# Discursive Strategies in Style Guides Negotiation on GitHub

Pierre Depaz - pierre.depaz@nyu.edu

## Introduction

Written and published in 1983 on a Usenet board, The story of Mel, the Real Programmer[[1]](#footnote-2) recounts[[2]](#footnote-3) the tale of Mel Kaye, an individual who wrote software on the 1959 ACT-1 compiler and has become a recurring reference in programmer’s lore[[3]](#footnote-4). The moral of the story focuses on Kaye’s ability to write both excellently efficient and completely inscrutable code. This obfuscating aspect of a code that only its writer can read, while being considered in this context a laudable feature of model programming work and informing ideals of programmers, slowly began to phase out in the 1970s. With the growth of the commercial software market[[4]](#footnote-5), more and more private companies started to constitute what is today a large part of the technology industry, composed of teams of multiple programmers working on the same codebase on the long-term. This evolution, from the individual programmer implementing ad hoc and personal solutions to a group of programmers coordinating across time to build and maintain large, distributed pieces of software, brought the necessity to harmonize and standardize how code is written. In response, style guides started to be published to normalize the visual aspect of source code, and became a recurring topic in both the software development and computer science research[[5]](#footnote-6); particularly, debates about programming style implementations and best practices have remained significant, if under-examined, parts of a given programmer’s practice in a commercial context.

In light of this tension between individual technical prowess and the social existence of source code, this article examines the production and communication processes involved in the construction of styleguides for the JavaScript programming language within contemporary software development environments. Specifically, it looks at the discursive techniques deployed to negotiate the adoption of styleguides within a particular socio-technical environment—the collaborative development platform GitHub. While style guides and written documents have been enforced in formal, more traditional professional institutions[[6]](#footnote-7), the GitHub development platform presents a couple of specific aspects. As the most popular repository of open-source software, it is the locus of semi-formal participation and enables by default the copying and modifying any document (a process known as forking). Large-scale private companies interact with distributed non-profit organizations and individual contributors in order to collaborate on software products, both free of charge and for-profit, which can then be modified seamlessly by other users of the platform. While GitHub is organized around self-contained software components, style guides are concerned with programming languages and aim at enforcing behaviour across projects. These discussions therefore cut across projects and applications to involve the aforementionned multiplicity of actors.

JavaScript is today both one of the most popular programming languages on GitHub, used by both amateurs and professionals alike. However, the language itself lacks a clear, original style standard, and therefore has been the subject of various discussions on what a style guide should enforce and how it should enforce it, relative to other languages. The discussions taking place on the GitHub platform therefore represent a wide variety of opinions, skill levels, and institutional belongings. In this context, this article aims at investigating *which discursive strategies are used by contributors around the formation of style guides? Furthermore, how are those strategies affected by the specificity of a socio-technical environment such as GitHub? How are users of these style guides defined and taken into account in these negotiations?* The answers to these questions will help further qualify the nature of linguistic exchanges in a complex, digital-first, working environment and uncover the arguments and attitudes towards practices in software development.

To do so, I propose to look at the discussions happening on three GitHub repositories, each representing different approaches to style guides. From least-open to most-open, they include the JavaScript style guide as published by the Airbnb company[[7]](#footnote-8), the popular, independent StandardJS[[8]](#footnote-9) and the exclusively format-oriented Prettier[[9]](#footnote-10). Building on previous critical discourse theory as applied to open-source discussions (Berry, 2006), these three repositories, their issues, pull requests and forks, will highlight the language deployed both from the maintainers, the contributors and the users and how different discursive approaches are used to justify arbitrary formal choices (e.g. single quotes vs. double quotes) through didactic approaches to non-arbitrary ends (code interoperability between teams) to improve productivity.

In the first part, I will address the origin of style in programming and, consequently, the need for style guides in contemporary, commercial programming practices, inscribing it further into both the sociology of style and the sociology of organizations. In particular, I will highlight continuities and breaks between style guides for human languages and style guides for programming languages, specificaly in terms of objective efficiency and subjective perception. Building on this dichotomy, the second part of this article highlights the specific technical intermediary objects[[10]](#footnote-11) involved in those discussions and implementations, which exist in this field at both a macro- and micro-level. Both GitHub and linters are significant components of the creation and implementation of style guides and will thus occupy a central role in this analysis. After developing on the research methods used in this study, which are centered qualitative discourse analysis of issues and pull requests, I will develop on the findings of this research, summing up discursive and structural differences between the three targeted repositories. I will conclude by focusing on the place of tools, read-only documents and read-write documents, as well as invisible communities, in the development of style guides and how these can inform our understanding of daily programming practices.

## The need for style in programming

The problem of style is a problem which echoes Simmel’s statement regarding the observation that *“the practical existence of humanity is absorbed in the struggle between individuality and generality”*[[11]](#footnote-12). Simmel’s investigation of the topic, which will serve as the foundation for our definition of style, focuses on the dichotomy between works of fine art and mass-produced works of applied arts. Indeed, Simmel draws a distinction between the former, as indiosyncratic objects displaying the subjectivity of its maker, and the latter, as industrially produced and replicated, in which the copy cannot be told apart from the original. The work of fine art, according to him, is *“a world unto itself, is its own end, symbolizing by its very frame that it refuses any participation in the movements of a practical life beyond itself”*, while the work of applied arts only exists beyond this individuality, first and foremost as a practical object. As these two kinds of work exist at the opposite extremes of a single continuum, we can insert a third approach: that of the crafted object. The work of craftsmanship exists in-between, as a repeated display of its maker’s subjectivity, destined for active use rather than passive contemplation[[12]](#footnote-13). So while style can be seen as a general principle which either mixes with, replaces or displaces individuality, style in programming doesn’t stand neatly at either extreme, and needs to be problematized further.

Source code, understod as the product resulting from programming work, finds some of its roots in craftmanship, as a practice both analyzed by scholars[[13]](#footnote-14) and claimed by practicioners[[14]](#footnote-15). Programming work has developed since the 1950s from an invisible labour to a fully organized and codified practice, through individual, glorified work. While computers were originally referencing female clerks implementing male mathematicians’ algorithms on mainframe computers[[15]](#footnote-16), the development of programming as a recognized practice brought about with it the figure of the individual (male) programmer, equally obsessed and talented[[16]](#footnote-17), echoing in some ways the portraits of the genius artist[[17]](#footnote-18)[[18]](#footnote-19). This individual, free-form implementation of specific computational problems came to an end with the rise of commercial software, and led to a re-examination of the practices and processes of software developers. This was led in part by the work of Edsger Dijkstra whom, in his 1972 publication of *Notes On Structured Programming*[[19]](#footnote-20), set forward clear practices for writing source code which would be meant to achieve two aims: be readable by someone else than the original author, as well as being more reliable and error-free[[20]](#footnote-21).

The emergence of style in computer programming is therefore concomitant of the development of the software industry, starting in the 1970s. A self-proclaimed highly complex undertaking[[21]](#footnote-22), the understanding of source code by someone who hasn’t written it (or, being the writer, hasn’t read it in a while) quickly proved itself to be particularly difficult; as a response, corporate literature arose in order to serve as a reference for best practices[[22]](#footnote-23). Style guides, in the context of programming, are textual documents whose rationale involve both social and technical components. On the social side, they are only truly useful if they are inconditionally adopted by all members working on a particular code-base, since *“all code in any code-base should look like a single person typed it, no matter how many people contributed.”*[[23]](#footnote-24); in the strict sense, guidelines are therefore reference documents which should provide an answer to the question of what is the preferred way of writing a particular programming statement (e.g. var vs. let, or camelCase vs. snake\_case). Beyond aesthetic preferences aimed at optimizing the clarity of a given source code[[24]](#footnote-25), style guides also include a technical component which aims at reducing programming errors by catching erroneous patterns in a given codebase (e.g. variable declaration before intialization, loose reference to the function-calling context). This technical component, because it can be judged by an objective standard (i.e. the amount of breaking errors in a program), is seldom the reason for internal disagreements within teams. In contrast, the aesthetic component, due to this lack of subjective criteria, opens itself up to discussion. It is both of these components which style guides aim at resolving.

This phenomenon of explicitly written rules, dependent no longer on their writer, but rather on the organization to which the writer belongs, presents similar patterns as those highlighted in the formalization of knowledge as it happened during the transition of societies from oral communication (and their constituting organizations) to written communication[[25]](#footnote-26). For instance, the written codification of hitherto implicit, idiosyncratic rulesets, has had the result of further preventing modification of said rules, and elevating them from a “personal” reach to a “universal” one; and these are patterns that re-emerge in the use of styleguides. However, the technical context for those studies of the impact of literacy is one in which the written word, once put down, is significantly harder to modify than both its spoken counterpart and its digital counterpart.

The digital word, the word stored on and communicated via computers, presents two important differences. First, it can be as easily modified as it can be retrieved. Since one of the basic operations of the computing machine is the act of copy and the act of retrieval, it is important here to remember that anything that can be said on a digital medium always holds within itself the possibility of duplication, modification and, therefore, variation[[26]](#footnote-27), illustrated on the most obvious-level by the ubiquity of “copy and paste” actions offered as a default by any word processor application on a modern computer. Second, source code written in programming languages is *executable*. This means that, beyond the linguistic act of writing a statement, the result of this writing act can be automatically executed and enforced, without stating the need for personal or social performance as a requirement. So while written code still exists within “chains of acts of writing”[[27]](#footnote-28), once its validity has been confirmed and merged into active code bases, its enforcement is significantly easier than that of guides and protocols written in human languages[[28]](#footnote-29).

## The GitHub development platform

Along with the low barrier to reproducibility and enforceability of code explicited above, another difference between organizations centered primarily on code and software and organizations in which code only constitutes a technical background is the porosity of the distinction between public and private. Particularly, when organizations maintain, or depend on, open-source software, the delimitation of which individual contributes to the organization’s product becomes more fluid and temporary than in more traditional organizations[[29]](#footnote-30). At its most radical form, it is entirely non-hierarchical and horizontal[[30]](#footnote-31), a structure in which anyone can comment on the product and suggest modifications, even though actual contribution remains subject to additional social and economic constraints[[31]](#footnote-32). This configuration directly affects the scope of a style guide. If anyone can potentially contribute to any code base within a given language, then the potential scope for any style guide is that of universal adoption. This relatively loose set of mutually-beneficial work relations (between private companies, open-source organizations and individual contributors) constitutes the background of our research field and is accentuated by the specificity of the GitHub platform.

As all platforms, GitHub conects multiple actors and provides the backdrop for economic, social and cultural practices[[32]](#footnote-33). Economically, GitHub provides a way to store, retrieve and modify text files (usually source code) for distributed teams of contributors, grouped into projects (called repositories), themselves administrated by either personal or organizational accounts. Any public repository is accessible to anyone, and these projects can then be built to add to the value of a given commercial product (e.g. Apache’s nginx, Facebook’s create-react-app). Socially, GitHub requires user registration to contribute to any of those repositories, and maintains a transparency policy which makes available all of a given users contributions on any given public repository. These contributions overwhelmingly take the form of commits (direct modification of text files), pull requests (requests to a given organization to integrate suggested changes to a text file), and issue creation and comment (asking or answering a question on a given project repository). Culturally, user interactions on GitHub depend on agreed-upon practices and discourses, specifically when a user raises an issue, responds to it, or concludes (closes) it[[33]](#footnote-34).

## Three repositories

Within this context, this article looks at the discussions taking place in three particular open-source repositories on the GitHub platform. Since our aim is to examine how discursive strategies are successful in the enforcement and popularization of style guides, a guiding criterion for narrowing our research field was first to use the common indicators of stars (number of users interested in the development of the repository, the GitHub equivalent of a bookmark)[[34]](#footnote-35) and in forks (number of users having copied the repository to their own account, possibly to modify it further, and to submit these changes back into the original repository). For the JavaScript language, the top three most popular repositories are airbnb/javascript (100k stars, 19.5k forks), prettier/prettier (37.4k stars, 2.5k forks) and standard/standard (24k stars, 1.9k forks). Additional JavaScript style guides include the IdiomaticJS style guide[[35]](#footnote-36), the Google JavaScript Style Guide[[36]](#footnote-37) and the style presented by Douglas Crockford[[37]](#footnote-38), but do not meet either the popularity criteria, or do not exist primarily on GitHub.

These three repositories, while sharing the same effective outcome of providing a style reference for other JavaScript programmers, do present differences in their approaches to achieving that goal, and in their organizational practices. The airbnb/javascript repository is the public-facing result of the guide used internally at the Airbnb company, a startup company privately valued at USD31 billion[[38]](#footnote-39). It consists of a main document, README.md, the first page displayed on any GitHub repository, as well as configuration files for linters. The standard/standard repository is a non-profit open-source organization aiming at providing a non-modifiable, non-extensible way to programmatically enforce both stylistic choices and technical error-checking through the use of the standard software, a command-line utility which aims at automatically enforcing the style described on an additional RULES.md document[[39]](#footnote-40). Finally the prettier/prettier repository limits itself to the strict formatting of source code, following the rules laid out on an external website[[40]](#footnote-41) and providing, similarly to standard/standard, a command-line utility to automatically enforce those changes[[41]](#footnote-42). GitHub is therefore a combination of read-write texts and read-only texts. The read-write texts consisting of the main text files of the repository, while the read-write texts are composed of all the discussions and suggestions taking place in the issues and pull requests sections. Drawing from literary theory[[42]](#footnote-43), this distinction allows us to identify the more authoritative text and the more negotiable ones.

This study focuses on the issues sections of each of these repositories. Traditionally used to keep track of bugs and technical enhancements for project maintainers[[43]](#footnote-44) and have a dedicated section on each repository. An issue can be created by anyone with a GitHub user account, can be commented on by anyone with a user account, but can only be closed (or “resolved”, in GitHub’s terminology) by the original creator, or by project maintainers. Additional features is that they can reference other issues across repositories, as well as internal and external code requests, essentially creating a web of inter-textuality, and displaying the complexity and inter-dependency of software packages. While originally designed as a bug-tracker, research has shown that they are now the locus of more complext discussion, involving affective rhetoric and over-arching design inquiries, well beyond specific technical fixes[[44]](#footnote-45)[[45]](#footnote-46).

## Automatic writing with code linters

A final relevant element constituting our research field is the code linter. Present in every examined repositories, and beyond the JavaScript ecosystem, a code linter is a piece of software which, given a set of syntactical rules, modifies one or multiple source code files to match said rules. This programmatic element, which can be seamlessly integrated into more complex workflows, represents both a transfer of agency from the human to the non-human, as well as an improvement in the systematic enforcing of style rules.

Due to the overwhelming presence of automatic tooling in modern development[[46]](#footnote-47), programming style guides make configuration files for linters one of the most essential part of their project, effectively connecting intimately human-readable texts with their machine-executable counterpart. The most popular of those linters in the JavaScript ecosystem is eslint, with over 13 million weekly downloads[[47]](#footnote-48). While not proposing entirely new linters, our examined repositories focus on maintaining configuration files for eslint which implement their own human-readable styleguides. As a tool in a programmer’s workflow, these linters with their associated configuration files are what Latour and Woolgar have called *intermediary objects*[[48]](#footnote-49), objects which contribute essentially to labour and knowledge-creation. Previous research in the sociology of work focused on how these intermediary objects affect the work processes (both conception, discussion and realization)[[49]](#footnote-50), particularly in establishing a framework within which work can take place[[50]](#footnote-51); coupled with the automated normativity of code[[51]](#footnote-52)[[52]](#footnote-53), these linters represent a significant part of the life and impact of a style guide.

## Methodology

After having provided the context in which this study takes place, we turn to our methodological approach. The methodology employed here to analyze (1) how different actors negotiate the adoption and negotiation of both subjective and both objective stylistic norms, and (2) the influence of the technical environment in which those discussions take place, is based on the discourse analysis of individual issues and pull requests in each examined repository. This approach will focus mainly on the most debated (i.e. those with the most comments) issues, highlighting the discursive strategies and patterns of the participants; since most of the least-commented issues are overwhelmingly related to technical issues and bugs which, by implying a single fix, stand outside of the field of argumentation. Specifically, this methodology aims at linking the meanings of the corpus not just to existing social realities[[53]](#footnote-54), and to different kinds of communicative competences but to technical realities as well. Habermas provides a useful typology of communicative competence, composed of “a statement of fact, an expression of feeling, or a command”[[54]](#footnote-55); and this typology will serve not as a normative framework, but rather as a starting point for approaching the positions, assumptions and aims of contributors in the discussion. Regarding the restriction of the analysis to the exclusive locus of issues and pull requests, as it has been mentionned above, GitHub is composed of an intricate web of texts which cross-reference each other, and therefore renders any clear distinction blurry. As such, we will consider that a read-only document (such as README.md) will be relevant to our analysis insofar as it *addresses* a specific user, with a specific voice: if this linguistic interaction doesn’t imply discussion, it certainly implies discourse.

Several studies have previously looked into the discourses on open-source communities[[55]](#footnote-56), mostly through the use of large-scale computational techniques[[56]](#footnote-57)[[57]](#footnote-58). The proposed approach intends to examine more specifically the nuances and implicit assumptions within the arguments deployed by the user, working under the hypothesis that, while broad strategies have been identified for issue resolution[[58]](#footnote-59), a more detailed approach in the specific field of styleguide adoption might provide insights in the daily practices of programmers, and to what extent they implement, re-appropriate, or hijack (*détournent*) the broad strategies of technical resolution assumed by the GitHub platform[[59]](#footnote-60).

## Analysis of corpus

### Airbnb

As pointed out by *mylestan*, the airbnb/javascript issues presents itself as much as a question-and-answer platform rather than as a traditional bug-reporting and fixing platform[[60]](#footnote-61).

*Hi! First off, thanks for this package. Also, I’ll preface this with: I’m not 100% sure I’m using the react/whitespace option correctly, so this is more of a question than a bug report.*

With a low ratio of open-to-closed issues, the airbnb/javascript style guide consists of bug reports on eslint[[61]](#footnote-62), individual projects[[62]](#footnote-63) or the guide itself[[63]](#footnote-64), even though all issues that are labeled as “bugs” represent less than 1% of the total issues opened. In this case, bugs are understood as inconsistencies between the *eslint* output and airbnb/javascript’s guidelines[^54]. What would be deemed inconsistencies internal to the guide itself are labeled as *editorial*, reflecting a broader concern with communication and understanding of concepts and paradigms over strict technical implementation issues.

Indeed, the unique aspect of airbnb/javascript is its dependency on the internal, well-organized structure of a private company. The constitution of the rules are, therefore, not up to debate. Such an implicit, private, out-of-reach existence of the origin of the guide is made explicit both by the creator of the publicly available guide, for whom the pronouns *we* and *our* do not refer to the open-source community of commenters and contributors, but rather to the internal team at Airbnb[[64]](#footnote-65)[[65]](#footnote-66), as well as references by both parties in discussions to the practices of the company (*“airbnb pushes for […]”*[[66]](#footnote-67), *“I’d like to see this covered by Airbnb’s standards”*[[67]](#footnote-68)) or to the codebase from which the styleguide emerged[[68]](#footnote-69). On a chronological level, it is then interesting to see that the actual negotiations which happened on the repository’s issues (mostly issues #1-#40) are those in which most of the interactions happen between Airbnb employees and take place as multi-sided discussions on how to improve the guide[[69]](#footnote-70)[[70]](#footnote-71)[[71]](#footnote-72) or on bug- and inconsistency-fixing[[72]](#footnote-73). What we see here is an instance of a private company using the tools and infrastructure of open-source processes embedded in the GitHub platform in order to improve their productivity by writing internally consistent code[[73]](#footnote-74), while, in a second moment, asking community members for their inputs without promising to implement them[[74]](#footnote-75).

From this early period of a public interaction between fellow members of the same private organization, to the current situation of a large-scale interaction of distributed individuals and organizations over one of the most popular JavaScript styleguides, the discursive strategies of both the maintainers and of the commenters and contributors have shifted. This is due in part to the fact that the creation of the styleguide is not collaborative, in the open-source sense of the term. Since rational discussion leading to a possible change of mind of the maintainer isn’t possible, the alternative chosen by the Airbnb team is the didactic explanation of non-negotiable rules. One example is featured in the explanation of an early issue opened on the styleguide about the broader need for styleguides; the creator responds with a pedagogical *métaphore filée* of painters and fine artists rather than logically, rationally approaching the need for consistent codebases[[75]](#footnote-76). Along with the fact that this interaction reveals the desire for the maintainers of the project (i.e. Airbnb employees) to explain, rather than discuss, the conclusion of this particular discussion also sheds light on the forking mechanism as used in a discursive situation. Along with this particular issue, others also confirm the fact that forking acts as an end-all conclusion to the debate, in which another discussion is created, taking as its axiom the contentious proposals evoked in the base branch[[76]](#footnote-77)[[77]](#footnote-78)[[78]](#footnote-79).

Additionally, a particular aspect of airbnb/javascript is the progressive inclusion of *why* such a rule has been decided in the read-only README.md[[79]](#footnote-80). By coupling exhaustivity with justification, the main task remaining for the maintainers of the project is to continue the explanation of why things are the way they are[[80]](#footnote-81). The case of prefer-default-exports highlights the pattern of a maintainer repeating the same justification for recurring questions[[81]](#footnote-82)[[82]](#footnote-83)[[83]](#footnote-84). What happens on the repository is the development of a pedagogical discussion in which the “bugs” opened are presented as misunderstandings from the commenter’s point of view, asking the maintainers, not to justify their style choices, but rather to *explain* how one can write code that would better match the guidelines explained[[84]](#footnote-85). If that explanation fails, as in the prefer-default-export examples above, another strand of conversation is engaged in by one of the core maintainers, user ljharb, former Airbnb employee, and then member of TC39, the technical commitee in charge of the design of ECMAScript, of which JavaScript is an offshoot. These conversations essentially consist in him providing *ad hoc* explanations when external users are confused about the purpose of a rule or, even further, in re-organizing their code[[85]](#footnote-86)[[86]](#footnote-87). In this case, ljharb not only acts as a community manager rather than a project maintainer *per se*, but also eschews any discussions based on subjective preferences by providing a technical solution to any question asked, therefore showing that the airbnb/javascript style guide is not only exhaustive, consistent, but also implementable.

What we’ve seen in airbnb/javascript here is that the technical environment of GitHub issues have been re-purposed from actual bug-tracking for Airbnb’s employees, towards rather as a didactic platform using a pedagogical approach, in a balance of both closed-source documents and open-source practices. The combination between one canonical document—uneditable beyond those with privileges, despite GitHub’s pull request mechanism—which exhaustively covers almost all controversial use-cases in the language, along with a skilled maintainer explaining issues to the commenters individually (in a reversal of the classical format of a bug in the software which needs to match the code, into a bug in the code, which needs to match the software), and ultimately the recourse to the process of forking as a concluding argument; all of these can explain both airbnb/javascript’s popularity and its high number of forks.

### StandardJS

standard/standard is the third most popular project in the JavaScript ecosystem, and offers a different approach. While airbnb/airbnb provides a main README.md, with all the style rules immediately available at first glance, along with a eslintrc.json configuration file as the result of the closed-source work of a private company, standard/standard puts forth the standard binary, an immediate, “out-of-the-box” solution which applies the project’s rules to any file where the program is executed[[87]](#footnote-88). The overall approach comes out of a more traditional open-source pattern, initiatet and spearheaded by a single individual, feross, which then turned into a proper organization once enough community traction had been garnered[[88]](#footnote-89)[[89]](#footnote-90).

#### Goals

The self-stated goal of standard/standard of *“no configuration”*[[90]](#footnote-91)[[91]](#footnote-92) has led its maintainers (mainly feross, along with rotating members such as linusU and rstacruz) to inflect their discourse under the influence of technical efficiency, rather than rule justification as airbnb/javascript does. Indeed, the rules themselves are both stated by feross as well as discussed by the community—groups of users, members and contributors—for each release of the package. By leaving open the possibility of modifying their style rules, and subsequently making them immediately enforceable by their standard binary, the discursive focus isn’t so the number of discussions of so-called “religious” issues, known for the amount of devotion and heat that they attract (e.g. semicolons[^70][[92]](#footnote-93)[^72], parentheses[[93]](#footnote-94)[[94]](#footnote-95), indentation[[95]](#footnote-96)), but rather on technical implementation, alternative possibilities and community support.

#### Technical requirement from ESLint

The first category of arguments laid out in the standard/standard repository is related to the actual, technical feasibility of the arbitrary[[96]](#footnote-97), but not non-sensical, rules laid out in the README.md. The status of feross as *“benevolent dictator”*[[97]](#footnote-98)[[98]](#footnote-99)[[99]](#footnote-100) nullifies most of the arguments (e.g. *“All very subjective though. I think @feross will just have to pick something :)”*[[100]](#footnote-101)), the remaining of which can sometimes be rationally discussed[[101]](#footnote-102)[[102]](#footnote-103). Given this status of most rules being somewhat indiscutable, upon what does the agreement of standard/standard rules rely? The status of the project as a convenient wrapper (i.e. a seamless *intermediary object* in a developer’s workflow) around eslint poses that core technology as one of the final assessments of the validity of a given argument (e.g. *“Probably better off making your case at eslint first, then coming back here”*[[103]](#footnote-104)). While this effectively avoids bikeshedding[[104]](#footnote-105) in cases where the ultimate goal is to settle with one choice rather than debating multiple of them, this technological dependency also prevents some desired changes[[105]](#footnote-106).

#### Package ecosystem justifications

If the stylistic preference of a commenter still weighs more than the ease-of-use of the standard/standard package, then the strategy of the maintainers is to redirect them towards another part of the ecosystem, be it the direct configuration file which implements all of the style rules[[106]](#footnote-107), or the adoption of other packages maintained by the standard organization[[107]](#footnote-108)[[108]](#footnote-109)[[109]](#footnote-110)[[110]](#footnote-111). Different from the dismissive tone of the airbnb/javascript maintainer’s repository, this redirection is an acknowledgement of the fact that opinions matter less than building a community, in line with the open-source ethos of group participation (e.g. the use of the *“we”* pronoun by feross[[111]](#footnote-112)). Indeed, the word *community* is used most often on airbnb/javascript by the commenteers, while it is used most often on standard/standard by the maintainers (given an equivalent number of issues on each, respectively 34[[112]](#footnote-113) vs. 39[[113]](#footnote-114)). This emphasis on community-building as a part of the overall strategy of standard/standard also affects their discursive strategy.

#### Community-breaking changes

With a set of rules overwhelmingly decided upon by feross, some rational argumentations for the modification of these rules, the reliance on eslint as a ground for whether or not a rule can be applied, and the redirection of unsatisfied, or unconvinced community members towards other packages of the standard ecosystem, such as standard/standardx, standard/semistandard or standard/doublestandard for “religious” issues, the last remaining argument deployed for the adoption or rejection of style rules is the extent to which the community is using the existing rules.

On the one hand, this project has garnered visibilty and is currently deeply embedded in an institutional network of projects using the standard/standard guide[[114]](#footnote-115), including GitHub itself for its JavaScript project, and such a network provides credibility by association[[115]](#footnote-116). Even though these associations have been documented in order to further grow the community of users[[116]](#footnote-117).

In practice, the biggest role that the community has in the negotiation of style guides is in whether or not any change in the guide would be a breaking change, and for how many projects using standard/standard. Used by project maintainers (*“This is one of those decisions that we can’t revisit. Nearly every repo that uses standard would break, and that’s not acceptable – even with major version bump.”*[[117]](#footnote-118), *“Although personally I agree with the fundamental reasons that you argue for this. As it stands, this would be so much of a breaking change, it will never be accepted.”*[[118]](#footnote-119)). The closing argument is therefore the number of existing projects which would have to refactor their code in order to comply with the new rule, and is often the last comment to take place on an issue before that issue is closed[[119]](#footnote-120)[[120]](#footnote-121)[^95], acting as a unilateral reason.

The discursive strategies that the commenters and maintainers deploy in standard/standard then revolve around issues of convenience, which have enabled what could be called a certain form of social path-dependency[[121]](#footnote-122), in a productive sense. The figure and work of feross, both agreeable and engaging in his exchanges and unilateral in his decisions, include in their discourse references to technical limitations in order to bypass subjective issues. Indeed, while the affordances of eslint are inherently digital, the adherence of silent majority of standard/standard users is represented in a quantified manner, through unit tests resulting the a percentage of organizations and packages failing or succeeding said tests; and such an approach seems to manifest itself as the reification of community choices in order to increase its effective discursive power.

### Prettier

The prettier/prettier project is both the most recent and the most popular (based on both stars and weekly downloads[[122]](#footnote-123)), and offers a third approach in our analysis. Developped in 2017 by Facebook employees, which is one of the world’s largest and most profitable companies, as well as the maintainer of the react project, the most popular front-end development framework by the end of 2019[[123]](#footnote-124). Beyond this institutional backing and internal success[[124]](#footnote-125), prettier/prettier presents two specificities which differentiate from airbnb/javascript and standard/standard. First, it doesn’t offer a clear and accessible style guide itself, and its README.md only contains one example to show the kind of work prettier/prettier does. Second, what it does is *essentially* different than the two previous examples looked, since it analyzes the source code, parses it and its inconsistencies and entirely reformats it according to internal rules—neither airbnb/javascript nor standard/standard offer default reformatting solutions. Such a language-independent approach (dealing directly with *Abstract Syntax Trees*, a context-free structure) might explain its popularity across different langauges, while the two other projects focus mainly on JavaScript. So, while not a style guide *de jure*[[125]](#footnote-126), it is *de facto* one, through its enforcement of rules, an enforcement much more thorough than its alternatives.

#### The co-creation of a style guide

Along with the absence of a clear styleguide announcing, if not explaining, the individual style decisions, the maintainers of the project (e.g. jlongster, vjeux, founders of the project while Facebook employees) tend to engage in conversations about the possibility to change existing styling rules[[126]](#footnote-127)[[127]](#footnote-128)[[128]](#footnote-129). This willingness to change the behavior of their tool, along with its automation power and the founders institutional capital might explain the fact that the prettier/prettier repository has about four times more issues in total than the two other repositories examined, with more thoughtful, thorough and rational explanations[[129]](#footnote-130)[[130]](#footnote-131), including detailed description of desired behaviour[[131]](#footnote-132). In particular, prettier/prettier distinguishes itself by allowing contributors to open up discussions and negotiate changes not through pure discursive argumentation, but rather by offering practical and thorough implementations of an alternative[[132]](#footnote-133), including atlernatives which concern “religious” issues[[133]](#footnote-134)—in effect, long and elaborate argumentation always require *code* examples[[134]](#footnote-135)[[135]](#footnote-136). The process seems to be as follows: prettier/prettier formats a given code, the developer who might not agree with such a formatting result opens up an issue, argues for her opinion through the presentation of practical use-cases and actionable pull-requests and, depending on the technical soundess of the solution and the size of the community already using the package, might get their change accepted. Finally, while the thoroughness and politeness of most replies on this repository point towards a healthy community, it should be noted that the verbosity of some of the contributions are considered intimidating for comments who consider they English level sub-par(e.g. *“I didn’t read the whole conversation above because you guys really know how to speak English well slightly\_smiling\_face Makes me envy a bit”*[[136]](#footnote-137)).

#### Technical efficiency as an argument

Without a reference document, and with the recurring statement that “readability” and “preference” are subjective arguments, prettier/prettier relies on technical arguments to solve debates, sometimes even before they happen. Before they even happen, because the seamlessness of the integration, as both a visible and invisible mediation, can eschew those debates altogether (*“Hi @jlongster, just would like to chime in and let you know that your project has hit the jackpot and solved something that even (the self-proclaimed) Standard and Semi-Standard couldn’t solve for me […] Easy team-wide style enforcement that’s based on an unbiased 3rd party algorithm to produce a one and only definitive way to indent absolutely each line of code”*[[137]](#footnote-138)). This immediacy of programmatic action, and its adoption in return, is reminiscent of the power of interfaces in their ambiguity, in which a self-implementing tool acts as its own rationale[[138]](#footnote-139).

However, most of the official discourse of prettier/prettier (through its website and through oft-referenced conference talks) presents the project not as a style-guide *per se*, but as an AST-parser and printer. These technical definitions steer the debate towards technical argumentation, rather than adoption by the community (such as standard), or explanation of practical raitonale (such as airbnb); explicitly mentioning AST representation often becomes the grounds for the final stylistic decision (*“It will be interesting to figure this out because the way it formats it now is because of how the AST is structured, which is based on operator precedence”*[[139]](#footnote-140)[[140]](#footnote-141)[[141]](#footnote-142)[[142]](#footnote-143)), and sometimes even used as counter-examples[[143]](#footnote-144).

Finally, prettier/prettier’s argument is that it is both opionated and efficient, which has led to the early development of a “no-options” philosophy[[144]](#footnote-145), which has moved from a read-write text, discussed by both maintainers and commenters, to a read-only text, as a canonical text on the official website[[145]](#footnote-146), which is then used subsequently as an argument to respond negatively to requests for personal additions[[146]](#footnote-147)[[147]](#footnote-148)[[148]](#footnote-149), combined with a request of responding through programming languages, rather than human languages[[149]](#footnote-150). This particular phenomenon echoes practices found in the airbnb/javascript project in Goody’s investigations of the immutability and canonization of written texts. Because the tool gets its value and usefulness from its invariability—and the communities which depend on its invariability[[150]](#footnote-151)—, discussions around the tool itself can be made moot.

#### Size of community and noise

A final feature of the prettier/prettier discussions stems from both their exhaustiveness, the popularity of the project and the maintainers’ decision to no longer accept options[[151]](#footnote-152). This results in several seemingly intractable discussions with over 100 comments, while the majority of other issues have less than 10 comments each)[[152]](#footnote-153)[[153]](#footnote-154)[[154]](#footnote-155)[[155]](#footnote-156). These issues bring up the original tension of a styleguide, in which each writer has an opinion[[156]](#footnote-157) on what is the best way to solve a given dilemma[[157]](#footnote-158), either as emotional, subjective statements[[158]](#footnote-159) (*“This issue is ridiculous. Just fix that”*[[159]](#footnote-160)), or along with examples[[160]](#footnote-161) and offers of pull requests. The pattern we see here is that, due to the popularity of the project, first-time users weigh in on complicated debates, engage in the conversation with other commenters for a couple of replies, and then drop out of the discussion altogether). Coupled with the original intent of the maintainers to integrate community-argued changes, these issues remain open for all to chime in (most of them being over 3 years old).

On the one hand, these discursive strategies seems to parallel the pattern of the dreaded *bikeshedding*[[161]](#footnote-162), in which anyone can chime in including linguistic patterns such as *“just my two cents”*, *“IMHO”* (“in my humble opinion”), *“just gonna weigh in here”*, apparently. As such, they don’t seem to appear very productive at first glance due to repeated arguments[[162]](#footnote-163)[[163]](#footnote-164) and affect the maintainer’s ability to discuss (*“The Prettier issue tracker naturally attracts and concentrates code style discussions. They’ve been moved from basically everywhere to here. I think this is a big factor to the maintainer burnout I’ve seen lately.”*[[164]](#footnote-165)).

On the other hand, however, the overall quality of arguments presented (combining use cases, opinions and responses to previous proposals) has led to a switch in the discursive strategy in which the discussion is either summed up by the maintainers[[165]](#footnote-166)[[166]](#footnote-167)[[167]](#footnote-168) in order to provide moderation to the debate or to be pursued privately by the maintainers[[168]](#footnote-169)[[169]](#footnote-170). The result is that, emerging from community-created discursive noise inherent to any popular open-source projects, the prettier/prettier team engages less in terms of justifying the existing style, but in gauging how feasible an alternative is, once this discursive noise reaches a particular threshold, acting as a filter to turn quantitative input into qualitative decision making, the existence of the issue itself acting as an argument.

## Findings

Style guides existing on GitHub have to tend to the specific issue of convincing individual users with strong subjective preference of adopting their recommendations in the midst of a technical environment which favors copying and customization. Starting from an approach of arguments based on the framework of *communicative competence*, we’ve highlighted that, beyond emotional and subjective statements (sometimes disguised as rational ones) as well as the social justification of institutional origin or belonging, rational statements and commands are being affected, to some extent, by the techincal *milieu* in which programmers write, read and work.

Particularly, we can see that there are multiple strategies which involve closely either source code, or compiled code. Source code acts as a prime argument when human languages fail to communicate the objectivity of their statement, and is often required by all parties of a discussion to base statements on what is considered a fact. Still, those comments are often overlooked: the requirement of *readability*, crucial to a decision in style guides, re-appears in all its subjectivity when it comes to reading the source code example provided, pitting the argument in multi-sided subjective perspectives. In airbnb/javascript’s case, when re-written code can act as an argument to the validity of the style guide, it is because previous code didn’t *work*, and became functional again through ljharb’s work.

If source code seems to be a novel type of argument, adding to the traditional realm of human rhetoric, and yet remains trapped in subjective appreciation, compiled code can act as its own argument. In both standard/standard and prettier/prettier, both of the projects rely on the fact that their style guides *work*, that they are efficient in what they do, and any change must first and foremost comply with the *feasibiliy* of the argument, no matter how sound the conceptual proposal is; as we’ve seen, prettier/prettier can replace a didactic guide by an efficient tool, while airbnb/javascript has to rely on a strict didactic approach, on top of providing an eslint configuration file, leading us to the consideration that code as actionable words, a silent argument of sorts, is perhaps the most efficient in the adoption of a project.

In terms of GitHub as a platform, the structure within which these projects take place, we’ve observed a multiplicity of approaches, preventing a unilateral interpretation of how technical environments can entirely shape the nature of a discussion. Between airbnb/javascript’s use of issues as a personalized question-answering forum, standard/standard’s desire to fix practical implementations of the tool they provide, and prettier/prettier’s use of a more classical open-source discussion, GitHub’s platform seems to be able to provide multiple kinds of discussions, ultimately geared by the social role of the maintainers and the ability to toggle between references to read-only (canonical) texts rather than keeping the discussion within read-write texts.

However, GitHub does provide the interesting case of a highly-networked, highly-transparent working environment. This study has examined the role of a quantified community in argumenting for or against a particular stylistic choice, particularly present in standard/standard’s strategy, but also present in prettier/prettier; it is necessary to mention the entangled nature of cross-references to other projects hosted on GitHub. Beyond naming exisiting alternatives[[170]](#footnote-171), being able to reference another project on an issue[[171]](#footnote-172), or to mention explicitly the name of a maintainer of one given project on a different project[[172]](#footnote-173), facilitates the interactions of maintainers[[173]](#footnote-174) and allows for an intricate web of intertextualities, allowing discussions happening on a given repository to act as arguments for another repository, or vice-versa. This added layer of complexity is beyond the scope of the current study, but offers a promising field for future research.

* calls to “fork” are always the conclusion of an argument
* connect to organizational style, communication, etc.
* add reference to the CNRS book
* develop on how the style guide embodies administrative work -> **how?** -> CSCW, smoothing out kinks

[54]: [https://github.com/airbnb/javascript/issues/1741#issuecomment-367442335]

[69]: [https://github.com/airbnb/javascript/issues/851#issuecomment-213834028]

[70]: [https://github.com/standard/standard/issues/78]

[72]: [https://github.com/standard/standard/issues/176#issuecomment-116312615]

1. Nather, Ed (2003-09-12) [1983-05-21], The story of Mel, a Real Programmer, FOLDOC [↑](#footnote-ref-2)
2. this is an inline footnote [↑](#footnote-ref-3)
3. https://www.gewina-studium.nl/articles/abstract/10.18352/studium.1459/ [↑](#footnote-ref-4)
4. ref needed [↑](#footnote-ref-5)
5. ref needed [↑](#footnote-ref-6)
6. ref needed - formal institutions (goody?) [↑](#footnote-ref-7)
7. https://github.com/airbnb/javascript [↑](#footnote-ref-8)
8. https://github.com/standard/standard [↑](#footnote-ref-9)
9. https://github.com/prettier/prettier [↑](#footnote-ref-10)
10. objets intermediaires [↑](#footnote-ref-11)
11. simmel, the problem of style [↑](#footnote-ref-12)
12. ruskin, seven lamps [↑](#footnote-ref-13)
13. sennett, the craftsman [↑](#footnote-ref-14)
14. clean code, martin [↑](#footnote-ref-15)
15. wendy huy kyong chun [↑](#footnote-ref-16)
16. computer power and human reason [↑](#footnote-ref-17)
17. the lives of artists [↑](#footnote-ref-18)
18. the blog post where they compare picasso, matisse, etc. [↑](#footnote-ref-19)
19. the lives of artists [↑](#footnote-ref-20)
20. djikstra [↑](#footnote-ref-21)
21. ref needed [↑](#footnote-ref-22)
22. more books on professional dev [↑](#footnote-ref-23)
23. IdiomaticJS readme [↑](#footnote-ref-24)
24. knuth on style [↑](#footnote-ref-25)
25. goody [↑](#footnote-ref-26)
26. manovich [↑](#footnote-ref-27)
27. frankel, 2006 [↑](#footnote-ref-28)
28. Brousseau et Moatty (2003), in [https://www.cairn.info/revue-anthropologie-des-connaissances-2012-1-page-21.htm?contenu=article] [↑](#footnote-ref-29)
29. G. Hendry, “Public participation in proprietary software developmentthrough user roles and discourse,” Int. J. Hum.-Comput. Stud., vol. 66,no. 7, pp. 545–557, Jul. 2008 [↑](#footnote-ref-30)
30. wealth of networks [↑](#footnote-ref-31)
31. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository [↑](#footnote-ref-32)
32. Gillespie, T. What are platforms? [↑](#footnote-ref-33)
33. Influence of social and technical factors for evaluating contribution in GitHub, [https://doi.org/10.1145/2568225.2568315] [↑](#footnote-ref-34)
34. [https://doi.org/10.1016/j.jss.2018.09.016] [↑](#footnote-ref-35)
35. [https://github.com/rwaldron/idiomatic.js/] [↑](#footnote-ref-36)
36. [https://google.github.io/styleguide/jsguide.html] [↑](#footnote-ref-37)
37. [https://www.crockford.com/style.html] [↑](#footnote-ref-38)
38. [https://www.nytimes.com/2020/08/19/technology/airbnb-ipo.html] [↑](#footnote-ref-39)
39. [https://github.com/standard/standard/blob/master/RULES.md] [↑](#footnote-ref-40)
40. [https://prettier.io/docs/en/] [↑](#footnote-ref-41)
41. [https://prettier.io/docs/en/install.html] [↑](#footnote-ref-42)
42. Barthes, La plissure du texte [↑](#footnote-ref-43)
43. [https://guides.github.com/features/issues/] [↑](#footnote-ref-44)
44. got issues? who cares about it? [↑](#footnote-ref-45)
45. Let’s Talk About It: Evaluating Contributions through Discussion in GitHub [↑](#footnote-ref-46)
46. M. Hilton, T. Tunnell, K. Huang, D. Marinov and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 2016, pp. 426-437. [↑](#footnote-ref-47)
47. [https://www.npmjs.com/package/eslint] [↑](#footnote-ref-48)
48. Latour, Woolgar, 1979 [↑](#footnote-ref-49)
49. Jeantet, A., 1998. Les objets intermédiaires dansla conception. Elémentspour une sociologie desprocessus de conception. Sociologie du travail,n°3/98, pp. 291-316 [↑](#footnote-ref-50)
50. Vinck, D. (2009). De l’objet intermédiaire à l’objet-frontière: Vers la prise en compte du travail d’équipement. Revue d’anthropologie des connaissances, vol. 3, 1(1), 51-72. https://doi.org/10.3917/rac.006.0051 [↑](#footnote-ref-51)
51. lessig, code is law [↑](#footnote-ref-52)
52. galloway, protocol [↑](#footnote-ref-53)
53. Bourdieu, Ce que parler veut dire [↑](#footnote-ref-54)
54. Habermas, Theory of Communicative action [↑](#footnote-ref-55)
55. David Berry [↑](#footnote-ref-56)
56. Some computational approach to github [↑](#footnote-ref-57)
57. Some other [↑](#footnote-ref-58)
58. Perceived Language Complexity in GitHub IssueDiscussions and Their Effect on Issue Resolution [↑](#footnote-ref-59)
59. The Practice of Everyday Life, Michou de Certeau [↑](#footnote-ref-60)
60. [https://github.com/airbnb/javascript/issues/1953#issue-379414953] [↑](#footnote-ref-61)
61. [https://github.com/airbnb/javascript/issues/1967] [↑](#footnote-ref-62)
62. [https://github.com/airbnb/javascript/issues/2261] [↑](#footnote-ref-63)
63. [https://github.com/airbnb/javascript/issues/828] [↑](#footnote-ref-64)
64. [https://github.com/airbnb/javascript/pull/455] [↑](#footnote-ref-65)
65. [https://github.com/airbnb/javascript/issues/1532#issuecomment-326755515] [↑](#footnote-ref-66)
66. [https://github.com/airbnb/javascript/issues/1365#issuecomment-350163209] [↑](#footnote-ref-67)
67. [https://github.com/airbnb/javascript/issues/1660#issuecomment-419087604] [↑](#footnote-ref-68)
68. [https://github.com/airbnb/javascript/issues/1185#issuecomment-262994841] [↑](#footnote-ref-69)
69. [https://github.com/airbnb/javascript/issues/4] [↑](#footnote-ref-70)
70. [https://github.com/airbnb/javascript/issues/18] [↑](#footnote-ref-71)
71. [https://github.com/airbnb/javascript/issues/9] [↑](#footnote-ref-72)
72. [https://github.com/airbnb/javascript/pull/10] [↑](#footnote-ref-73)
73. E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer and D. M. German, “Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub,” 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 574-585, doi: [https://doi.org/10.1109/ICSE.2015.74] [↑](#footnote-ref-74)
74. [https://github.com/airbnb/javascript/issues/1089] [↑](#footnote-ref-75)
75. [https://github.com/airbnb/javascript/issues/102#issuecomment-28157738] [↑](#footnote-ref-76)
76. [https://github.com/airbnb/javascript/issues/102#issuecomment-28259657] [↑](#footnote-ref-77)
77. [https://github.com/airbnb/javascript/issues/1365#issuecomment-663973664] [↑](#footnote-ref-78)
78. [https://github.com/airbnb/javascript/issues/1982#issuecomment-451191979] [↑](#footnote-ref-79)
79. [https://github.com/airbnb/javascript/issues/269] [↑](#footnote-ref-80)
80. [https://github.com/airbnb/javascript/issues/1880] [↑](#footnote-ref-81)
81. [https://github.com/airbnb/javascript/issues/2302#issuecomment-703436286] [↑](#footnote-ref-82)
82. [https://github.com/airbnb/javascript/issues/2191#issuecomment-596139441] [↑](#footnote-ref-83)
83. [https://github.com/airbnb/javascript/issues/1842#issuecomment-400194978] [↑](#footnote-ref-84)
84. [https://github.com/airbnb/javascript/issues/21] [↑](#footnote-ref-85)
85. [https://github.com/airbnb/javascript/issues/1103#issuecomment-400711388] [↑](#footnote-ref-86)
86. [https://github.com/airbnb/javascript/issues/851#issuecomment-215479719] [↑](#footnote-ref-87)
87. [https://github.com/standard/standard/issues/94#issuecomment-87332587] [↑](#footnote-ref-88)
88. [https://github.com/standard/standard/issues/846] [↑](#footnote-ref-89)
89. [https://github.com/standard/standard/issues/259#issuecomment-141881255] [↑](#footnote-ref-90)
90. [https://github.com/standard/standard/blob/master/README.md] [↑](#footnote-ref-91)
91. [https://github.com/standard/standard/issues/3#issuecomment-71952384] [↑](#footnote-ref-92)
92. [https://github.com/standard/standard/issues/962] [↑](#footnote-ref-93)
93. [https://github.com/standard/standard/issues/414#issuecomment-183459932] [↑](#footnote-ref-94)
94. [https://github.com/standard/standard/issues/219#issuecomment-170877700] [↑](#footnote-ref-95)
95. [https://github.com/standard/standard/issues/58#issuecomment-77710035] [↑](#footnote-ref-96)
96. [https://github.com/standard/standard/issues/3#issuecomment-71950165] [↑](#footnote-ref-97)
97. [https://github.com/standard/standard/issues/108#issuecomment-90990325] [↑](#footnote-ref-98)
98. [https://github.com/standard/standard/issues/710#issuecomment-521095379] [↑](#footnote-ref-99)
99. [https://github.com/standard/standard/issues/628#issuecomment-366484860] [↑](#footnote-ref-100)
100. [https://github.com/standard/standard/issues/309#issuecomment-152899586] [↑](#footnote-ref-101)
101. [https://github.com/standard/standard/issues/111#issuecomment-116885571] [↑](#footnote-ref-102)
102. [https://github.com/standard/standard/issues/309#issuecomment-180208214] [↑](#footnote-ref-103)
103. [https://github.com/standard/standard/issues/720#issuecomment-266878246] [↑](#footnote-ref-104)
104. [https://github.com/standard/standard/issues/1356#issue-480058723] [↑](#footnote-ref-105)
105. [https://github.com/standard/standard/issues/257#issuecomment-142417059] [↑](#footnote-ref-106)
106. [https://github.com/standard/standard/issues/240#issuecomment-224125128] [↑](#footnote-ref-107)
107. [https://github.com/standard/standard/issues/771#issuecomment-375609384] [↑](#footnote-ref-108)
108. [https://github.com/standard/standard/issues/1155#issuecomment-399769283] [↑](#footnote-ref-109)
109. [https://github.com/standard/standard/issues/962#issuecomment-319714580] [↑](#footnote-ref-110)
110. [https://github.com/standard/standard/issues/1500#issuecomment-648019851] [↑](#footnote-ref-111)
111. [https://github.com/standard/standard/issues/1356#issue-480058723] [↑](#footnote-ref-112)
112. [https://github.com/airbnb/javascript/issues?q=is%3Aissue+is%3Aclosed+community] [↑](#footnote-ref-113)
113. [https://github.com/standard/standard/issues?q=is%3Aissue+is%3Aclosed+community] [↑](#footnote-ref-114)
114. [https://github.com/standard/standard#who-uses-javascript-standard-style] [↑](#footnote-ref-115)
115. sth about bourdieu and symbolic belonging idk [↑](#footnote-ref-116)
116. [https://github.com/standard/standard/issues/744#issue-200237553] [↑](#footnote-ref-117)
117. [https://github.com/standard/standard/issues/219#issuecomment-127446961] [↑](#footnote-ref-118)
118. [https://github.com/standard/standard/issues/240#issuecomment-135968036] [↑](#footnote-ref-119)
119. [https://github.com/standard/standard/issues/298#issuecomment-179571496] [↑](#footnote-ref-120)
120. [https://github.com/standard/standard/issues/720#issuecomment-515722463] [↑](#footnote-ref-121)
121. [https://github.com/prettier/prettier/issues/40#issuecomment-271769512] [↑](#footnote-ref-122)
122. [https://www.npmjs.com/package/prettier] [↑](#footnote-ref-123)
123. [https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190] [↑](#footnote-ref-124)
124. [https://github.com/prettier/prettier/issues/5377#issuecomment-566173911] [↑](#footnote-ref-125)
125. [https://github.com/prettier/prettier/issues/5246#issuecomment-429788464] [↑](#footnote-ref-126)
126. [https://github.com/prettier/prettier/issues/7884#issuecomment-605519802] [↑](#footnote-ref-127)
127. [https://github.com/prettier/prettier/issues/73#issuecomment-272537791] [↑](#footnote-ref-128)
128. [https://github.com/prettier/prettier/issues/187#issuecomment-313229534] [↑](#footnote-ref-129)
129. [https://github.com/prettier/prettier/issues/840#issuecomment-689978905] [↑](#footnote-ref-130)
130. [https://github.com/prettier/prettier/issues/4125#issuecomment-588725113] [↑](#footnote-ref-131)
131. [https://github.com/prettier/prettier/issues/3368#issuecomment-357312678] [↑](#footnote-ref-132)
132. [https://github.com/prettier/prettier/issues/3368#issuecomment-374764905] [↑](#footnote-ref-133)
133. [https://github.com/prettier/prettier/issues/736] [↑](#footnote-ref-134)
134. [https://github.com/prettier/prettier/issues/7884#issuecomment-605898046] [↑](#footnote-ref-135)
135. [https://github.com/prettier/prettier/issues/5814#issuecomment-468978755] [↑](#footnote-ref-136)
136. [https://github.com/prettier/prettier/issues/7884#issuecomment-619147696] [↑](#footnote-ref-137)
137. [https://github.com/prettier/prettier/issues/40#issuecomment-271804674] [↑](#footnote-ref-138)
138. galloway, the interface effect [↑](#footnote-ref-139)
139. [https://github.com/prettier/prettier/issues/2#issuecomment-269872486] [↑](#footnote-ref-140)
140. [https://github.com/prettier/prettier/issues/5814#issuecomment-468827633] [↑](#footnote-ref-141)
141. [https://github.com/prettier/prettier/issues/9358#issuecomment-705940711] [↑](#footnote-ref-142)
142. [https://github.com/prettier/prettier/issues/187#issuecomment-281097771] [↑](#footnote-ref-143)
143. [https://github.com/prettier/prettier/issues/7884#issuecomment-605755932] [↑](#footnote-ref-144)
144. [https://github.com/prettier/prettier/issues/40] [↑](#footnote-ref-145)
145. [https://prettier.io/docs/en/option-philosophy.html] [↑](#footnote-ref-146)
146. [https://github.com/prettier/prettier/issues/8559#issuecomment-643796145] [↑](#footnote-ref-147)
147. [https://github.com/prettier/prettier/issues/187#issuecomment-627906479] [↑](#footnote-ref-148)
148. [https://github.com/prettier/prettier/issues/840#issuecomment-458280945] [↑](#footnote-ref-149)
149. [https://github.com/prettier/prettier/issues/840#issuecomment-482353176] [↑](#footnote-ref-150)
150. [https://github.com/prettier/prettier/issues/5377#issuecomment-646868677] [↑](#footnote-ref-151)
151. [https://github.com/prettier/prettier/pull/8540s] [↑](#footnote-ref-152)
152. [https://github.com/prettier/prettier/issues/5377] [↑](#footnote-ref-153)
153. [https://github.com/prettier/prettier/issues/187] [↑](#footnote-ref-154)
154. [https://github.com/prettier/prettier/issues/840] [↑](#footnote-ref-155)
155. [https://github.com/prettier/prettier/issues/5814] [↑](#footnote-ref-156)
156. [https://github.com/prettier/prettier/issues/187#issuecomment-282690495] [↑](#footnote-ref-157)
157. [https://github.com/prettier/prettier/issues/187#issuecomment-345493031] [↑](#footnote-ref-158)
158. [https://github.com/prettier/prettier/issues/5377#issuecomment-669877250] [↑](#footnote-ref-159)
159. [https://github.com/prettier/prettier/issues/187#issuecomment-385114296] [↑](#footnote-ref-160)
160. [https://github.com/prettier/prettier/issues/187#issuecomment-350849520] [↑](#footnote-ref-161)
161. [https://www.freebsd.org/doc/en/books/faq/misc.html#bikeshed-painting] [↑](#footnote-ref-162)
162. [https://github.com/prettier/prettier/issues/187#issuecomment-383546996] [↑](#footnote-ref-163)
163. [https://github.com/prettier/prettier/issues/5814#issuecomment-597403394] [↑](#footnote-ref-164)
164. [https://github.com/prettier/prettier/issues/840#issuecomment-522295504] [↑](#footnote-ref-165)
165. [https://github.com/prettier/prettier/issues/840#issuecomment-618186522] [↑](#footnote-ref-166)
166. [https://github.com/prettier/prettier/issues/5814#issuecomment-469736563] [↑](#footnote-ref-167)
167. [https://github.com/prettier/prettier/issues/5377#issuecomment-650607819] [↑](#footnote-ref-168)
168. [https://github.com/prettier/prettier/issues/5377#issuecomment-669877250] [↑](#footnote-ref-169)
169. [https://github.com/prettier/prettier/issues/5377#issuecomment-650153308] [↑](#footnote-ref-170)
170. [https://github.com/prettier/prettier/pull/123#issuecomment-272029925] [↑](#footnote-ref-171)
171. [https://github.com/prettier/prettier/issues/3806#issuecomment-451615213] [↑](#footnote-ref-172)
172. [https://github.com/standard/standard/issues/811#issuecomment-294034120] [↑](#footnote-ref-173)
173. [https://github.com/prettier/prettier/issues/3503#issuecomment-352538566] [↑](#footnote-ref-174)