# How do Scratch Users Name Variables and Procedures?

Author1
Uni1
Address1
Email: email1.com

Author2
Uni2
Address2
Email: email2.com

***Abstract*—Research shows the importance of choosing good names to identifiers in software code. More meaningful names improve the comprehension and readability of software code, which leads to increased efficiency in maintenance tasks. In particular, several guidelines encourage long and descriptive variable names. A recent study analyzed the use of variable names in five popular programming languages, with a focus on single-letter variable names because of the apparent contradiction between their frequent use and the fact that these variables violate the aforementioned guidelines.**

**In this paper, we focus on single-letter variables in Scratch, a popular block-based visual language which focuses on teaching children programming. We start by replicating the previously mentioned study for Scratch. We augment this study by analyzing single-letter procedure names, as well as by investigating the use of Scratch specific naming patterns including spaces in variable names and textual labels in procedure definitions and calls.**

**The results of our analysis show that Scratch users often use variable and procedure names between 4 and 10 characters in length. For the single letter variables, the most recurring names are x, y and i. While single letter procedures are less popular, the usage of letters is not linked with one character. *I do not understand the meaning of the previous sentence***

**Concerning Scratch specific features, 20% of projects have variables that include spaces in the name. The usage of textual string between parameters appears as not so common, however textual patterns used imply an inference from textual languages by using brackets for example. Finally, when compared to the other programming languages, Scratch variable names tend to be longer than of the mainstream programming languages, and the usage of single uppercase letters seems to be very similar to the pattern found in Perl, while for the lowercase letters—to the pattern found in Java.**

***AS: I'm missing conclusion, some kind of take away message.***

## I. Introduction

The naming of identifiers in the source code has been extensively studied (see, e.g., recent studies of this subject [1], [2], [3], [4], [5], [6], [7], [8]). Still, the impact of the variable name choice on code readability and maintainability is controversial, as witnessed, e.g., by recent studies of Beniamini et al. [3] and Hofmeister et al. [5] reaching contradictory conclusions. Furthermore, computer science and programming education seems to focus on the programming concepts and the syntax of the languages as opposed to practices in naming variables and identifiers. Indeed, while "meaningful variable names" are advocated by some teachers [9] and practitioners [10] neither the ACM Curriculum Guidelines for Undergraduate Programs in Computer Science[1] nor the Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering[2] discuss this topic. In fact, standard metasyntactic variables used in syntax examples are *"foo"* and *"bar"* [11]. The names of these identifiers are meaningless, and to some extent, they represent a refusal to name, suggesting the learner that naming is less important, or irrelevant, to the programming task.

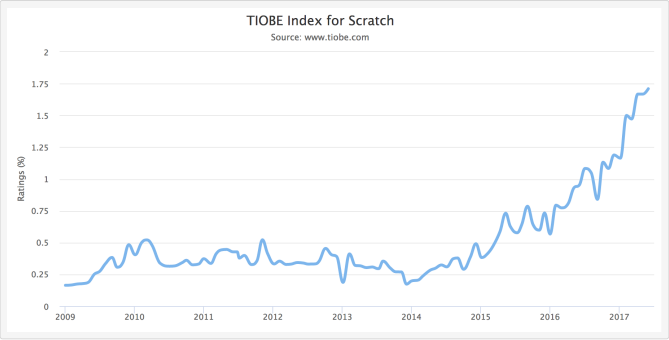


Fig. 1. TIOBE Programming Community index: evolution of the popularity of Scratch: https://www.tiobe.com/tiobe-index/scratch/

In this paper, **we analyze the use of variable and procedure names in Scratch**. Scratch is a block-based visual language that was developed by MIT with the aim of helping young people learn the basic concepts of programming and collaboration. Scratch has recently become very popular among school-age children and even in introduced as part of the school curriculum as a means to teach programming [12]. Moreover, the overall popularity of Scratch is witnessed by Scratch being currently rated 19 in the TIOBE index[3], topping such languages as Lua, Scala and Groovy and since early 2014 exhibiting an increasing trend shown in Fig. 1.

We believe that there are several reasons why understanding the naming practices in the Scratch community is important. First, it is important for the Scratch community itself as bad naming practices can easily propagate from one program to another through 'remixing' [13], [14], code sharing practice similar to GitHub forking. Second, it is important for re-

[1] http://www.acm.org/education/CS2013-final-report.pdf
[2] http://www.acm.org/binaries/content/assets/education/se2014.pdf
[3] https://www.tiobe.com/tiobe-index/

searchers. Software engineering researchers can learn how to support novice programmers, taking their first steps in programming. Researchers on software engineering education can obtain insights in how to define naming guidelines for educational materials, and analyzing the differences between Scratch and textual languages can help in reasoning about the transition from block-based languages to textual ones.

We start by a **general discussion of naming practices** in Scratch and analyze the previously published collection of 250,000 Scratch projects [15]. We replicate two studies from a recent paper by Beniamini et al. [3]. Similarly to Beniamini et al. we investigate the distribution of the lengths of variable names and study popularity of single-letter variable names such as $i$ and $x$. As opposed to Beniamini et al. that focused on variable naming in five mainstream programming languages we focus on Scratch. Furthermore, while we Beniamini et al. solely focused on the names of the variables we repeat their study for procedures as well.

> Variable and procedure names in Scratch range mostly between 4 and 10 characters. For the single letter variables, the most commonly used names are $x$, $y$ and $i$, procedures—$a$ and $r$. When compared to the other programming languages, we observe that single-letter variable names are less common in Scratch and that overall Scratch variables have longer names. The usage of single uppercase letters is similar to the pattern found in Perl, for the lowercase—to the pattern found in Java.

Next we focus on **Scratch-specific features in naming identifiers**. In particular, we aim at understanding to what extent spaces within identifiers (e.g., variable *max i*), digits as identifiers (e.g., a variable named *6*) and textual labels used between the parameters. For example instead of printing the first $n$ letters from a string $s$ with a procedure called "printnof($n$,$s$)", in Scratch one can define a procedure called "say $n$ characters from text $s$", as shown in Figure 4. This feature exists in some textual languages too (most notable in SmallTalk) but is not common in most mainstream languages.

Investigating the use of these Scratch specific naming patterns is interesting to understand their role in novice programming. If they are popular among Scratch developers, this might be because they ease novice programming, and that means one could even advocate that these features should be integrated in the mainstream programming languages, if only to ease the transition from block-based languages into textual languages.

> Spaces in variable names are common: 20% of projects use this feature. Digits as identifiers are rarely used, and mostly represent constants or parts of the data structure. The usage of textual string between parameters appears as not so common, however textual patterns used imply an inference from textual languages, e.g., by using brackets.

## II. Background and motivation

Naming identifiers in software code has been studied extensively in the past decades [1], [16], [2], [3], [4], [17], [5], [18], [6], [7], [19], [8]. In practice, identifiers constitute a major part of the source code: e.g., Deißenböck and Pizka found that in Eclipse 3.0M7 which is tantamount to 2 MLoC, 33% of the tokens and 72% of characters correspond to identifiers [20]. **AS falling asleep, sorry** For a human to read that code, it is crucial to understand what the identifier means, and then can deduce what the code does. With no surprise, several studies have confirmed the link between good identifier naming and code's readability and comprehension. The comprehension is not a target in its own; the true reason is that better comprehension lets the developers perform maintenance tasks more effectively and efficiently. But what is a good naming approach? It is out of this paper's scope to explore the various recommendations of good identifier naming. However, it is worth to mention that there are different perspectives on what a good name is. Some researchers emphasize the usage of actual and complete words from a dictionary, or known abbreviations, which reflect the context of the program's purpose. Others argue that consistency in naming style is the most important. The usage of single-letter identifiers attracts much attention in research. For programmers, it is tempting to choose single-letter identifiers for quicker code writing that involves less mental load on the choice of a name. Additionally, single-letter named variables such as *"i"* or *"j"* have become almost a standard choice for index values when coding loops. Research has shown however that shorter identifier names are longer to comprehend, and the length of a variable should reflect its scope.

Despite the agreement on the importance of identifier names, and efforts to introduce naming guidelines and additional tools to help the programmer choose better names, developers find giving appropriate names to identifiers as a difficult task. In the end, it is the decision of the individual writing the code, and many factors may contribute to that decision. One area to consider here is the computer science and in particular programming education. There seems to be a great focus on the programming concepts, while less to zero attention is given to beautifying the code. For example, the usage of the identifier names *"foo"* and *"bar"* is prevalent in code examples. These names have no real meaning, and the student cannot link them to what the code does.

We are inspired by the work done by Beniamini et al. [3] where they explored the single-letter naming for multiple software repositories from different programming languages. It is important for the software community to understand the patterns in which software developers apply naming to identifiers. We argue that this understanding can improve the quality of the guidelines and introduce research-supported tools that meet the needs of the programmers. For this sake, we believe it is even more important to focus on the novice programmers, students, and learners. For these future developers, the usage of a particular naming pattern in this level

forms a (mis)conception that could move along with them to other programming languages and future careers. The work was done by **add ref** excludes the visual languages which have recently become a favorable choice for elementary and high schools as an introduction to programming, for example, Scratch and Alice. These block-based languages allow the user to create and assign their identifiers for variables and blocks. By studying these identifiers, we can understand better how novice programmers apply standard and language-specific naming, and how it compares to other textual based languages previously explored in the literature.

## III. Relevant Scratch Concepts

This paper is not an introduction into Scratch programming. Interested readers are referred to [21] for an extensive overview. However to be able to understand the remainder of this paper, we introduce a number of core features in this section.

Scratch is a block-based programming language aimed at children, developed by MIT. Scratch can be used to create games and interactive animations, and is available both as a stand-alone application and as a web application. Figure 2 shows the Scratch user interface in the Chrome browser.

### A. Sprites

Scratch code is organized into 'sprites': two-dimensional pictures that each have their own source code. Scratch allows users to bring their sprites to life in various ways, for example by moving them in the plane, having them say or think words or sentences via text balloons, but also by having them make sounds, grow, shrink and switch costumes.

The Scratch program in Figure 2[4] consists of one sprite, the cat, which is Scratch's default sprite and logo. The code in the sprite will cause the cat to jump up, say "hello", and come back down, when the green flag is clicked, and to make the 'meow' sound when the space bar is pressed.

### B. Scripts

Source code within sprites is organized in scripts: a script always starts with an event, followed by a number of blocks. The Scratch code in Figure 2 has two distinct scripts, one started by clicking on the green flag and one by pressing the space bar. It is possible for a single sprite to have multiple scripts initiated by the same event. In that case, all scripts will be executed simultaneously.

### C. Variables

Like most textual languages, Scratch users can use variables. Variables are untyped, but have to be 'declared' through the Scratch user interface, shown in Figure 3. This figure also shows that, contrary to most programming languages, variable names in Scratch may contain spaces.

[4]https://Scratch.mit.edu/projects/97086781/

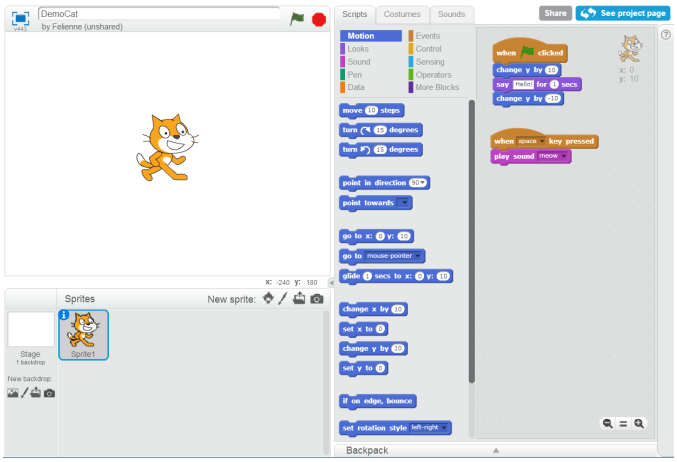


Fig. 2. The Scratch user interface consisting of the 'cat' sprite on the left, the toolbox with available blocks in the category 'motion' in the middle and the code associated with the sprite on the right.

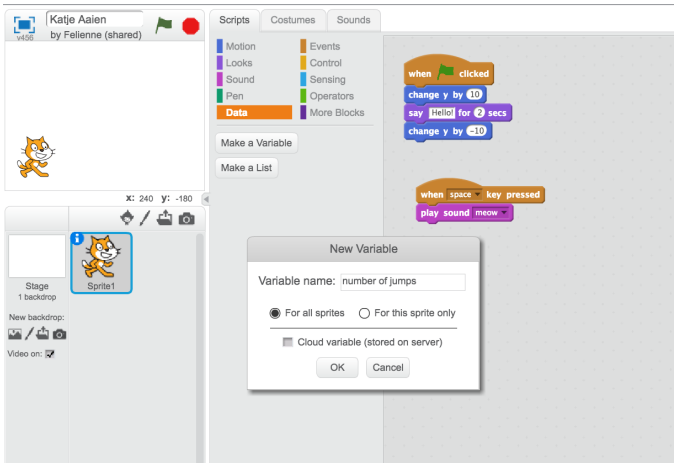


Fig. 3. The Scratch user interface to create a variable

### D. Procedures

Scratch also allows users to create their own blocks, called procedures. They can have input parameters, and labels in between them. Procedures are created with an interface similar to the one to create variables. Figure 4 shows the definition and invocation of a procedure.

## IV. Research design and dataset

### A. Overall design

As our goal is to compare the naming practices among the Scratch-developers with those of the developers in the mainstream programming languages we start by partially replicating the recent work of Beniamini et al. on the use of single-letter variables in Java, C, PHP, Perl and JavaScript [3]. In terms of the classification of Shull et al. [22] we perform a dependent replication of the studies summarized in Figures 1 and 2 of the original work by Beniamini et al. [3]. Inherently, the programming language is the only factor we vary when

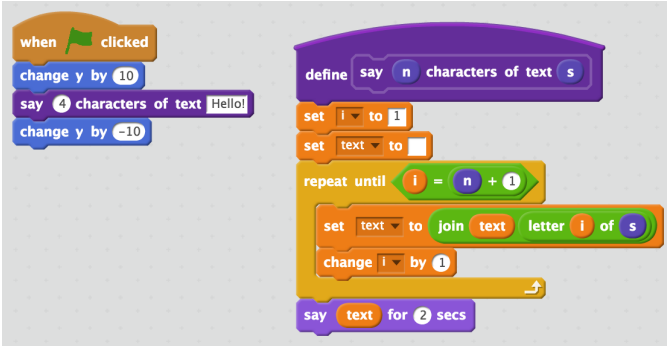


Fig. 4. Scratch code to define and invoke a procedure

compared to the original study. However, Scratch programs are not available on GitHub, so we gather those from the Scratch public repository. In this repository, Scratch users can share there programs, and just like on GitHub projects can be viewed and "remixed", which is, like a fork on GitHub, creating a linked copy of a project for further development. More specifically, we use the dataset previously scraped and processed by Aivaloglou and Hermans [15].

Next, we perform a conceptual replication of the study of the single-letter variable types of Beniamini et al. [3]. While the original study has conducted a survey to understand the type-related user perceptions ("what type would you consider for a variable called ...?") we focus on the types as defined in the program. Scratch is meant for people in their first steps of learning how to program, such as school-age students. In this early level, it is not expected they have established perceptions on data types of single-letters variables. In the replicated study however, 30% of survey respondents claim a 10-years experience in programming, while 23% have programming knowledge in six different languages or more.

Finally, to augment this study we also investigate the ways Scratch developers can benefit from Scratch-specific naming practices such as spaces in variable names, numeric values as variables and the use of textual labels in between parameters.

### B. Dataset

For this paper we use the dataset created by Aivaloglou and Hermans [15], consisting of 250.000 Scratch projects scraped from the Scratch website in March 2016. From this dataset, we have selected the projects that use variables or procedures, together this are 73.473 projects (29%) of Aivaloglou and Hermans's original dataset. Variable use is more common that procedure use. In total 69.045 projects (27.6%) use variables, while 17.605 use procedures (7.0%). We used Python to process the original dataset and generate the graphs in this paper. The code we used is available at https://github.com/Felienne/ScratchVars.

### C. Data analysis

To augment the visual comparison of the variable name data derived from Scratch as well as the programming languages considered by Beniamini et al. [3], we conduct statistical analysis.

Understanding differences in variable name lengths requires comparison of multiple distributions, traditionally performed as a two-step process consisting of (1) testing a global null hypothesis, that can be intuitively formulated as "all distributions are the same", using ANOVA or its non-parametric counterpart, the Kruskal-Wallis test, and (2) performing multiple pairwise comparisons of different distributions, testing specific subhypotheses such as "distributions 2 and 4 are the same". However, it has been observed that such a two-step approach can result in inconsistencies when either the global null hypothesis is rejected but none of the pairwise subhypotheses is rejected or vice versa [23]. Moreover, it has been suggested that the Wilcoxon-Mann-Whitney test, commonly used for subhypothesis testing, is not robust to unequal population variances, especially in the unequal sample size case [24] Therefore, one-step approaches have been sought. We opt for one such approach, the $\widetilde{\mathbf{T}}$-procedure of Konietschke et al. [25]. This procedure is robust against unequal population variances, respects transitivity, and has been successfully applied in empirical software engineering [26], [27], [28]. In particular, we use the Tukey (all-pairs) contrasts to compare all distributions pairwise.

Furthermore, to understand differences and similarities between the distributions of single-letter variable names in different languages we represent each programming language as a 26-dimensional vector with the dimensions corresponding to 'a', ..., 'z', and apply hierarchical clustering. **why hierarchical?**

## V. Results

This section presents an overview of our analysis of variable and procedure name use in our previously published Scratch dataset [15].

### A. Variable name length

The original study of Beniamini et al. [3] has concluded that the single-letter variable names "are approximately as common as other short lengths except in PHP" and that "in C, Java, and Perl they make up 920% of the names." Figure 5 shows the distribution of lengths in the Scratch corpus. A closer look at the data reveals that the single-letter variables constitute ca. 4.9% of all the names used in the program, i.e., less than the 9-20% observed by Beniamini et al.

Compared to mainstream languages, single-letter variables seem to be less common in Scratch, and the maximum length value for the variable names (250 characters) is the largest by a significant margin. These observations lead us to wonder whether overall the variable names in Scratch tend to be longer than in other programming languages. To this end we apply the $\widetilde{\mathbf{T}}$-procedure described in Section IV-C. Statistical analysis reveals that indeed, variable names in Scratch tend to be longer than in the mainstream languages. Moreover, variable names in Java tend to be longer than those in PHP, variable names in PHP than those in C, variable names in C than those in

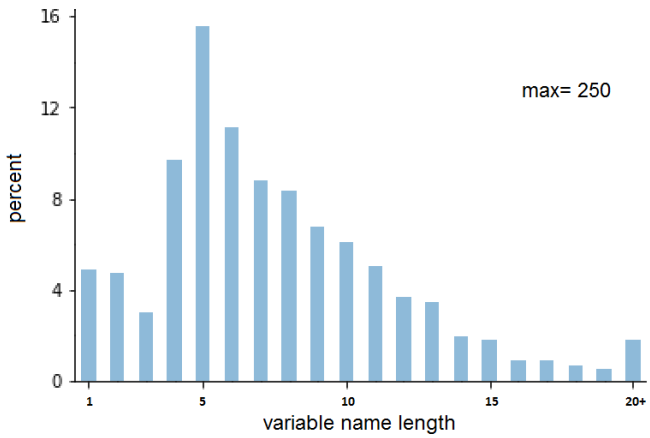


Fig. 5. The distribution of variable's name length in Scratch projects

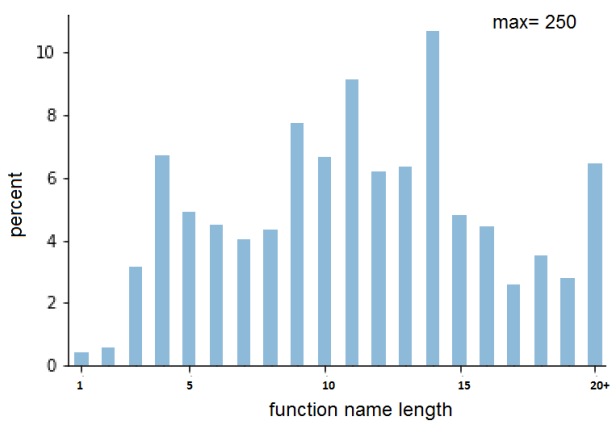


Fig. 6. The distribution of procedure's name length in Scratch projects

JavaScript and finally variable names in JavaScript longer than those in Perl. In all cases $p$-values have been too small to be computed precisely.

We conclude that single-letter variable names are less common in Scratch than in other programming languages and that overall Scratch variables have longer names than variables in other programming languages.

### B. Single-letter variable names

Further we investigate the case of single-letter variable names. For the previously studied programming languages, the authors in [3] highlight the following observations about the single-letter usage:

1) The most commonly occurring single-letter variable name is $i$.The authors attribute this to $i$ being commonly used as a loop counter. In Scratch loops are performed using predefined blocks, which is different than the studied textual language. As figure shows, the majority of these blocks do not require a variable to control the loop iterations. As a result, we expect the usage of the variable name $i$ to be less common in Scratch.

2) Apart from the popularity of $i$ the distribution is language-dependent. Since Scratch is quite different from the mainstream programming languages considered by Beniamini et al., we expect the distribution of the single-letter variable names to be different from the distributions in these languages. Hence, we expect the *similarity* between Scratch and the languages considered by Beniamini et al. to be lower than the *similarity* between the languages considered by Beniamini et al.

3) Finally, they observed that the lower case letters are used more frequently than the upper case letters. Since this is also the case for regular text in most natural languages as well, we expect the Scratch programs to follow the same pattern.


Fig. 7. Scratch blocks that are used to repeat specific actions

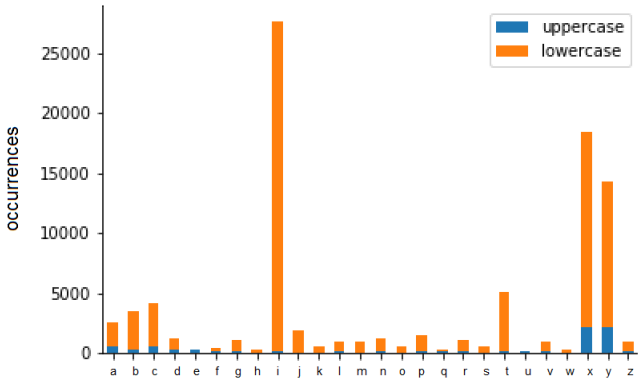


Fig. 8. A histogram of single-letter variables occurrences in Scratch projects

Figure 8 shows the distribution of variables of one letter, in upper and lower case, in the Scratch corpus. Inspecting the data we observe that similarly to the previous study $i$ is the most commonly occurring variable. Hence, we conclude that Observation 1 above also holds for Scratch. Furthermore, we observe that $x$ and $y$ are extremely popular in Scratch. This can be explained by noting that $x$ and $y$ represent the coordinates of the sprite object on the stage, which is the are where all the scripts are executed. Hence, they are the basis of moving the object in the 2-D stage. Scratch default blocks often use $x$ and $y$ as shown in Fig. 9. It seems that Scratch users are inspired by the Scratch language to name their own variables.

**From Alaaeddin: the following paragraph is the most ambiguous for me. It's is not clear to me if observation**

**2 and 3 are validated or negated. two analyses are done here to compare Scratch to other programming language: based on length distribution and based on single-letter patterns. upper and lower should be considered. I do not know exactly the outcome of these analyses**
**TODO-To be revised: Secondly, we observe that the distribution of single-letter variables in Scratch is indeed different than the studied languages by Beniamini *et. al*. [3]. The use of lower case is prevalent among the single-letter Scratch variables. We further analyze the similarity between Scratch and the mainstream languages for the single-letter pattern. We use the method as detailed in Section IV-C. Figure ?? shows the result of the analysis. It indicates that Scratch pattern of single-letter variables is highly similar to the pattern of single-letters in Perl.**

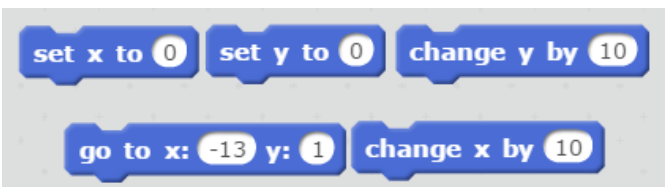


Fig. 9. Some Scratch blocks that use x and y characters

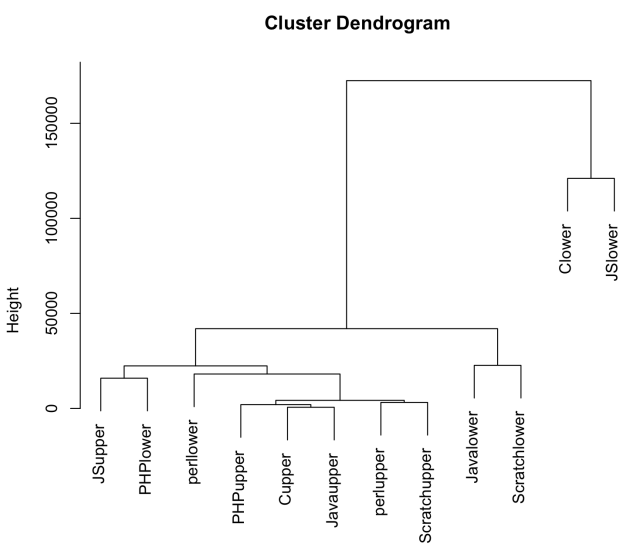


Fig. 10. A hierarchical diagram of Scratch compared to other programming languages for the single-letter pattern

***Types***: Beniamini et al. [3] also explored the types of one letter variables, by performing a questionnaire among experienced developers. One area of investigation in the survey is the data type which developers would associate to the alphabetic letters. They observe that some letters are highly associated by developers with the type which starts with these letters. For example char for $c$ and string for $s$. Integer data


Fig. 11. Two variables, one of type string and one of type integer

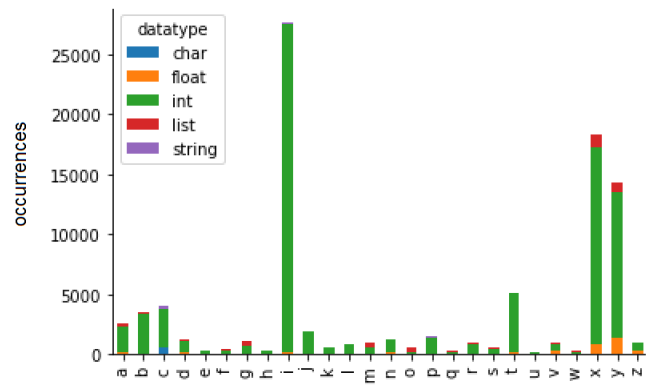


Fig. 12. Inferred types for variables of one letter

type is a common association for many other letters. and for generic letter names such as $x$, $y$ and $z$ there seems a balance between integer and float associations.

This lead to the idea to explore types of one letter variables in the context of our dataset too. While Scratch variables have no types, we can deduce their type from assignment statement which we also have in the dataset. For example, the two variables in Figure 11 represent a string and an integer respectively. With this process we can compare our results to the types of Beniamini et al. [3]. Figure 12 shows the distribution of variables of one letter in the corpus.We observe that the integer data type is massively common among single-letter variables in Scratch. One surprising observation is the nearly complete absence of string data type. For the observations of Beniamine et. al., we notice that the letter $c$ has a noticeable use of char data type which starts with 'c'. However it does not construct a majority and this association is not observed in other similar letters.

### C. Procedure names

Going beyond the study of Beniamini, we additionally consider the naming length of procedures in Scratch. For a detailed explanation of procedures in Scratch see Section III-D and Figure 4). Figure 6 shows the distribution of procedures name length in the Scratch dataset. By inspecting this figure we observe that the procedure names tend to be longer compared to Scratch variable names. Short names are not common, even less common than short names of variables, with single-letter names composing less than 1% of the extracted names. The maximum length for a procedure name is

250 characters, which is the same as the maximum length for the variable names. We suspect this exact match is caused by a language constraint that was imposed in a previous versions of Scratch. Current version, however, allows for names longer than 250 characters. In the same manner as the variable names, we investigate the distribution of single-letter names for the procedures. Figure 13 shows the number of occurrences for each alphabetic letter. From that figure, we observe that capital letters are more common compared to Scratch variable names, sometimes exceeding the lower letter's occurrences such as in the case of $r$. The top used single-letter name is $a$, the first letter in the alphabet, which may indicate its popularity.

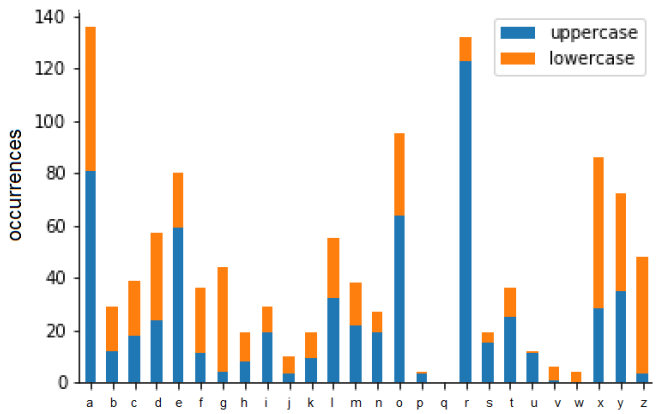


Fig. 13. A histogram of single-letter procedures occurrences in Scratch projects

### D. Scratch specific constructs

In this section we analyze the occurrence on naming practices that are allowed in Scratch, but not is most mainstream textual languages.

*1) Use of spaces in variable names:* For reasons of convenience, most textual programming languages so not allow for spaces in variable names. FORTRAN ignored spaces, so technically you could use a space, however that would mean that 'apples' and 'app les' would refer to the same variable. Apart from SQL and some Scheme implementations, spaces in variable names are not commonly allowed. Scratch does allow users to use spaces in variable names and their use is quite common even. About 30.000 projects use one or more variables with a space in it, versus 60.000 that use only space-free variable names. Figure 14 shows the distribution of spaces in variable names. We have found that many introductory Scratch programming materials **reference** demonstrate the use of space-free variables, and that children—and adults—that already have programming experience deem the use of spaces in variables as non-natural **reference** , even though arguable 'number of apples' is more natural that 'nApples'.

*2) Use of Numeric Variable Names:* In addition to spaces in variable names, Scratch even allows the use of numbers and even floating point numbers as variables. We found 718

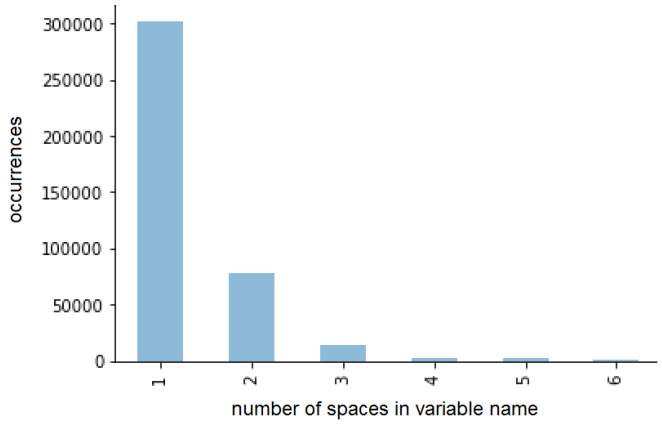


Fig. 14. Number of spaces in variable names

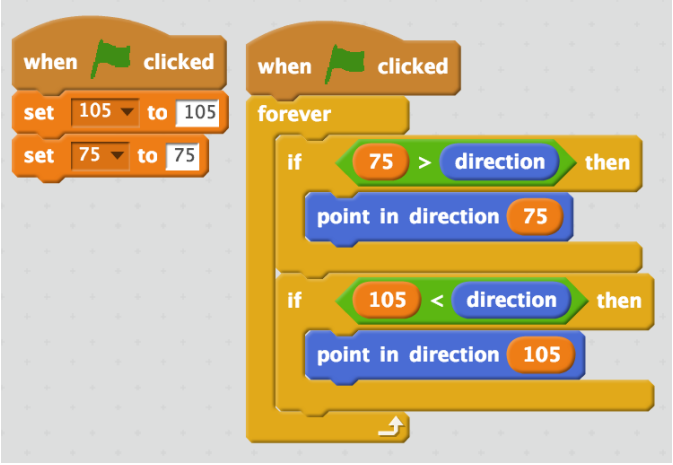


Fig. 15. Numeric variable used as a constant

projects with integer variable names and 19 with floating point names. While their use is rare, we manually examined some projects and numbers are used in interesting and clever ways.

There seem to be two main uses of numeric variable names. The first one is constants, as shown in Figure 15. This seems to indicate the Scratch users prefer dragging a contant in, over repeatedly typing the value in.

A second use is the use of integer variables as simple list structures. For example, one of the projects we analyzed is a tic-tac-toe game. In that project, the Scratch user defined nine variables named from 1 to 9. Each variable represents one of the nine boxes. Scratch supports lists, so the user here could have also used a list of 9 items, however, they did not. Maybe because they were nor familiar with the procedure of lists, or maybe they thought this would be easier for the user to memorize the game logic.

**Maybe add the screenshot (thanks Alaaeedin) Not sure if we need it, lets see**

*3) Use of Textual Labels between Parameters:* Scratch is influenced by the SmallTalk family of languages and this is

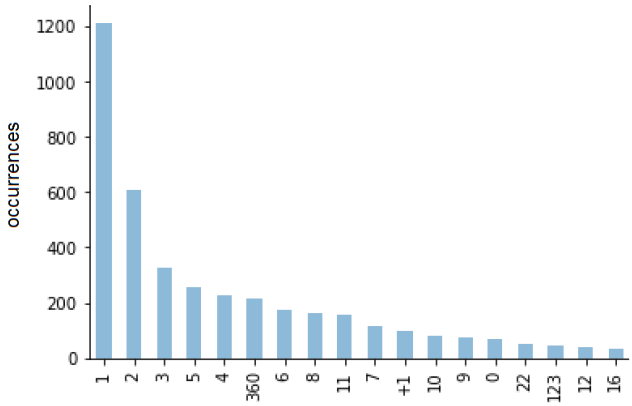


Fig. 16. The most popular numeric values used as variable names

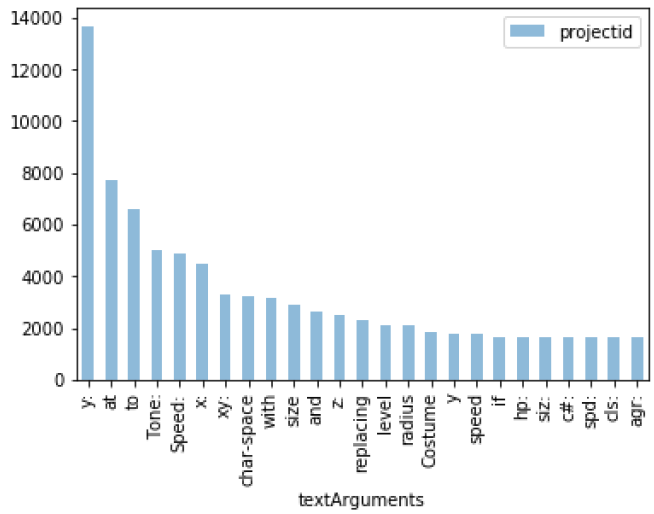


Fig. 17. The most used textual labels in between parameters of procedures

visible form the fact that Scratch allows users to insert textual labels in between parameters in order to make procedures more readable, as can be seen in Figure 4.

This practice seems particular idiomatic to Scratch, since normal Scratch blocks use a similar syntax, for example in the "say ... for ... seconds" block.

In total 4,415 projects use textual labels, so their use is not that common. We do however find some interesting patterns. Figure 17 shows the most commonly used labels. Here we see some patterns common in textual languages, like the use of labels for the names of the parameters 'x:' and 'y:'. Furthermore we see the use of ':' at the end of many patterns, which could come from the users being inspired by Scratch default blocks, which use the colon as shown in Figure 9. Finally the use of the space (char-space in Figure 17) is interesting, since Scratch already leaves some room between the parameters, also when a space is not used. The use of space as a separator could indicate that Scratch users feel room between variables is currently too small.

## VI. Discussion

### A. Threats to validity

Representation of the sample Statistical analysis chosen

### B. Naming in Scratch

We only considered features of Scratch that enforce the user to input textual values. For example the name of the sprites can also be considered as an identifier. However, Scratch assigns default names to sprites once they are created, and the user may opt not to change it. Default names can be considered as a bad naming practice, but it is out of the scope of this paper.
**other ideas for the discussion: for example:**
**what effects would the outcome of this analysis have on Scratch-related material**
**or maybe we skip the discussion?**

## VII. Conclusion

In this paper, we study naming patterns for variables and functions in the Scratch programming language, a block-based programming language aimed at novice programmers. We use a previously released dataset consisting of 250.000 Scratch programs.

Our analysis shows that Scratch users most often use variable and procedure names between 4 and 10 characters in length. For the single letter variables, the most commonly used names are x, y and i. Spaces in variable names, a feature relatively unique to Scratch are used in 20% of projects have variables that include spaces in the name. The usage of textual string between parameters appears as not so common, however textual patterns used imply an inference from textual languages by using brackets for example. Finally, when compared to the other programming languages, Scratch variable length distribution, and the usage of single-letter seems to be most similar to Perl.

The paper makes the following contributions:

- A detailed analysis of one letter variable names, replicating [3] on the Scratch programming language
- An analysis of function names in Scratch
- An analysis of naming patterns unique to Scratch, including spaces in variable names, textual lables in procedures and numeric variable names

This paper gives rise to a number of directions for future work. Firstly, Beniamini et al. [3] included a survey in which they ask developers to predict the type of a (one letter) variable. It could be interesting to ask a similar question of children to for common variable names. Furthermore, a detailed study into the readability of variable names with and without spaces, and procedures with and without labels would help us to create naming guidelines for Scratch.

## REFERENCES


[1] H. Aman, S. Amasaki, T. Sasaki, and M. Kawahara, "Empirical analysis of change-proneness in methods having local variables with long names and comments," in *ESEM*. IEEE, 2015, pp. 50–53.

[2] E. Avidan and D. G. Feitelson, "Effects of variable names on comprehension an empirical study," in *ICPC*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 55–65.

[3] G. Beniamini, S. Gingichashvili, A. Klein-Orbach, and D. G. Feitelson, "Meaningful identifier names: the case of single-letter variables," in *ICPC*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 45–54.

[4] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *CSMR*, R. Capilla, R. Ferenc, and J. C. Dueñas, Eds. IEEE, 2010.

[5] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *SANER*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE, 2017, pp. 217–227.

[6] M. Lungu and J. Kurs, "On planning an evaluation of the impact of identifier names on the readability and quality of smalltalk programs," in *Workshop on User Evaluations for Software Engineering Researchers*. IEEE, 2013, pp. 13–15.

[7] G. Scanniello and M. Risi, "Dealing with faults in source code: Abbreviated vs. full-word identifier names," in *ICSM*. IEEE, 2013, pp. 190–199.

[8] P. Tramontana, M. Risi, and G. Scanniello, "Studying abbreviated vs. full-word identifier names when dealing with faults: an external replication," in *ESEM*, M. Morisio, T. Dybå, and M. Torchiano, Eds. ACM, 2014, p. 64:1.

[9] T. Kato, Y. Kambayashi, and Y. Kodama, *Data Mining of Students' Behaviors in Programming Exercises*. Cham: Springer, 2016, pp. 121–133.

[10] K. Rother, *Cleaning Up Code*. Springer, 2017, pp. 195–212.

[11] E. S. Raymond, *The New Hacker's Dictionary*. MIT Press, 1996. [Online]. Available: https://books.google.be/books?id=g80P_4v4QbIC

[12] J.-M. Sáez-López, M. Román-González, and E. Vázquez-Cano, "Visual programming languages integrated across the curriculum in elementary school: A two year case study using scratch in five schools," *Computers & Education*, vol. 97, pp. 129–141, 2016.

[13] B. M. Hill and A. Monroy-Hernández, "The remixing dilemma," *American Behavioral Scientist*, vol. 57, no. 5, pp. 643–663, 2013.

[14] R. Davis, Y. Kafai, V. Vasudevan, and E. Lee, "The education arcade: Crafting, remixing, and playing with controllers for scratch games," in *International Conference on Interaction Design and Children*. ACM, 2013, pp. 439–442.

[15] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the scratch repository," in *ICER*, 2016.

[16] N. Anquetil and T. C. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *CASCON*, S. A. MacKay and J. H. Johnson, Eds. IBM, 1998, p. 4.

[17] B. Caprile and P. Tonella, "Restructuring program identifier names," in *ICSM*. IEEE, 2000, pp. 97–107.

[18] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "Effective identifier names for comprehension and memory," *ISSE*, vol. 3, no. 4, pp. 303–318, 2007.

[19] A. A. Takang, P. A. Grubb, and R. D. Macredie, "The effects of comments and identifier names on program comprehensibility: an experimental investigation," *J. Prog. Lang.*, vol. 4, no. 3, pp. 143–167, 1996.

[20] F. Deißenböck and M. Pizka, "Concise and consistent naming [software system identifier naming]," in *IWPC*, May 2005, pp. 97–106.

[21] K. Brennan, C. Balch, and M. Chung, *CREATIVE COMPUTING*. Harvard Graduate School of Education, 2014.

[22] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Software Engineering*, vol. 13, no. 2, pp. 211–218, 2008.

[23] K. R. Gabriel, "Simultaneous test procedures—some theory of multiple comparisons," *The Annals Mathematical Statistics*, vol. 40, no. 1, pp. 224–250, 1969.

[24] D. W. Zimmerman and B. D. Zumbo, "Parametric alternatives to the Student t test under violation of normality and homogeneity of variance," *Perceptual and Motor Skills*, vol. 74, no. 3(1), pp. 835–844, 1992.

[25] F. Konietschke, L. A. Hothorn, and E. Brunner, "Rank-based multiple test procedures and simultaneous confidence intervals," *Electronic Journal of Statistics*, vol. 6, pp. 738–759, 2012.

[26] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, "On the variation and specialisation of workload—A case study of the gnome ecosystem community," *Empirical Software Engineering*, vol. 19, no. 4, pp. 955–1008, 2014.

[27] B. Vasilescu, A. Capiluppi, and A. Serebrenik, "Gender, representation and online participation: A quantitative study," *Interacting with Computers*, vol. 26, no. 5, pp. 488–511, 2014.

[28] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information & Software Technology*, vol. 74, pp. 204–218, 2016.

[29] G. Scanniello, D. Lo, and A. Serebrenik, Eds., *ICPC*. IEEE / ACM, 2017.