Advantages of utilizing Scratch over Python as a programming language to instruct K-12 students

Montana Merkle
Department of Computer Science & Information Systems
The University of North Alabama
Florence, Alabama, United States
mmerkle@una.edu

1. INTRODUCTION

In today's world, children are exposed to technology incredibly early and most utilize mobile devices daily. It is no surprise that children who are proficient with technology are better suited for the future workplace since they can understand and utilize basic devices and applications. Computer programming is a skill most kids are not exposed to until later in their academic careers, but this is not in their best interest. Programming helps children learn logic, inspires creativity, and can improve a child's understanding of how the devices they constantly use can work! The question now is what the best way is to introduce this complicated topic to children and what programming language is best suited to teach while keeping young students engaged. Popular coding languages recommended for kids include Python, Java, Ruby, and Scratch [1]. This paper focuses on the advantages of utilizing Scratch as a programming language to teach coding to students in K-12 and how it compares to another popular language such as Python in this circumstance.

2. BACKGROUND

2.1 Historical Background

2.1.1 Scratch's Launch and Purpose

Scratch was officially launched in May of 2007 with the main goal of creating a language that abstracted away syntax and punctuation errors to glean better accessibility for their target audience of inexperienced programmers [2]. According to an interview with one of the creators of Scratch, Mitchel Resnick, the language was developed for children aged 8 and up when the group at MIT felt that there was a need for a new type of programming language designed specifically with kids in mind [3]. This programming language was developed by the Lifelong Kindergarten research group at the MIT Media lab that included Mitchel Resnick, Natalie Rusk, Brian Silverman, and John Maloney [4].

2.1.2 Influences of Scratch

Resnick was heavily influenced by the Logo programming language and one of Logo's creators, Seymour Papert when developing Scratch [5]. Logo, developed in the 1970s, is a dialect of Lisp and most notably introduced the idea of the turtle [6]. Users of Logo could access an environment that involved the turtle, which is a graphics object depicted from a birds-eye view able to receive commands such as *forward* and *right*, the turtle is also able to utilize a pen tool to draw a line following where the object moves [7]. This feature of Logo has been implemented with variations in Scratch depicted in Figure two, with the graphic object being called a sprite with similar abilities and the same visual appeal to users as the turtle seen in Figure one [4].

Figure 1: Implementation of the turtle in Logo [7]**.**

Figure 2: Depiction of the cat sprite in Scratch [4]**.**

Even after adopting the turtle, Resnick still had more ideas to evolve Logo into a simpler language easy to teach young children. His main worry with Logo involved its syntax and punctuation, which was not incredibly difficult to master, but would still require learners to understand and implement while coding [4]. The answer to this problem came in the form of LogoBlocks, which was developed in 1996 at the MIT Media Lab [8]. This form of Logo was one of the first block-programming languages and allowed users to drag and drop set blocks of instructions into a window and then run the instructions without the hassle of continuous syntax errors due to missing punctuation, like a semicolon [4]. This style of programming abstracted syntax and punctuation away from users, allowing kids to focus on visualization instead of text-based code. With the creation of this new style of Logo, Scratch found its missing piece and after general testing with children and educators, the development team decided to utilize this style moving forward in their language as well [4].

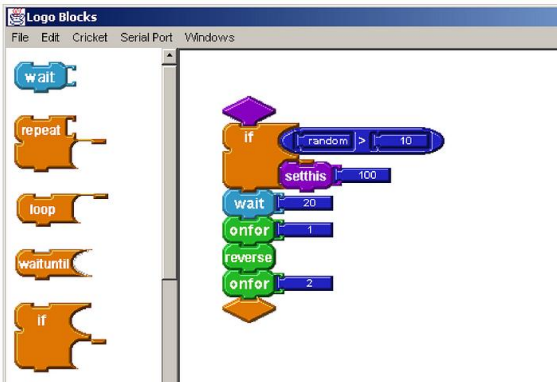

Figure 3: Early LogoBlocks code [4].

2.1.3 Scratch's Integrated Online Community

Another important feature decided upon in the early days of Scratch in 2005 was to allow users of the language to share projects online with other people on the website [4]. With this added feature, "Scratch became the first programming language with its own integrated online community," which was a major part of how Scratch was able to distance itself from other languages at the time [4]. Flickr, a website dedicated to online photo-sharing, had just been launched and was the main influence behind this idea [4]. Collaboration between students sharing projects online could help kids to be more creative by allowing users to view others' work and build upon it. This new feature on top of the utilization of music, images, and color, inspired by another graphical programming language, Squeak Etoys, developed by a group at Disney, led Resnick to believe Scratch would empower users to engage and create a wide range of projects that could be updated in real-time [4].

2.1.4 Impact of the Language

Since Scratch's release in 2007, "annual activity on the Scratch website has grown by a factor of 250, an average growth rate of 45% per year" and since 2022, "more than 30 million people around the world programmed and shared stories, games, and animations with Scratch" [4]. Figure four depicts this growth in active users of Scratch spanning from the initial release of the language in 2007 through 2023 [9]. According to the Scratch Foundation, an independent nonprofit organization that supports Scratch, the most common age range of new Scratch users is between 10 and 14 years old, as seen in Figure five [9]. With the release of Scratch 2.0 in 2013, the language was available worldwide with an online interface and helped solidify the language's position as one of the top coding communities for young students [10]. Aspects of Scratch that allow for global interaction spanning 196 countries include that it is accessible for free, available worldwide, and has 70 plus language translations handy [10].

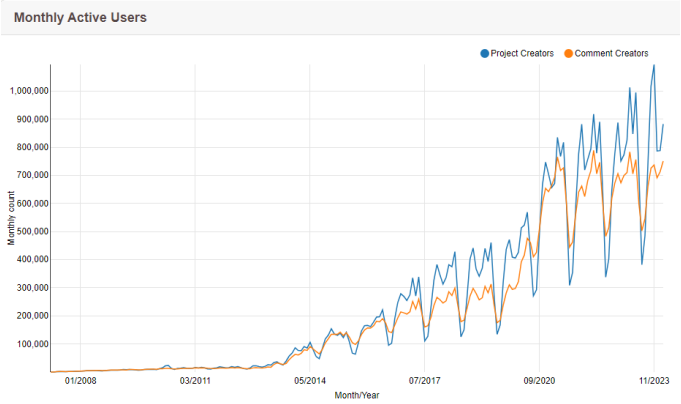

Figure 4: Monthly active users of Scratch spanning from 2008 through 2023 [9].

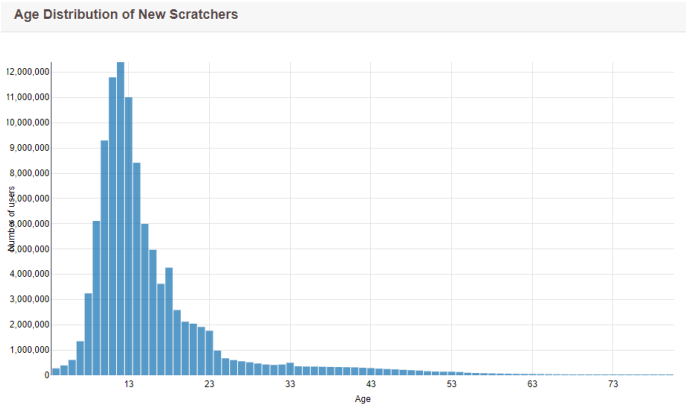

Figure 5: The age distribution of new Scratch users [9].

2.2 Language Overview

2.2.1 Scratch Basics

As mentioned in the historical background, Scratch is a block-based programming language. This means that unlike many text-based programming languages, such as Python, users of the language utilize pre-defined blocks to create a program, called a script in Scratch, instead of typing out instructions. This comparison is seen in Figure six.

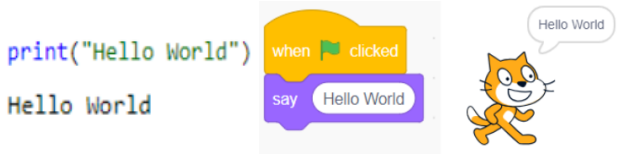

Figure 6: Python (left) compared to Scratch's (right) implementation of the "Hello World" program.

Command blocks are snapped together to control the actions of the graphic objects, called sprites, displayed on an adjacent area to the right of the block area called the stage [11]. Through the utilization

of blocks, Scratch code can be understood much easier by inexperienced programmers and gives a visual representation of the flow of the script as it executes by highlighting the block currently being executed. While some may overlook the power of Scratch due to its simple nature, the language can produce a variety of complex programs such as animations, interactive art, simulations, and interactive stories [4].

2.2.2 User Interface of Scratch

Scratch's user interface is displayed in Figure seven and contains the block palette (far left), script area (middle), stage area (top right), sprite information pane (bottom right), and the costume and sound panes (tabs to the right of the code tab) [12].

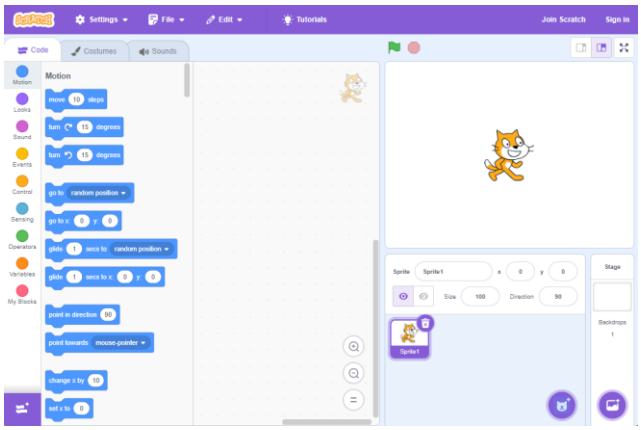

Figure 7: Scratch's user interface.

The block palette is where users of the language can drag and drop preset code blocks into the script area to formulate instructions for the sprite to execute. Each block shape is different based on what the block can do, for example, the *repeat* block has a C shape that allows many blocks to fit into it, and thus be repeated [8]. Blocks are also separated into categories based on how they interact with the environment and the sprite. Once many blocks are snapped together vertically, they are called a script [8].

The script area is where code can be assembled to create a program in Scratch, scripts can be run by either clicking on a stack of them or by clicking the green flag while starting a block stack with the block *when green flag clicked* and stopped by clicking the red hexagon next to the flag. Many stacks of code blocks can be present in the script area but will not run unless the segment contains an event type block (most begin with the word when) or is clicked on by the user. This feature allows programmers to tinker with multiple segments of code with ease without having to create new projects to test the output of a new set of blocks. Code blocks can be deleted by either right-clicking the block and selecting *delete block* or by dragging it back into the block palette to the left of the script area. Another innovative aspect of the script area is that in the top right corner, a faded image of the current sprite selected is displayed, as seen in Figure eight. This is helpful since each sprite has its own unique script area and this feature reminds the user of the current one selected to manipulate.

The stage area visually displays the effects of the instructions executed in the script area and contains the sprite cartoon character/characters chosen by the user. This area of the user interface has many possible backgrounds to choose from and can be highly customized by the user. As code blocks are executed, the stage area will update in real-time based on the instructions, giving the inexperienced programmer a visual look into how each instruction changes the script's output.

Another feature of the stage area is that it can display the values of any created variable in a project as seen in Figure eight. The example script found in Figure eight will increment the variable *Count* by 1, wait one second, add the new value of the variable to the list, and repeat 10 times after the green flag is clicked to start the execution of the instructions. The two code blocks to the left of the larger segment of code instruct *Count* to be set back to zero and the list to be deleted. As the script is executed, the variable *Count* visibly changes values in the stage area as it is incrementing, the list gets updated with each new *Count* value, and the length of the list (found at the bottom of the list in the stage area) increments as well. Scratch's ability to display and update any list or variable that the user creates as the program is executed can be a great tool to help users understand how a variable or list works and how each step of the instructions manipulates the output.

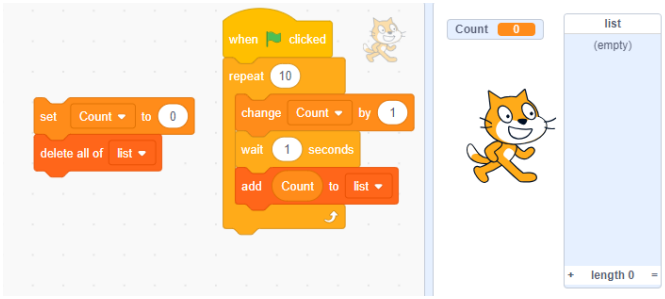

Figure 8: Code from Scratch demonstrating the ability to display the value associated with a variable and the content and length of a list.

The sprite information pane gives an in-depth look into what graphic objects the user has selected to include in the stage area. This area of the user interface in Scratch allows programmers to create, delete, name, set the visibility, set the direction of movement, and manipulate the size or location of the sprite. It is important to note that while a user's code might not give the graphic object any instructions to change its behavior, a sprite still needs to be present for any code blocks to be executed.

The costume pane of the Scratch interface grants the user even more customization while designing how they want their sprite to act during a script, adding another layer of creativity to the programming language. The costume tab allows the programmer to manipulate the look of the sprite, including features such as

painting, reshaping, and adding text to the sprite. What distinguishes this interface from the sprite information pane is that while the sprite information pane contains basic information, the costumes pane is much more attentive to the details of the individual sprite. Each sprite automatically comes with preset costumes, which are slightly varied versions of the sprite that can be swapped on and off the graphic object. This swapping is done through a specific code block classified as a looks instruction named *switch costume to* with a drop-down menu displaying the possible costumes of the sprite. This is incredibly user-friendly and allows the programmer to animate the object without having to design the changes themselves, however, that is a possibility for those who choose to pursue it. For example, the basic cat sprite in Scratch has a default second costume, which if swapped makes it look like the cat is running as seen in Figure nine.

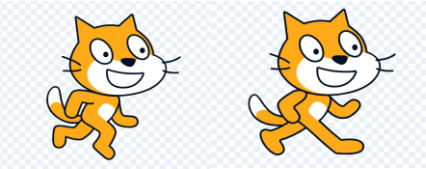

Figure 9: The cat sprite's default costumes in Scratch.

The sound pane implements another feature of Scratch, sound. Sound can be added and manipulated by a script through code blocks classified as sound instructions. The sound pane grants users the ability to add sounds from the provided selection of sounds on Scratch, speed up the sound, and change the volume or pitch of the sound selected.

2.2.3 Scratch's Programming Capabilities

Scratch can arguably fit into many different programming paradigms; however, most consider the language to be a visual block-based imperative procedural object-based programming language. This is because code is written as a list of detailed instructions to the computer (in the form of a collection of blocks), the language gives the user the ability to make and implement functions (in the form of custom blocks), and the language contains objects that can change states and behavior (in the form of sprites) [11]. However, since there is no inheritance or classes the language is not defined as object-oriented, only object-based [11]. On top of this, Scratch is an interpreted language, meaning that it is not compiled, and instructions can be changed during execution [13].

2.2.4 Syntax in Scratch

Since Scratch is a block-based programming language, the shape of a block and the rules associated with blocks of those shape determine how they can be snapped together, thus this is the equivalent of syntax in text-based programming languages [11]. Scratch automatically prevents users from snapping together code blocks that are not grammatically correct in the language. Thus, all blocks are considered *failsoft*, meaning that code blocks are designed to "do something sensible" even though input data might be out of range [11]. This feature of Scratch is a massive contributor to the abstraction of syntax errors and punctuation, and the reason why the language works even though it has no way to have a script fail with an error message [11]. Although a user will not be constantly plagued with error messages while programming, this does not mean that a script cannot be written with errors. Users can still create and execute segments of code blocks that have unintended results.

2.2.5 Types of Code Blocks in Scratch

Scratch has nine major sets of code blocks available separated by color and category based on what they're able to do.

The first category is motion (blue) which includes blocks like *point in direction 90* and *move 10 steps* where the 10 and 90 can be changed with user input. This category contains blocks that can manipulate the location of the sprite in the stage area and utilize values of x and y to set coordinates of movement.

The next category is looks (violet) which includes blocks like *next costume* and *say hello* where hello can be changed with user input. Blocks in this category are used to manipulate what the sprite says, thinks, and what costume the current sprite has on.

Sound is the following category of blocks (magenta) which has blocks like *start sound Meow* and *stop all sounds* where meow can be replaced with another sound the user chooses. This category is utilized with the sound pane to incorporate noise during the execution of a script.

The fourth category is the events group (yellow) which includes blocks like *when this sprite clicked* and *when green flag clicked*. These code blocks are utilized to execute a block of code based on outside actions by the user.

The next grouping of code blocks is the control group (orange) which includes blocks like *wait*, *if/then*, and *repeat 10* where 10 can be replaced with user input to represent times the block should repeat. This category is the block-based equivalent of control structures in text-based languages. Scratch allows for the repetition of code segments with *forever*, *repeat*, and *repeat until* looping code blocks shown in Figure 10. The code block *repeat until* allows for the user to insert a boolean variable into the slot represented by an empty hexagon (the boolean variable shape in Scratch). Conditional statements are also supported in the form of *if/then*, *if/then/else*, and *wait until* code blocks shown in Figure 10. These blocks allow the user to determine the path of execution based on the values of the boolean variables inserted.

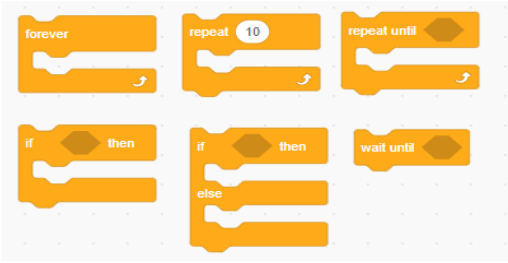

Figure 10: Loops (first row) and conditional blocks (second row) in Scratch.

The sixth category of code blocks provided by Scratch is the sensing group (light blue) which includes blocks such as *touching color* and *key space pressed*, which are set boolean variables in Scratch. This category is used to provide tools that can understand how the environment of the stage area is changing and is useful when creating animations.

The next category is the operators category (green) which contains blocks with logical operators, such as *not* and *or*. This category also contains mathematical operations like addition and modulus. Operator blocks are useful to create custom boolean variables, calculate math problems, and do basic string manipulation with blocks such as *length of apple* and *letter 1 of apple* that allow the user to change apple for whatever string they want.

The last two categories, variables (dark orange) and my blocks (pink) allow the user to create variables, lists, and custom blocks for functions. This grouping also provides set code blocks to manipulate variables, lists, and to call the custom code block as well. Examples of these code blocks include *set my variable to 0* and *add thing to list*, where zero and thing can be changed by the user. These preset blocks make it easy for a user to manipulate how a custom block behaves during the script.

2.2.6 Semantics in Scratch

Figure 11 shows the basic use of the control block *forever* to iterate through the code block segment indefinitely. After the green flag is clicked, the cat sprite will move 10 steps to the right, say hello for two seconds, and then increment the variable named my variable. After six iterations of the loop, the result is shown to the right of the block segment with the variable now set to six.

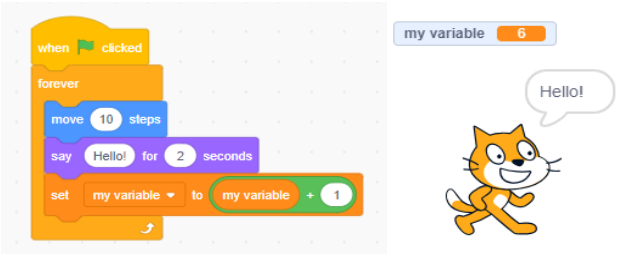

Figure 11: Code segment from Scratch using the *forever* control block.

Figure 12 exhibits a segment of blocks in Scratch utilizing the *if/then* control block and the movement block *turn 15 degrees*. When the space key is pressed, the script begins to execute and first sets the variable named my variable to apple. Since the value in the variable is equivalent to apple, the sound code block is executed, and then the cat sprite is turned 15 degrees clockwise. Every time the user hits the space key, the script is executed again.

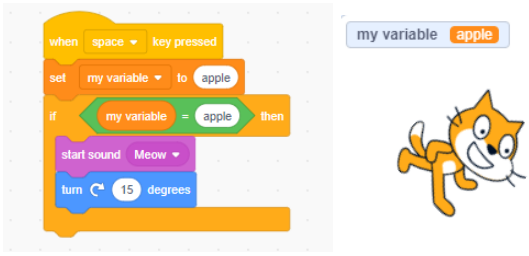

Figure 12: Code segment from Scratch using the *if/then* control block, a sound block, and a movement block.

2.2.7 Naming in Scratch

Scratch has a unique set of rules when naming functions. Unlike many other languages, Scratch allows a combination of characters to name functions or variables that include spaces, numbers, and even floating-point numbers according to Swidan et al. [14]. There does not seem to be a character limit on the names of functions/variables, as a variable with 926 characters was able to be utilized. Since Scratch utilizes set code blocks, there is no stress placed on the restriction of names to prevent variables/functions from having the same name as a keyword, thus variables/functions can be named basically whatever the user chooses.

2.2.8 Variable Typing in Scratch

In Scratch, there are only three data types: number, string, and boolean [11]. Each data type has its own shape of block that helps the user determine what type it is, a hexagon for boolean and a circle for number and string. When a user creates their variable, the data input is untyped meaning that the language does not require the user to explicitly state what type of data the variable will contain [11]. Thus, because of this fact Scratch is a dynamically typed language, which is common for interpreted languages. With untyped variables, the responsibility to determine the type of data bound to a variable is handled by Scratch at runtime and abstracted away from the user. The language does this through "automatic conversion" of data based on the input [11]. For example, if a variable is set to the string *four* and then utilized in an operator block that adds two variables, the value of the operator block will be zero since the variable containing *four* is automatically typed by the system as a string type. Boolean variables can only be created using the set code blocks seen in Figure 13 or utilizing these blocks with the *set variable to* block.

Figure 13: Collection of boolean variables provided in Scratch.

2.2.9 Scope and Binding of Variables in Scratch

The scope of a variable in Scratch is determined by user selection when first defining the variable. As seen in Figure 14, the two options during variable creation are *for all sprites* or *for this sprite only*. This is the Scratch equivalent of global and local variables in a text-based language. Variables set to all sprites can be accessed by all sprites and thus a global variable, while variables set to one sprite are local meaning that other sprites will not have access to it. Variables in scratch are dynamically bound, therefore the value is determined at run-time. A specific feature that Scratch implements with its target audience of young students in mind is the ability to change the value of a variable while a script is being executed. The user can do this by simply changing the *set variable to* block as the segment is being executed. This feature allows inexperienced programmers to see the value update in real-time.

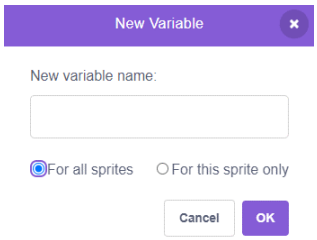

Figure 14: Interface to create a new variable in Scratch.

2.2.10 Functions in Scratch

Custom blocks in Scratch allow the user to input a label and add inputs that are numbers, text, or booleans as seen in Figure 15. In more general terms not specific to Scratch, the language allows the user to define a function with a specific name and pass parameters into said function. Each parameter has a specific shape associated with the type of parameter allowed (hexagon for boolean and circle for numbers and strings), which prevents the user from inserting the wrong type of parameter into a block resulting in a syntax error [11]. After a custom block is created, two preset code blocks are dropped into the script area as seen in the bottom half of Figure 15.

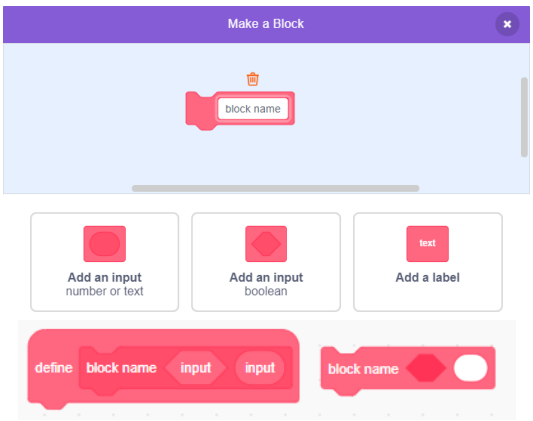

Figure 15: Interface to create a custom block in Scratch and the preset blocks created as a result.

2.2.11 Concurrency in Scratch

Concurrency in most programming languages requires the careful use of locks or semaphores to prevent race conditions between concurrent processes, but in Scratch, this happens a little differently. Like other situations that cause errors due to inexperienced coders, Scratch prevents the possibility of a race condition happening by abstracting the controls away from the user and only allowing thread switches to occur when a command waits explicitly or after a loop [11]. But again, just because the language does not allow race conditions does not mean there are no problems with concurrency, like when an event block (such as *when*) triggers many different code segments in some order [11]. According to Maloney et al. [11], the creators of Scratch, the manipulation of synchronization controls by users is rarely used. This makes sense since concurrency is a higher-level programming concept that most novices would not be exposed to during experimentation with Scratch.

2.3 Literature Review

2.3.1 Effects on Motivation, Self-Efficacy, and Engagement After Scratch

An interesting study was conducted by Campbell and Atagana with first-year computer science college students in North Central Nigeria utilizing Scratch [15]. Before explaining the study's result, the authors reference multiple accredited sources to make the point that introductory CS1 classes fail to engage students and have a high failure rate, leading most interested students to choose another path of education [15]. This is a major problem for the future, as computer scientists are in high demand and utilized by companies for website design, security, database management, and much more. Campbell and Atagana suggest that higher levels of engagement could lead to a reduced dropout rate as seen in the study conducted on incarcerated youth in 2020, as well as motivate students to learn in general [16] [15].

Not only does engagement pertain to introductory students in a college setting, but also adults first learning programming as well. Krafft, Fraser, and Walkinshaw conducted a study on novice adult programmers that revealed the difference in levels of motivation and engagement when Scratch is implemented before Python versus just Python [17]. Both the study conducted on adults and first-year college students found that the implementation of Scratch resulted in a high positive impact on the motivation and engagement of the novices, with college students having "an overall engagement score of 77.4%" and the adults exposed to Scratch before Python "significantly improved their motivation whereas the non-Scratch users showed no significant change" [15] [17]. While both studies relied on qualitative data for the most part in the form of surveys after classes, the results suggest Scratch can be used as a formidable tool to keep introductory students engaged in programming and continue to learn the subject. On top of this, even though the study conducted by Krafft, Fraser, and Walkinshaw indicated no difference in students' performance when Scratch was introduced before Python, the study, in this case, supported the hypothesis that Scratch is better suited to motivate

introductory students compared to Python, at least when adults are involved [17]. As adults and children learn differently, a study conducted on a younger test group would offer more beneficial information to answer the overall research question of the advantages of Scratch over Python for K-12 students.

Christina Zdawczyk and Keisha Varma organized a study in 2023 that compared Python and Scratch directly, much like Krafft and other contributors, but with a younger test group of elementary to middle schoolers [18] [17]. A strength of Zdawczyk and Varma's research was their unique goal to analyze how the different programming languages (Scratch and Python) attracted novices based on gender, as well as assessing the contrasting levels of self-efficacy and interest in computer science after the language is utilized [18]. With groupings of students exposed to three different conditions (No Visual, Python, and Scratch) of programming environments, both girls and boys who participated in the study had "significantly higher self-efficacy in Scratch than the other two conditions, indicating that both genders may have a greater belief in their ability to successfully learn computer programming through Scratch" [18]. These results imply that Scratch is regarded to be easier to learn than Python and agree with the studies conducted in Nigeria and with adult subjects of Scratch's positive effects on student motivation and engagement [18] [15] [17]. On top of this, according to the data analyzed by gender, girls demonstrated "greater preference and self-efficacy in Scratch," which could point to Scratch in comparison to Python and other non-visual languages as a solution to mitigate gender disparity in the computer science field [18].

2.3.2 Consequences of Abstraction in Scratch

Most literature on the subject of block-based programming agrees that the abstraction of syntax is a major positive for inexperienced programmers, saving time and frustration spent on syntax errors [19] [20] [21]. On the other hand, abstraction comes with a tradeoff, leading to a possible lack of explanation of high-level concepts such as objects and variable scope. In addition, Papadakis and other contributors claim that users are at risk to "concentrate on designing and modifying the costumes and sounds of the sprites, without paying any conscious attention to CS concepts" [21]. This is a common worry of those assessing the language, with even the creators of the language acknowledging this possibility at a conference in 2012 while analyzing types of code blocks used by an experienced compared to novice Scratch user [22]. As seen in Figure 16 taken from the conference proceedings, most novice users are not exposed to many programming concepts due to their focus on basic block types that manipulate the sound and looks of their sprite (purple splotches), thus leaving operator (green) and variable (orange) blocks basically unused, which gives some evidence towards Papadakis conclusion [22] [21].

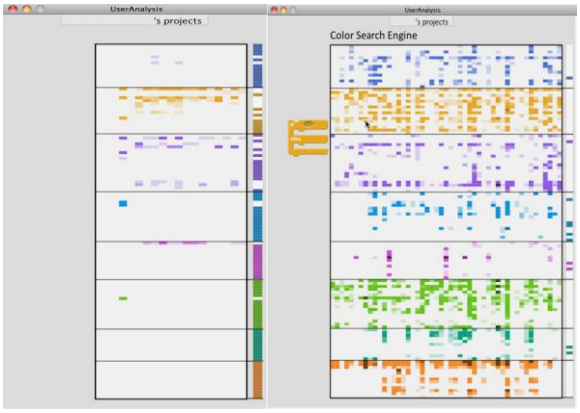

Figure 16: Visualization of the usage of code block types for novice (left) and experienced (right) Scratch users [22]**.**

However, this problem can easily be prevented through the correct implementation of teaching techniques to expand the types of code blocks programmers utilize, and therefore the concepts they are exposed to. A journal written by Mladenović and fellow authors highlights the importance of a teacher's method of instruction, which extends beyond the choice of introductory language [19].

2.3.3 Efficiency and Retention of Concepts After Scratch Use

A research article written by Armoni et al. [19] contained a promising investigation into the effects of utilizing Scratch with middle schoolers to determine if Scratch leads to improved learning when transitioning to different programming languages. The study was implemented over several months, which was a strength since it limited the possibility of inaccurate results due to an environment unlike regular classes. Even in 2015 with an outdated version of Scratch compared to 2024, after students were introduced to programming through Scratch "teachers reported increased efficiency, finding that a concept that took six lessons to teach could now be taught in three" [19]. Improvements in the speed at which teachers can introduce topics could lead to better concept recognition during higher-level programming courses and in tum better retention of material. However, in a study conducted by Monika Mladenović, Žana Žanko, and Marin Aglić, visualization tools utilized during learning had a negative effect on the retention of concepts by students [23]. The authors of the piece hypothesized that the group not exposed to visualization "had to put in more effort to construct valid mental models" of the programming concepts taught, thus leading them to have a better recollection of the material months later based on quantitative data collected after the course concluded [23]. As a note, the study's research group aged from 10 to 11 years old, therefore a study of the same principle conducted on an older group might have different results. Since Scratch's interface focuses on the visualization of variables and code execution, the study's results point toward possible hindrances for young users. If visualization in this instance resulted in lessening students' ability to recall and retain programming concepts, text-based programming with less visualization may be a

better introduction to the subject and lead to better recollection of basics later in a student's academic career.

3. METHODOLOGY

To answer the question of what advantages Scratch provides for novice K-12 programmers compared to Python, gathering scholarly journals and conference proceedings to find studies that could provide some definitive answers was the first step. To gather sources for data, digital libraries were the best place to look and provided several sources with unique goals and methods of gathering answers. While most of the sources did not have the comparison of Scratch and Python as the main purpose of their research, many were able to detail the advantages of block-based programming languages as a whole and how they compare to text-based languages for introductory courses, which was beneficial in answering my research question. Although, a drawback of many of these studies was present in their approach to data collection. Those who choose qualitative data collection could be unreliable and unintentionally manipulated by those assessing test subjects. Therefore, an important step in researching was to collect a mix of both qualitative and quantitative data from studies to make sure the conclusions drawn were based on fact. Studies that focused on the comparison of the effects of Scratch and Python environments on K-12 students were the most helpful but were more difficult to find.

4. RESULTS

While researching the advantages of Scratch compared to Python as a programming language for children, the main points that separate the two are engagement, self-efficacy, levels of retention, and overall complexity. According to a study in the literature review with students in K-12, Scratch is perceived to be less complex by inexperienced users, and with a choice between which language, Scratch or Python, they believed they could learn to program in there was a significant majority who selected Scratch [18]. On the other hand, when study participants involved an older test group, like the study conducted on first-year computer science college students in Nigeria conducted by Cambell et al. [15] and the study on adults by Krafft et al. [17] in 2020, Scratch continued to prove a high level of engagement which is useful to keep students interested in learning programming and continue as computer scientists. At another angle of analysis, according to the results of a study conducted by Armoni et al. [19], when Scratch was introduced to the tenth-grade students before the text-based programming language, higher teaching efficiency was seen on top of higher self-confidence compared to the group who was only introduced to programming through text-based languages. Most journals seem to come to the same conclusion that while Scratch is a useful tool for inexperienced students leading to higher engagement of students, better self-confidence of students, and motivates children to continue to learn programming concepts, it is better suited to be an introductory course for text-based languages, such as Python, and not a replacement [19] [21] [15]. After a close investigation into the current studies published, I have come to agree. Scratch can be an important tool that utilizes visualization to establish a basic understanding of CS concepts, as long as it is taught with careful precision since visualization in general may result in a lack of retention as seen in the study conducted by Mladenović, Žanko, and Čuvić in 2021 [23]. One major point that solidifies Scratch's place as solely an introductory language in my eyes is the stigma surrounding the language as not real, or not as professionally accepted as Python since Scratch is a block-based language [18]. While Zdawczyk and Varma consider this perception as a positive to prevent students' preconceptions about computer science from influencing their behavior in the study, the identification of Scratch as lesser due to its structure does hold some merit when comparing them in a K-12 scope [18]. Students will eventually be forced to move away from block-based programming in favor of more advanced languages that use text-based programming, and thus Python after Scratch is the best way to keep the level of programming low while providing an introduction to syntax errors and other basics of programming they have not experienced in Scratch. Scratch is a useful tool that has been proven to improve self-efficacy, engagement, and motivation in students, which is incredibly important to inspire students to continue to text-based programming and therefore should be utilized as a first introduction to programming, with other major programming concepts still being introduced through Python after Scratch.

A major constraint with this paper is the limited age group of K-12 defined by the research paper. Although it is hard to compare Python and Scratch without first limiting the age group and the level of education the language will be implemented in. For example, Figure 17 shows the most common languages taught at the top 39 computer science departments in the United States for the first introductory college programming course [24]. If the age group researched in this paper were high-level college students, Python would have far more advantages than Scratch. It makes sense why Python would be taught more than Scratch in this setting since its syntax is like English, thus easier to learn than other advanced languages, and is used by software developers to create complex programs for web development, computing, robotics, and tech security [18]. In comparison, while Scratch can also be used to create complex projects, it is less likely to impress a future recruiter at an interview after students complete their degree. Thus, the age grouping of the research question limits the scope of the comparison between Python and Scratch and may result in more biased observations of the languages. Without this limitation, the benefits of each language through each stage of education could be compared and result in a more accurate depiction of the advantages of each language in general.

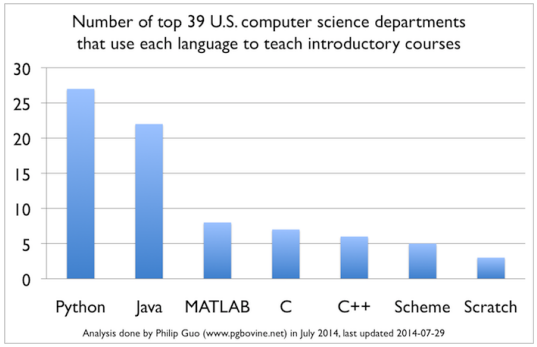

Figure 17: The most common programming languages for introductory college courses at the top 39 computer science departments in the United States in 2014 [24].

5. FUTURE WORK

Another interesting question in extension to the comparison of Scratch and Python as a tool for K-12 students is how the problems with Scratch discovered in this research paper pertain to a potential cybersecurity extension of Scratch called CryptoScratch. Nathan Percival and his team at the University of Massachusetts Lowell wrote on this possible expansion of the Scratch block-based programming scheme to include cryptographic algorithms in 2022 [25]. This topic was chosen based on the critical position cybersecurity plays in protecting a company's data and defending against cybercrimes on top of the need to integrate cybersecurity into young students' curriculum [25]. Figure 18 shows an example of the CryptoScratch interface and blocks available to students in this extension of Scratch. An evaluation of the proposed expansion was conducted through the teaching of several cybersecurity concepts such as hashing algorithms and symmetric cryptography during a workshop and then giving the 16 middle school students eight challenges to complete using the extension [25]. The study found that approximately 60% of students had a basic understanding of the concepts taught during the workshop and approximately 90% of the students showed "increased comfort with cryptography after the end of the workshop" [25]. Thus, Percival and his team agreed that CryptoScratch could be a useful tool to introduce the concepts of cryptography in K-12 classrooms [25]. This result was the same conclusion drawn from this research paper regarding Scratch with its teaching of programming basics.

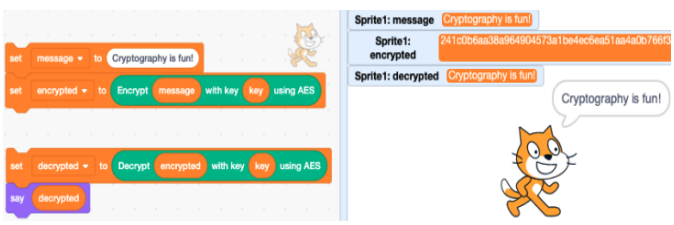

Figure 18: Example of CryptoScratch's use of block-based programming [25].

One problem the team at the University of Massachusetts Lowell had while developing this extension of Scratch was the inclusion of a new Task Block that allowed for automated feedback when utilized by students [25]. While the need for additional code blocks was not a topic concentrated on in this research paper, it is another limitation of block-based programming within Scratch. The new Task Block created in the proposed CryptoScratch extension allows students to drag and drop set blocks inside the structure (like some blocks of Scratch) and also allows a student to select to *get help* and be shown how to complete a task [25]. While the problems found about Scratch and CryptoScratch during analysis are not the same, they both bring valid questions to light about how best to improve block-based programming for K-12 students. One topic of discussion in the future could be whether Scratch should implement a block that also provides feedback like the Task Block in CryptoScratch. Or does increasing the instruction set result in more confusion for users, thus Scratch should stick with its basic set of integrated code blocks?

6. REFERENCES

[1] L. Simmons, "Best Programming Languages for Kids," computerscience.org, 22 March 2023. [Online]. Available: https://www.computerscience.org/resources/best-programming-languages-for-kids/. [Accessed 14 March 2024].

[2] J. Fildes, "Free tool offers 'easy' coding," BBC News, 14 May 2007. [Online]. Available: http://news.bbc.co.uk/2/hi/technology/6647011.stm. [Accessed 14 March 2024].

[3] J. Shapiro, "Your Five Year Old Can Learn To Code With An IPad App," Forbes, 6 August 2014. [Online]. Available: https://www.forbes.com/sites/jordanshapiro/2014/08/06/your-five-year-old-can-learn-to-code-with-an-ipad-app/?sh=59dc60b79b1e. [Accessed 14 March 2014].

[4] M. Resnick, "10 Sparks that Lit the Flame of Scratch," Medium, 2022 2 May. [Online]. Available: https://mres.medium.com/10-sparks-that-lit-the-flame-of-scratch-595a27d44334. [Accessed 14 March 2024].

[5] MIT Media Lab, "Celebrating 10 years of Scratch," MIT News, 11 May 2017. [Online]. Available: https://news.mit.edu/2017/celebrating-10-years-of-scratch-0511. [Accessed 14 March 2024].

[6] Logo Foundation, "Logo History," Logo Foundation, [Online]. Available: https://el.media.mit.edu/logo-foundation/what_is_logo/history.html. [Accessed 14 March 2024].

[7] Logo Foundation, "A Logo Primer," Logo Foundation, [Online]. Available: https://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html. [Accessed 14 March 2024].

[8] M. Tempel, "Blocks Programming," *CSTA Voice*, vol. 9, no. 1, pp. 3-13, March 2013.

[9] Scratch Foundation, "Statistics," Scratch Foundation, [Online]. Available: https://scratch.mit.edu/statistics/. [Accessed 15 March 2024].

[10] Scratch Foundation, "Our Story," Scratch Foundation, [Online]. Available: https://www.scratchfoundation.org/our-story. [Accessed 14 March 2024].

[11] J. H. Maloney, M. Resnick, N. Rusk, B. S. Silverman and E. Eastmond, "The Scratch Programming Language," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1-15, 1 November 2010.

[12] I. Wellish, "User Interface," adafruit, 18 January 2019. [Online]. Available: https://learn.adafruit.com/guide-to-scratch-3/user-interface. [Accessed 15 March 2024].

[13] "Geeks for Geeks," Geeks for Geeks, 3 October 2022. [Online]. Available: https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/. [Accessed 9 April 2024].

[14] A. Swidan, A. Serebrenik and F. Hermans, "How do Scratch Programmers Name Variables and Procedures?," in *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Shanghai, China, 2017.

[15] O. O. Campbell and H. I. Atagana, "Impact of a Scratch programming intervention on student engagement in a Nigerian polytechnic first-year class: verdict from the observers," *Heliyon*, vol. 8, no. 3, 2022.

[16] K. Mork, T. Migler and Z. Wood, "Introducing Computing to a Cohort of Incarcerated Youth," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*, New York, 2020.

[17] M. Krafft, G. Fraser and N. Walkinshaw, "Motivating Adult Learners by Introducing Programming Concepts with Scratch," in *Proceedings of the 4th European Conference on Software Engineering Education (ECSEE '20)*, New York, 2020.

[18] C. Zdawczyk and K. Varma, "Engaging Girls in Computer Science: Gender Differences in Attitudes and Beliefs about Learning Scratch and Python," *Computer Science Education*, vol. 33, no. 4, pp. 600-620, 2023.

[19] M. Armoni, O. Meerbaum-Salant and M. Ben-Ari, "From Scratch to "Real" Programming," *ACM Transactions on Computing Education*, vol. 14, no. 4, pp. 1-15, 2015.

[20] P. G. Feijoo Garcia and F. De la Rosa, "RoBlock – Web App for Programming Learning," *International Journal of Emerging Technologies in Learning*, vol. 11, no. 12, pp. 45-53, 2016.

[21] S. Papadakis, M. Kalogiannakis, V. Orfanakis and N. Zaranis, "The Appropriateness of Scratch and App Inventor as Educational Environments for Teaching Introductory Programming in Primary and Secondary Education," *International Journal of Web-Based Learning and Teaching Technologies*, vol. 12, no. 4, pp. 58-77, 2017.

[22] K. Brennan and M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *Proceedings of the 2012 annual meeting of American Educational Research Associaction Meeting*, Vancouver, Canada, 2012.

[23] M. Mladenović, Ž. Žanko and M. A. Č. , "The impact of using program visualization techniques on learning basic programming concepts at the K–12 level," *Computer Applications in Engineering Education*, vol. 29, no. 1, pp. 145-159, 2021.

[24] P. Guo, "Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," Communications of the ACM, 7 July 2014. [Online]. Available: https://cacm.acm.org/blogcacm/python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/. [Accessed 19 March 2024].

[25] N. Percival, P. Rayavaram, S. Narain and C. S. Lee, "CryptoScratch: Developing and evaluating a block-based programming tool for teaching K-12 cryptography education using Scratch," in *2022 IEEE Global Engineering Education Conference (EDUCON)*, Tunis, Tunisia, IEEE, 2022, pp. 1004-1013.