sakulniwat et al. were able to demonstrate, in a case study of the `with open` idiom, that over time developers tend to adopt idioms to improve their codebase [21], and experienced developers stated in the interviews conducted by Alexandru et al. that year after year, their code became more pythonic [1]. However, to complete programming novices or newcomers to Python, as Alexandru et al. also contend, it is not completely obvious how to incorporate the so-called *pythonic idioms* in their code [1]. In their study, many interviewees also indicated that junior Python programmers can even be distinguished from more experienced ones simply by observing the usage of pythonic idioms, and further, the interviewees agreed that they learned pythonic code from experience — from reading books, source code from other projects and StackOverflow responses [1].

As such, Alexandru et al. identified a lack of research in the phenomenon of pythonicity as they felt that there was no clear definition as to what "pythonic" means and what should developers do to make their code pythonic. They conducted a literature review to identify the pythonic idioms from numerous sources such as *The Zen of Python* [17], *Writing Idiomatic Python* [9], *The Hitchhiker's Guide to Python* [20], *Effective Python* [24], *The Little Book of Python Anti-Patterns* [18], as well as direct interviews with developers with varying levels of expertise. Moreover, they wrote an idiom detection library to corroborate with empirical evidence that idioms were actually in use in 1,000 of the most popular open-source Python projects on GitHub.

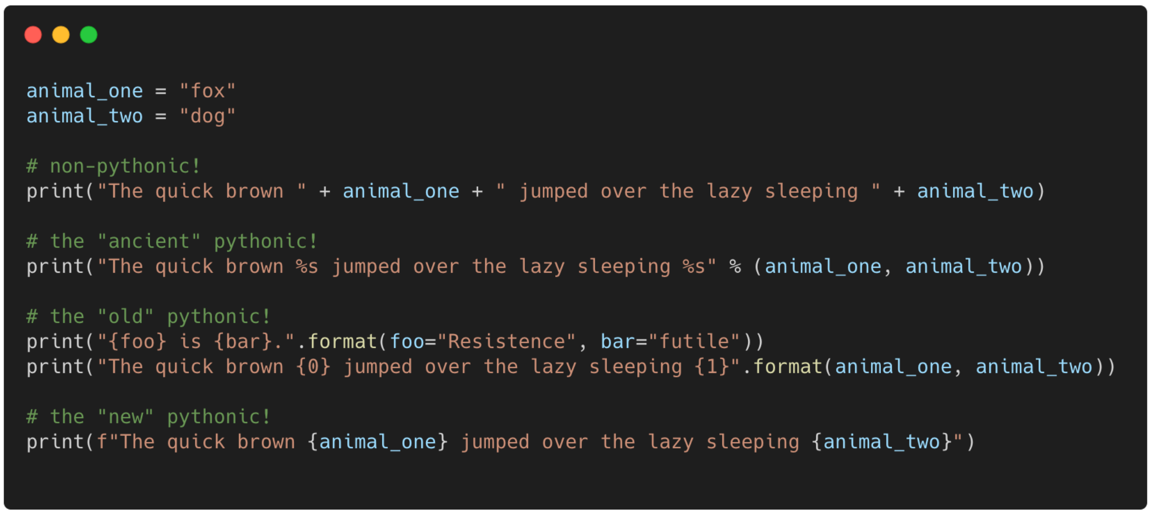
1.2 Related work

Despite Python being among the most popular programming language on GitHub right now according to the PYPL index [6], the authors of the original paper claim to be the first to attempt forming a tangible definition and catalog of what constitutes pythonic code. At the time of writing, we were only able to identify one other paper by Sakulniwat et al. [21] which attempts to improve upon their results.

The paper from Alexandru et al. was published in 2018, along with a catalog of idioms[1] and a repository with the idiom detection code, which makes use of the LISA library[2].

[1]Online: `https://pythonic-examples.github.io/`
[2]LISA library: `https://bit.ly/3xSFg1m`

35th Twente Student Conference on IT July. 2nd, 2021, Enschede, The Netherlands.



```python
animal_one = "fox"
animal_two = "dog"

# non-pythonic!
print("The quick brown " + animal_one + " jumped over the lazy sleeping " + animal_two)

# the "ancient" pythonic!
print("The quick brown %s jumped over the lazy sleeping %s" % (animal_one, animal_two))

# the "old" pythonic!
print("{foo} is {bar}.".format(foo="Resistence", bar="futile"))
print("The quick brown {0} jumped over the lazy sleeping {1}".format(animal_one, animal_two))

# the "new" pythonic!
print(f"The quick brown {animal_one} jumped over the lazy sleeping {animal_two}")
```

Figure 1: An example of a new pythonic idiom Alexandru et al. did not cover, known as f-strings, a much less cumbersome and more readable approach to traditional string formatting methods [26].

However, the list of idioms is not complete. The experiment was conducted before 2018, which coincides with the release of Python 3.7. Since then, Python 2 has also been officially deprecated [13], and several major Python versions have been released (at the time of writing, the most recent version is 3.9.4), each of which adds a number of features to the language [14]. There is obviously some adoption time for newer versions, and for these reasons, there may have been significant shifts in the popularity of idiom usage; one such idiom is seen in Figure 1. It is also known that even at the time of writing, the list of idioms in the paper of Alexandru et al. was, as they say, "inexhaustive" [1], so it can be extended to cover a larger set of idioms.

Researching this topic is crucial so that software languages can continue to improve and move forward. One initiative is the Software Language Engineering Body of Knowledge[3] (*SLEBoK*), which makes an effort to compare and consolidate the implementation of features and paradigms across programming and software languages. In doing so, the developers of software languages may identify discrepancies between their language and others, and then improve their own feature set.

An additional application is technical debt remediation in Python. Feltosa et al. describe the notion of *technical debt* as the result of cutting corners in the short term on the "long term sustainability" of the software project [27]. As pythonic code is considered generally more maintainable, efficient, and overall state-of-the-art, it suffices to say that being able to detect the usage of such idioms would go a long way in quantifying code quality. A potential future application of the results of this paper could be automated detection of *anti-idioms*[4], or malpractices, in the pursuit of preventing technical debt from accumulating in the first place. A similar practice is widespread and accepted as useful in other languages, such as Java [11, 28].

[3]SLEBoK: http://slebok.github.io/
[4]Online: http://omz-software.com/editorial/docs/howto/doanddont.html

As Shull et al. explain, replicating results of empirical studies in software engineering is key in proving their veracity, citing the difficulty of extrapolating results due to "uncontrollable sources of variation from one environment to another" [23]. The same holds here; the efforts of Alexandru et al. need to be verified through an external replication.

As such, the contributions of this paper will initially be an extended catalog of pythonic idioms rooted in a literature review, followed by a replication of Alexandru et al.'s experiment. We then go beyond the replication by extending Alexandru et al.'s detection library to detect a subset of our newly discovered idioms. Further, we analyze usage statistics of a selection of the idioms to generate new insights about the popularity of pythonic idioms in open source Python projects, as well as how the usage has evolved over time.

2. RESEARCH QUESTIONS

To guide our research, we devised the following research questions which by the end of this paper, we intend to answer or comment on. Based on the sentiment from the developers Alexandru et al. interviewed that they do not go back and make their old code pythonic [1], we hypothesize that since the publishing of the results from Alexandru et al., the popularity of each idiom they identified has not changed.

1. *What idioms should be included in an updated, extended catalog of pythonic idioms?*
   By updating the catalog of idioms that Alexandru et al. already found based on a literature review from Python books, we can form a more complete picture of what idioms make code pythonic.

2. *How widely adopted are the new idioms that we discovered?*
   We will also need to find empirical evidence to support the claim that these newly documented idioms are accepted as pythonic in the Python community,

as described in the next question. This means extending the idiom detection code of the original authors to include the newly found idioms and analyzing the statistics we find.

3. *How has the usage of pythonic idioms evolved in software projects over time?*
   As stated previously, some years have passed since the experiment of Alexandru et al. From the idioms they found, it could be that certain idioms have gone out of style and other, possibly new, idioms have become more popular. By answering this question, we can provide empirical evidence to not only support the results of **RQ2** but also to comment on our hypothesis.

3. LITERATURE REVIEW

With the literature review, we intend to provide an answer for **RQ1**. The goal is to not only confirm the idioms that Alexandru et al. were able to identify but to further discover new pythonic idioms as well as idioms that were not covered in their research.

To discover our idioms, we made use of grounded theory in a bottom-up approach: searching the internet for the most popular Python books, then scanning literature based on a set of keywords and cross-referencing the results across books. As such, we are confident that our methodology leads to uncovering all of the most commonly used pythonic idioms since the findings are rooted in a large variety of the literature available.

The literature sources were uncovered by searching the internet using key terms such as:

- *python tricks book*
- *python cookbook*
- *books "pythonic"*
- *books "idioms" "python"*

The results we found were programming blog posts, REDDIT threads, and STACKOVERFLOW questions where users provided their favorite Python books. We took note of the books that were talked about the most across these sites (as well as which responses were upvoted the most) and created a list of books, articles, and conferences discussing pythonic idioms.

From all the books we were able to identify, we first eliminated the "complete beginner" books because after reviewing them, we discovered that they focus on the fundamentals of programming in general and introducing syntax. This is not appropriate for our research, as opposed to books covering good programming practices. We also eliminated some "advanced" books which tend to cover Python for very specific applications and patterns, for example, data science. These are also not appropriate for our research because we want to find generalized results about the Python language as a whole rather than idioms that are only used in domain-specific applications.

The optimal balance we found was with "intermediate-level" books which assume that readers have prior programming knowledge of some form and generally understand the Python syntax, but want to improve their Python skills. Each book here made some form of reference to pythonicity, programming patterns, and idioms in the description or blurb.

From the selection process, we started with the books *Python Tricks: A Buffet of Awesome Python Features* [4], *Practical Python Design Patterns: Pythonic Solutions to Common Problems* [3], *Learn Python The Hard Way* [22], *Python Cookbook, Third Edition* [7], and *Effective Python: 90 Specific Ways to Write Better Python* [25]. We also reviewed several online sources, such as blog posts, which we used to confirm our previously found idioms rather than to identify new ones.

We eliminated *Learn Python The Hard Way* from this list; after further review, it did not provide any useful references to pythonic idioms. Similarly, we also eliminated *Practical Python Design Patterns* because it was focused on specific use cases and design patterns rather than generalized scenarios.

Additionally, we re-reviewed a selection of 2 of the books Alexandru et al. chose (*Writing Idiomatic Python* [9] and *The Little Book of Python Anti-Patterns* [18]) to make comparisons between our newly identified idioms and the results of the original paper.

We scanned each source for keywords and phrases such as: "pythonic", "clean[er]", "readable", "idiom", "style", "pattern", "easy/easier", "fast", "quick", "commonly used" and "maintainable". Topics that mentioned these terms were noted down in the form of a spreadsheet, matching the topic on one axis with the sources on the other.

3.1 Identified idioms

Having created the spreadsheet, we noticed that nearly all the new idioms we managed to identify were also present in the two older sources we chose from the original paper. Conversely, almost every one of the idioms discovered in the original paper were mentioned in the newly identified literature as well. This validates the approach of the original authors, and also shows that the sources we chose were generally reliable and accurate.

We managed to find a significant amount of new idioms (29) using this approach. 4 of these idioms were filtered out due to a lack of explanation as to the use case or usefulness, being refuted as not pythonic by another conflicting source, or not being mentioned in a significant amount of sources (for example, only 1 source).

Some of the newly identified idioms, such as the "*f-strings*" feature which was released at the end of 2016 [15], were not mentioned in the older sources due to being Python features that were not widely known or used at the time of publishing; however, they have since gained attention and received mentions in our new sources. Meanwhile, the "*walrus operator*" was released with Python 3.8 [16] at the end of 2018 [12]; however, almost all of our sources were published before 2018, except for *Effective Python's Second Edition*, the only book that mentioned it. Perhaps in the future, it will gain some popularity and be discussed in newer books, but for now, we exclude it from our list.

Conversely, the "*using else after a for-loop*" idiom was discussed in the older literature sources but not in the new ones, so we also decided to filter this out.

Having filtered out 4 idioms, we are left with 25 newly identified idioms, and together with the 21 idioms that Alexandru et al. had already covered, this comes to a total of 46 idioms covered. An overview of these numbers is given in Table 1.

3.2 Formation of the online catalog

After identifying the pythonic idioms, we compiled our results in the form of an online catalog[5].

[5]Our pythonic idiom catalog: https://bit.ly/3cBHLwQ

Original list of idioms	21
Newly identified idioms	29
Filtered from new list	4
Final number of new idioms	25
Total set of idioms	46
Detectable idioms from original list	21
Detectable idioms from new list	6
Total number of detectable idioms	27

Table 1: Overview of idiom counts.

Initially, the idioms were categorized into distinct groups so that separate pages could be made for each topic. We provided definitions and explanations for each idiom, followed by simple examples of how to incorporate them in example use cases. We also provide references to a list of resources on each idiom category: links to relevant Python documentation, books that mention the topic, and where possible, links to the relevant detection code.

All of the identified idioms were discussed either in the Python documentation[6] or as a *PEP* (Python enhancement proposal)[7]. By taking these into account, as well as definitions from our chosen literature sources, we also wrote a condensed definition and purpose for each idiom. In addition, there are examples of what the "not pythonic" implementation is, which should be avoided, and provided the converse "pythonic" implementation using the idiom, taking inspiration from the Python docs and literature sources for the examples.

4. EXTERNAL REPLICATION

As previously stated, one of the goals of this paper was to verify the idiom usage count results of Alexandru et al. by employing an external replication of their experiment.

Experiment 1 — replicating original results

Initially, we reached out to the authors and requested their idiom detection code which they used to produce their results. We studied their code to understand how it worked and observed whether there were any outdated dependencies, if the project was still able to compile, and if running the project produced any fatal run-time errors that would produce incorrect results.

Next, we replicated the experiment where Alexandru et al. ran their detector on 1,000 popular Python GitHub repositories, and observed whether or not the results were in line with what they had recorded in their paper. The replication package contained a list of the repositories that they used in the original experiment, together with the results from when the experiment was run. We re-ran the detector using the same list of repositories, with some slight differences that are discussed below.

Because the replication experiment is conducted on the latest code of each repository in the original list, some years after the original experiment, the results from this experiment will additionally help us to answer **RQ3** as we can compare the results Alexandru et al. from some time ago to new results from today.

Discussion

After analyzing their idiom detector, we conclude that the approach Alexandru et al. used was appropriate.

[6]Python docs: `https://docs.python.org/3/`
[7]List of Python PEPs: `https://www.python.org/dev/peps/`

The idiom detector, written in Scala, works by pulling a Git repository using a given link, then calling a Python script that parses every Python file in the repository, making use of the built-in AST module. This results in an *abstract syntax tree*, which the detector can then analyze to count the occurrences for each idiom we are interested in by looking for patterns such as function call identifiers, keywords, or the usage of certain Python features.

The counts are accumulated per project in the form of CSV files, and the authors also include a separate Python script that can aggregate the results across all the CSVs to produce a LaTeX table.

Included in their source code was also a set of tests with sample files, where each file contained one variation of the idiom they intended to detect. We verified that these tests were appropriate and ensured that they still passed.

A limitation we identified with this approach during Experiment 3 was that the detector can only find *instantiations* of certain data structures or classes, such as "collections.namedtuple", but not track how many times the variables are then used. This is rather difficult to detect in Python due to the lack of strong typing, and as such, there are additional uses that are not included in the results.

In the original experiment, the authors ran their detector on 997 repositories. They include the list of repositories in the form of a `.txt` file in the replication package in addition to the resulting CSV data files. However, we noticed that only 396 of the repositories in the data files overlap with the 997 sources given in the `.txt` file, which is a flaw with the replication package. We believe that sometime after the experiment, someone inadvertently reran the repository collection script, overwriting the original list. Nonetheless, we attempted to reconstruct the original list based on metadata from the CSV files but could not do so for 9 repositories due to incomplete metadata.

An additional issue was that 11 of the repositories used in the original experiment no longer exist. As a result, our re-run experiment had 977 repositories instead of the original 997. To counteract this, we excluded the data pertaining to the 20 missing projects from the "original" results so that we can make a meaningful comparison for the projects that were still available.

Results

The results of this experiment can be seen in Table 2.

When drawing conclusions based on our results, it is important to keep in mind that the use count of idioms increasing also results from the projects themselves naturally growing as their developers work on their projects. The most indicative metrics to consider are when the *number of projects* using a particular idiom strictly *increases* with a margin of error of 3% (7 idioms), which indicates adoption by more Python developers, or when the *use count* for an idiom strictly *decreases* (3 idioms), signaling that Python developers have begun to move away from them.

However, we also note that overall, the number of lines across all projects increased between the original experiment and the re-run by 5.67% which we can also consider as a reasonable margin of error; on average, differences larger than this indicate increased adoption as well (15 idioms).

From Table 2, we conclude that there were 5 idioms where the usage remained more or less constant, supporting the hypothesis we made. However, 15 idioms increased in pop-

Idioms	Original results (repaired)		2021 results (repaired)		Percentage difference		Conclusions
	Projects	Use Count	Projects	Use Count	Projects	Use Count	
List comprehension	851	74763	848	87104	0.35%	16.51%	Group 1
Dict comprehension	143	791	194	1145	35.66%	44.75%	Group 2
Generator expression	697	32867	713	41493	2.3%	26.25%	Group 1
Decorator	753	113841	765	166569	1.59%	46.32%	Group 1
Simple magic methods	748	77999	746	81870	0.27%	4.96%	Group 0
Intermediate magic methods	412	13227	417	13007	1.21%	1.66%	Group 0
Advanced magic methods	189	2612	184	2548	2.65%	2.45%	Group 0
finally	496	18881	509	18887	2.62%	0.03%	Group 0
with	835	141435	833	219089	0.24%	54.9%	Group 1
enumerate	671	19178	676	21605	0.75%	12.66%	Group 1
yield	661	56396	676	56537	2.27%	0.25%	Group 0
lambda	653	109369	667	45632	2.14%	58.28%	Group -1
collections.defaultdict	310	2908	320	3996	3.23%	37.41%	Group 2
collections.namedtuple	258	2197	275	2589	6.59%	17.84%	Group 2
collections.deque	176	1685	186	1862	5.68%	10.5%	Group 2
collections.Counter	130	1036	158	1360	21.54%	31.27%	Group 2
@classmethod	512	22129	523	29615	2.15%	33.83%	Group 1
@staticmethod	482	11486	503	15986	4.36%	39.18%	Group 2
zip	544	14812	550	17013	1.1%	14.86%	Group 1
itertools	126	835	129	918	2.38%	9.94%	Group 1
functools.total_ordering,	29	81	35	98	20.69%	20.99%	Group 2

Table 2: A comparison between the results of Alexandru et al. and the reconducted experiment results (Experiment 1)
🟥 More insight needed | 🟦 projects ± same, usage ± same | 🟨 Projects ± same, usage up | 🟩 projects, usage up

ularity by looking at the increase in use count (indicated with yellow and green), thus disproving the hypothesis. Further, 7 of these idioms saw increased adoption by developers, as they were used in a significant amount of projects they previously were not (indicated in green).

Lastly, we do not consider the "lambda" idiom in our conclusions as we discovered that an anomalous project[8] contained an exceptionally high count (74,593) in the original data but not in the re-run (4,482), thus heavily skewing the results. We investigated this further in the New empirical results section to ensure this was not due to a bug with the detector and to form a conclusion about it.

5. BEYOND REPLICATION

Previously it was discussed that one of the desired outcomes when answering **RQ2** was to provide some evidence with regards to the popularity of our newly discovered idioms, and as such, demonstrate that they are accepted as being pythonic. We can do this through Experiment 2. We also wanted to comment on how the usage of pythonic idioms had evolved to answer **RQ3**, and Experiment 3 is how we can derive these results.

Experiment 2 — detection on 1,000 new repositories

Since we had previously established in Replication that the existing detection techniques were able to accurately detect the usage of the original set of idioms, we chose 6 of the 25 new idioms we identified during the Literature review (bringing us to a total of 27 detectable idioms, as can be seen in Table 1) such that we could detect them by simply making adjustments to the existing code. Thus, we can be sure that the usage counts are accurate.

An additional reason we chose these 6 idioms is that they are built-in functions that are typically only used for one intended purpose, and insight into the context the idioms were used is not required. One example where it would be required is when detecting the use of `set()` to clear duplicate elements in a list; there are several possible reasons to call `set()` and pinpointing the developers' intent to the purpose we are interested in is not possible through automated detection.

We subclassed some of the analyses from the original project and extended them to detect the "heapq", "pprint", "@property", "__repr__ and __str__", "format" and "join" idioms. The new detectors can be found in a GitHub repository[9], and this repository is also referred to in our online catalog.

A fresh set of 1,000 popular Python repositories was collected using the BASH script the original authors used, and the detector was run on this new set of repositories, using the enhanced analyses we wrote. If the usage counts of the new idioms align with those we were already able to discover, we can conclude that the newly identified idioms are demonstrably in wide use, and as such embraced as pythonic.

Results

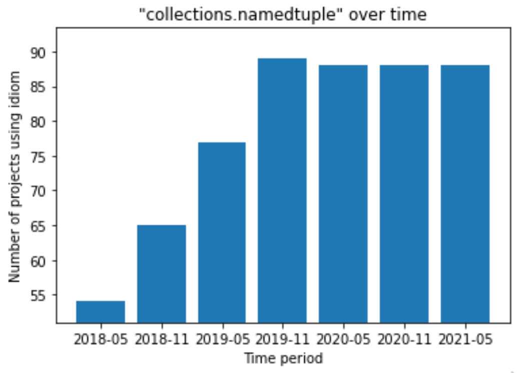
The results from running the detector using the enhanced analyses can be found in Table 3, and the scripts, list of repositories, and result data files can be found in our GitHub repository[10].

As can be seen from the table, the inter-idiom differences in usage for Experiment 2 are aligned with the results from Experiment 1. We also observe that "__repr__ and __str__" and "@property" are two idioms that have particularly high usage counts, being more widely used than 17 and 16 other idioms respectively. Under the assumption that the idioms Alexandru et al. discovered were pythonic, and as the usage for the new idioms are demonstrated to have similar usage statistics, we can therefore conclude that our newly identified idioms are also pythonic.

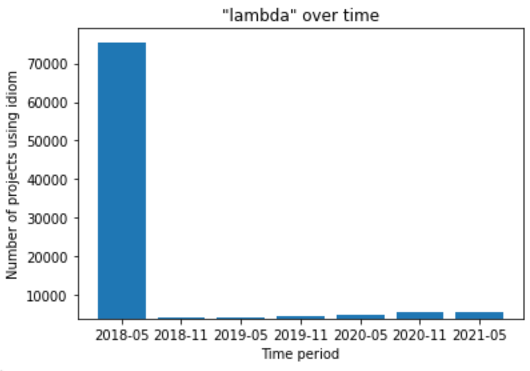
[8]Link to anomalous project: https://github.com/sympy/sympy

[9]Idiom detection code: https://bit.ly/3qnSL70

[10]See footnote 9.



(a) collections.namedtuple



(b) lambda

Figure 2: Number of uses of two idioms over time, snapshots taken 6 months apart

Experiment 3 — repository snapshots over time

We selected 10 repositories that were included both in the original list of repositories and our newly collected ones. The repositories we selected have been and are still under active development, which we verified by checking if there was at least one commit 6 months apart for the past 3 years, from May of 2021 going back to May of 2018. This was verified automatically using a Python script which queried the GitHub API for this information[11]. The repositories were chosen using these constraints because they are demonstrably well-maintained and receive regular updates and feature improvements, consistent with developments in Python itself. As such, they are best suited for a study on how pythonic idiom usage evolves over time.

In the same script, we collected the hashes for the first commit that landed in the chosen time periods (one commit from May and November of each year up until May 2021) for each of the 10 repositories and stored these in text files. The detector from Alexandru et al. contains an agent which takes a file containing a list of Git repository links and automatically clones them, runs the detection, and then deletes the files afterward. We extended this agent to additionally checkout a given commit after cloning, and then run the detector. By doing this, we can essentially use snapshots of projects to detect and observe how the usage of idioms evolved every 6 months from May 2018.

Results

Two of the resulting graphs are seen in Figure 2, and the rest can be found in a separate report [8]. We categorized the results into distinct groups in Table 4.

From this table, it is seen that the usage count against time grew over time for 21 out of the 27 detectable idioms, including 12 of the 15 idioms we previously said to have gained popularity in Replication. For the remaining idioms out of the 27, 3 idioms had usage that remained constant, and 3 had results where additional insight was needed.

With regards to the idioms from Replication, 7 out of 15 the idioms we said to have increased in popularity saw some differences in results that are worth pointing out. In Figure 2(a), the usage of "collections.namedtuple" nearly doubled, then remained constant for some time. This is still an increase from where it started, however, it is worth pointing out that the usage stagnated. The same pattern is seen with "collections.Counter". The reason behind this is most likely the limitation we identified with the detector in Replication.

The same pattern was also seen with "yield", "pprint" and "join" and we believe this is due to a "saturation point" that has been reached in the projects where it was simply not necessary to use the idioms more than the already had been, or because these idioms are not as widely applicable as other idioms.

Conversely, we observed that for each of the idioms we previously claimed to have remained constant of usage, the usage actually increased for this selection of projects. Meanwhile, the usage of "heapq", "functools.total_ordering", and "itertools" remained more or less constant in this selection of projects. While these results are noteworthy, they only pertain to a small sample of projects, and the larger sample of 977 projects shows more generalized results across the Python community.

We also observed once again that in Figure 2(b), as with the results of Experiment 1, the usage of the anomalous "lambda" idiom decreased from roughly 75,347 uses to 3,940. This is due to the same anomalous project from the Replication being present in our selection of repositories here. To investigate this further, we cloned the repository using the commit hash from the original experiment's metadata and ran a Ctrl+F search in the repository. By doing so, we verified that the exceptionally high use count of "lambda" was legitimate and not due to a bug with the detector. We then cloned the repository from the next timeframe (November 2018) and ran the search again, and indeed the usage of "lambda" was significantly lower, caused by a major refactor.

The conclusion here is that while the numbers in Table 2 were correct, the choice of including this repo in Alexandru et al.'s experiment heavily skewed the popularity of "lambda" in the original results. However, after the anomalous time period, the usage then rose again steadily to 5,498 as seen in Figure 2(b), we can conclude here that "lambda" grew in popularity for this selection of repositories.

A similar result was observed with "@staticmethod", which increased in usage since May of 2018, but curiously, the usage decreased by nearly a quarter between November 2020 and May 2021. After investigating these two snapshots, we observed a similar result as with "lambda" — once again, the same anomalous project underwent another refactor which caused the usage of "@staticmethod" to decrease

[11]Python commit fetching script: `https://bit.ly/35dYxhC`

Idioms	Re-run original experiment		Experiment with new list	
	Projects	Use Count	Projects	Use Count
List comprehension	848	87104	829	56115
Dict comprehension	194	1145	144	699
Generator expression	713	41493	592	22719
Decorator	765	166569	644	101208
Simple magic methods	746	81870	650	44586
Intermediate magic methods	417	13007	263	7173
Advanced magic methods	184	2548	102	1196
finally	509	18887	325	8282
with	833	219089	872	109501
enumerate	676	21605	705	18071
yield	676	56537	518	35768
lambda	667	45632	557	23498
collections.defaultdict	320	3996	258	2589
collections.namedtuple	275	2589	200	1472
collections.deque	186	1862	126	697
heapq	—	—	43	193
collections.Counter	158	1360	129	840
@classmethod	523	29615	380	16711
@staticmethod	503	15986	435	11841
@property	—	—	464	37795
zip	550	17013	552	12553
itertools	129	918	74	445
functools.total_ordering,	35	98	21	58
__repr__ and __str__	—	—	470	15031
pprint	—	—	131	1076
format	—	—	165	2720
join	—	—	143	4617

Table 3: Results of Experiment 1 next to Experiment 2.

from 515 to 314, skewing the results again. Nonetheless, we again conclude that "@staticmethod" grew in popularity by referring to our generalized results from Table 2.

As for the 6 newly identified idioms we were able to write detectors for, "heapq" and "pprint" stay constant in terms of usage, while "@property" went up. The usage of "format" decreased, and we hypothesize that this is due to the introduction of "f-strings", one of the idioms we covered that was introduced in Python 3.6 [26], and as we previously illustrated, a cleaner way to approach string interpolation. Additionally, the usage of "join" increased then stayed constant, and "__repr__ and __str__" increased steadily.

6. CONCLUSIONS

Through the course of this paper, using the great efforts of Alexandru et al. as a foundation, we dove deeper into the ecosystem of Python and its pythonic values. In this final section of the paper, we provide definitive answers for our research questions through the results we gathered across each of our experiments and the literature review.

Research Question 1

Through a literature review rooted in grounded theory, we were able to uncover 25 new pythonic idioms, increasing the total number of pythonic idioms discovered to 46. An overview of the idiom counts is seen in Table 1. Every idiom is given with detailed definitions, use cases, links to detectors, and examples inspired by the literature sources, and these can be found on our online catalog[12]. Further, the data files, list of repositories, aggregation scripts, and detectors can be found in a GitHub repository[13].

Research Question 2

We extended Alexandru et al.'s detectors to include 6 out of the 25 newly identified idioms. From the results of our experimentation (Experiments 2 and 3), we have established that each of the 6 idioms are under wide use in the most popular Python projects, and as such, are pythonic. The usage statistics are comparable to the idioms previously identified by Alexandru et al. as pythonic; in some cases, even more so, by observing the number of projects using the "__repr__ and __str__" and "@property" idioms in Table 3.

Research Question 3

As can be seen in Table 1, we were able to experiment on a set of 27 idioms, of which 21 were from the original set of idioms, and 6 were part of our newly identified idioms. We uncovered from the results in Experiment 2 and Experiment 3 that from the original list of pythonic idioms provided by Alexandru et al., 16 out of 21 idioms saw an increase in popularity, including the "lambda" idiom which required additional research through Experiment 3. 5 idioms' usage remained constant (all 3 of the "magic method" categories, as well as "finally" and "yield").

In addition to the original set of idioms, 2 of the 6 new idioms we wrote detectors for had constant usage ("heapq" and "pprint"), 3 saw increased usage ("@property", "join", "__repr__ and __str__") and the usage of "format" decreased (for which we conjectured potential reasons, but this requires additional research).

[12]Our pythonic idiom catalog: `https://slimshadyiam.github.io/ZenYourPython`

[13]Online: `https://bit.ly/3qnSL70`

Idioms	Conclusions
List comprehension	🟩 Group 2
Dict comprehension	🟩 Group 2
Generator expression	🟩 Group 2
Decorator	🟩 Group 2
Simple magic methods	🟩 Group 2
Intermediate magic methods	🟩 Group 2
Advanced magic methods	🟩 Group 2
finally	🟩 Group 2
with	🟩 Group 2
enumerate	🟩 Group 2
yield	🟨 Group 1
lambda	🟥 Group -1
collections.defaultdict	🟩 Group 2
collections.namedtuple	🟨 Group 1
collections.deque	🟩 Group 2
heapq	🟦 Group 0
collections.Counter	🟨 Group 1
@classmethod	🟩 Group 2
@staticmethod	🟥 Group -1
@property	🟩 Group 2
zip	🟩 Group 2
itertools	🟦 Group 0
functools.total_ordering,	🟦 Group 0
__repr__ and __str__	🟩 Group 2
pprint	🟨 Group 1
format	🟥 Group -1
join	🟨 Group 1

Table 4: Results of Experiment 3, categorized.
🟥 More insight needed | 🟦 usage ± constant | 🟨 usage up, then stagnated | 🟩 usage up

Threats to validity

With regards to the internal validity of our literature review to discover idioms, as our literature sources were chosen based on sentiment from posters on online media such as forums and blogs, it is possible that the most prominent results are purported by highly opinionated individuals, not representative of the majority. We also assume that the idioms that are discussed in literature sources and idioms that are used in actual code overlap heavily. Further, it is probable that the idioms authors choose to write about are biased to their own personal preferences, which means that certain idioms are given more attention than others, or that opinions may conflict across sources.

It is for this reason that we cross-reference our results across a large variety of different sources, including some from Alexandru et al.'s original study, and through our filtering process, we hope to rule out any such instances of bias.

A threat to the external validity of this research could arise from the choice of only using prominent open source projects during our idiom detection phase, which threatens the generalizability of our results to the pythonicity of closed source projects. There is some debate about the code quality of open source projects as opposed to proprietary or closed source projects [19].

Some may contend that since open source projects are in the public eye, those who do contribute would want to contribute high quality code as well, because as Raghunathan et al. agree, this signals talent to the open source community and potential employers [19]. Conversely, it has been experimentally demonstrated that providing extrinsic motivation to workers through rewards (for example, paying them a salary) conflicts with intrinsic motivation [5]. This means that monetary incentives harm the quality of code in the long term, because removing the extrinsic motivation causes workers to be less interested, and therefore produce lesser quality code. Meanwhile, those who were merely working out of intrinsic interest will continue to show the same interest [5].

In the end, Raghunathan et al. conclude that the quality of code in closed source projects is not necessarily higher than that of open source projects [19]; as such, we believe that our results are generalizable to closed source projects as well.

Future work

A wide array of research based on this paper can be conducted, such as writing new detectors for the remaining new idioms we did not write detectors for, as well as discovering their usage in projects. An interesting application, as we previously described, would also be to determine and detect all of the "anti-patterns" in Python, for which we illustrated potential uses in the Introduction section.

References

[1] C. V. Alexandru, J. J. Merchante, S. Panichella, S. Proksch, H. C. Gall, and G. Robles. 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*, Onward!, page 1–11. Association for Computing Machinery, 2018.

[2] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of*

Software Engineering, volume 16-21-November-2014, pages 281–293, 2014. Cited By :161.

[3] W. Badenhorst. *Practical Python Design Patterns: Pythonic Solutions to Common Problems*. Apress, 2017.

[4] D. Bader. *Python Tricks: A Buffet of Awesome Python Features*. Dan Bader, 2017.

[5] R. Bénabou and J. Tirole. Intrinsic and Extrinsic Motivation. *The Review of Economic Studies*, 70(3):489–520, 07 2003.

[6] P. Carbonnelle. Popularity of programming language, 2021. Online: https://pypl.github.io/PYPL.html.

[7] B. K. J. David Beazley. *Python Cookbook, 3rd Edition*. O'Reilly Media, Inc., 2013.

[8] A. Farooq. How to zen your python – graphs. Technical report, University of Twente, June 2021. https://doi.org/10.6084/m9.figshare.14782170.v1.

[9] J. Knupp. *Writing Idiomatic Python 3.3*. Createspace Independent Pub, 2013.

[10] A. Langer. Java programming idioms. In *"Technology of Object-Oriented Languages and Systems*, pages 197–198, 2001. Cited By :2.

[11] M. J. Munro. Product metrics for automatic identification of "bad smell" design problems in java source-code. In *Proceedings - International Software Metrics Symposium*, volume 2005, pages 15–24, 2005.

[12] Python Software Foundation. Pep 569 – python 3.8 release schedule. Online: https://www.python.org/dev/peps/pep-0569/.

[13] Python Software Foundation. Sunsetting python 2. Online: https://www.python.org/doc/sunset-python-2/.

[14] Python Software Foundation. What's new in python. Online: https://docs.python.org/3/whatsnew/index.htmll.

[15] Python Software Foundation. What's new in python 3.6. Online: https://docs.python.org/3/whatsnew/3.6.html.

[16] Python Software Foundation. What's new in python 3.8. Online: https://docs.python.org/3/whatsnew/3.8.html.

[17] Python Software Foundation. The zen of python, 2004. Online: https://github.com/python/peps/blob/master/pep-0020.txt.

[18] Quantified Code. The little book of python anti-patterns, 2014. Online: https://github.com/quantifiedcode/python-anti-patterns.

[19] S. Raghunathan, A. Prasad, B. Mishra, and H. Chang. Open source versus closed source: Software quality in monopoly and competitive markets. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35:903 – 918, 12 2005.

[20] K. Reitz and T. Schlusser. *The Hitchhiker's Guide to Python: Best Practices for Development*. O'Reilly Media, 2016.

[21] T. Sakulniwat, R. G. Kula, C. Ragkhitwetsagul, M. Choetkiertikul, T. Sunetnanta, D. Wang, T. Ishio, and K. Matsumoto. Visualizing the usage of pythonic idioms over time: A case study of the with open idiom. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 43–435, 2019.

[22] Z. A. Shaw. *Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code*. Addison-Wesley Professional, 1st edition, 2017.

[23] F. Shull, V. Basili, J. Carver, J. Maldonado, G. Travassos, M. Mendonca, and S. Fabbri. Replicating software engineering experiments: addressing the tacit knowledge problem. In *Proceedings International Symposium on Empirical Software Engineering*, pages 7–16, 2002.

[24] B. Slatkin. *Effective Python: 59 Specific Ways to Write Better Python*. Addison-Wesley Professional, 1st edition, 2015.

[25] B. Slatkin. *Effective Python: 90 Specific Ways to Write Better Python, 2nd Edition*. Addison-Wesley Professional, 2019.

[26] E. V. Smith. Pep 498 – literal string interpolation, 2015. Online: https://www.python.org/dev/peps/pep-0498/.

[27] J. Tan, D. Feitosa, P. Avgeriou, and M. Lungu. Evolution of technical debt remediation in python: A case study on the apache software ecosystem. *Journal of Software: Evolution and Process*, 33(4):e2319, 2021. e2319 smr.2319.

[28] E. Van Emden and L. Moonen. Java quality assurance by detecting code smells. In *Proceedings - Working Conference on Reverse Engineering, WCRE*, volume 2002-January, pages 97–106, 2002. Cited By :225.

[29] E. S. Wiese, M. Yen, A. Chen, L. A. Santos, and A. Fox. Teaching students to recognize and implement good coding style. In *L@S 2017 - Proceedings of the 4th (2017) ACM Conference on Learning at Scale*, pages 41–50, 2017. Cited By :11.