SCRATCH: A GENTLE INTRODUCTION TO PROGRAMMING

Kyle Snowden
ksnowden@email.arizona.edu

Dylan Fox
dylanfox@email.arizona.edu

April 29, 2020

ABSTRACT

Scratch is a high-level programming language meant to serve as an introduction to programming for kids. In order to appeal to that demographic, it has ample support for visuals, is capable of user interaction, and has a unique drag-and-drop structure to build code. Since it is meant to teach only the basics of coding, it actively hides the implementation of functions and handles errors leniently.

1 Introduction

Scratch is not your traditional programming language. It's what is called a "block-based visual programming language", meaning that instead of typing code out with a keyboard, the code is constructed with drag-and-drop blocks that fit into each other to create logic.

Scratch aims to simplify creating animations, games, and interactive stories. The modern interface has self-contained image and sound editors, allowing users to create all assets within one suite. Scratch includes these features to appeal to 8-16 year old kids and give them a glimpse into Computer Science for the first time.

Throughout this paper, you will see how unique Scratch is compared to other languages. It is important to remember that all of the unique features—and missing features—were deliberate choices which serve the purpose the language was created for. While the language gives the impression of being simple, it was developed by incredibly knowledgeable professionals who aimed to promote both computer science in schools and the development of critical thinking in children. The language intentionally leaves out features like classes, truly custom functions, and advanced data structures so that kids will be able to learn it quickly. [3] The web-interface also carefully limits its tutorials, because it has been found that children learn better when they're allowed to build things on their own. [1]

Scratch is not a useful language for large projects, and has few real-world applications. Instead it is meant to be simple, understandable, and easy for children to make progress with quickly.

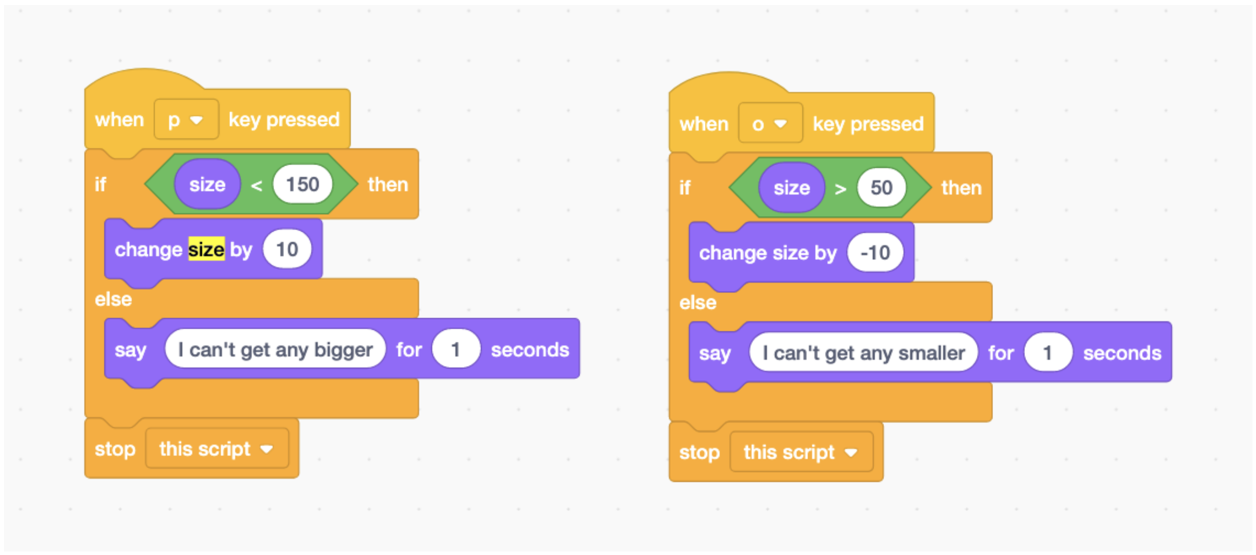
2 History

Scratch began development by the MIT Media Lab in 2003 when the early desktop version of the language was developed. In this initial testing phase, it had heavy input from its young audience by being tested in "Computer Clubhouses" with kids present. Scratch has gone through multiple iterations, and is currently on version 3.0 released in 2019. Since Scratch was developed as an open source platform by MIT, there have been offshoots of the language meant to add certain functionalities. The biggest offshoot is called Snap!, which maintains the use of blocks but adds more advanced features like first-class lists, objects, and functions. Scratch, however, maintained its focus on simplicity and is now used across the globe in over 70 languages. [5]

To further its goal of introducing children to programming, the language's website (https://scratch.mit.edu/) has added tutorials that make learning the language easy and fun. [4] The site makes it easy to post complete projects, search from projects that other people have completed, and communicate with other Scratch users to get answers to questions. Even though these aren't features of the language itself, it's important to consider the context around which people use the language. The developers have provided a safe, kid-friendly interface for Scratch users to program, share, collaborate, and learn. [2]

3 Control Structures

The first thing to realize when it comes to control structures is that Scratch is structured to build visual programs heavy in user interaction. Since this is the focus, Scratch code is commonly organized into many small 'scripts'. Each of these scripts begin with an 'event hat block', which is essentially a handler that waits until a certain condition is met. Commonly these are key presses, mouse interactions, or introductions of new sprites or backgrounds to the screen. The following example highlights these hat blocks, as well as the commonly used 'if-then-else' block:

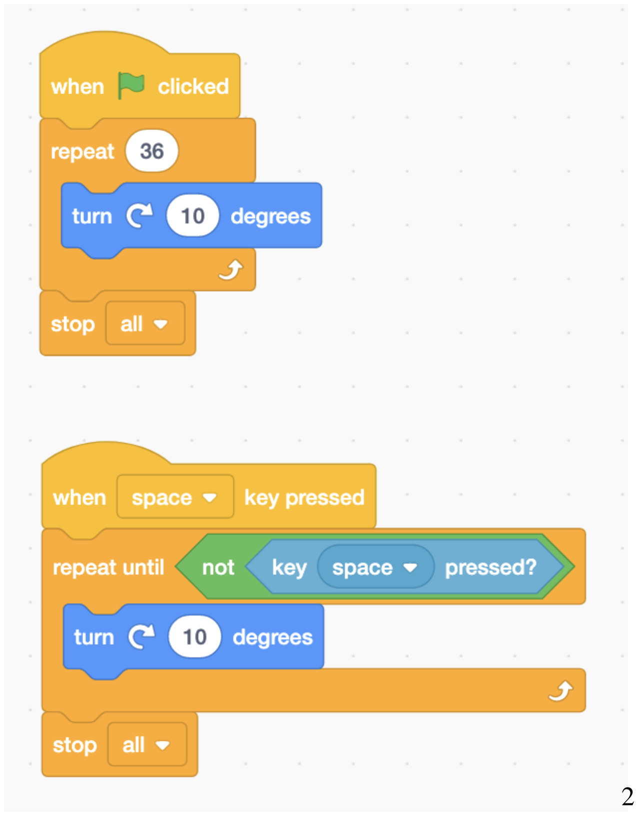


```
when p key pressed
if size < 150 then
    change size by 10
else
    say I can't get any bigger for 1 seconds
stop this script
```

```
when o key pressed
if size > 50 then
    change size by -10
else
    say I can't get any smaller for 1 seconds
stop this script
```

In this example, the scripts respond to key presses by increasing or decreasing the size of the sprite on the screen. The size of the sprite is capped and if it would be changed beyond these caps, instead there is a visual cue letting the user know the limit has been reached.

Another common family of control blocks are the 'forever' and 'repeat' blocks, which essentially function like while loops.



```
when clicked
repeat 36
    turn 10 degrees
stop all
```

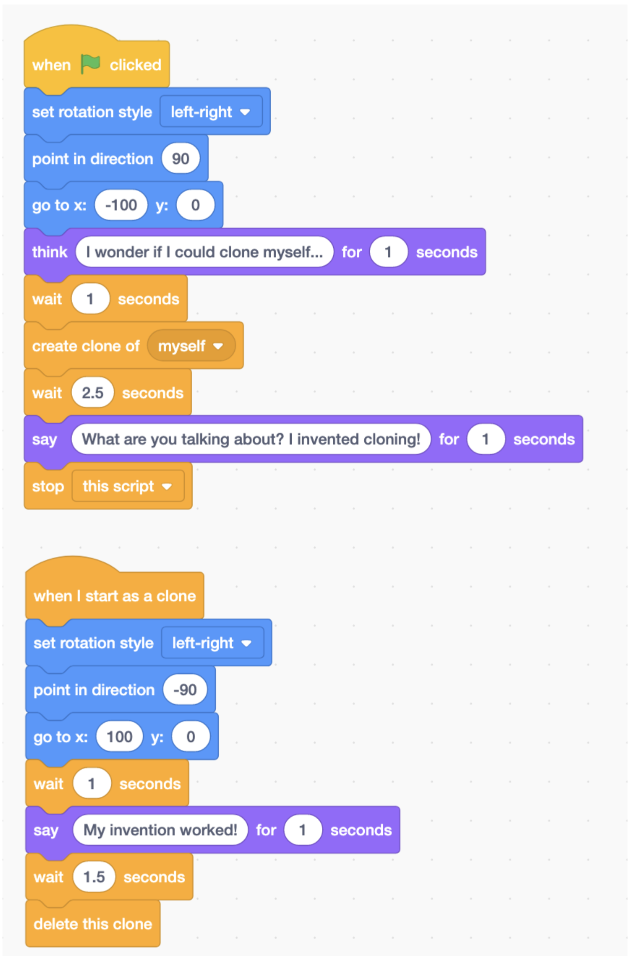
```
when space key pressed
repeat until not key space pressed?
    turn 10 degrees
stop all
```

In the example to the left, both handlers begin with user input. The top script begins when user clicks on the flag—the typical way to start the program—at which point a 'repeat' block will turn the sprite around clockwise. In the bottom script, the sprite will rotate clockwise while the spacebar is pressed. It is important to note that since Scratch is meant to be very user-friendly, it often hides complexity. In both of these cases, the 'turn 10 degrees' command takes an amount of time to execute, creating a smooth rotation.

Another block in this family is the 'forever' loop that will not escape until the 'stop' block is called. This is functionally the same as the 'repeat until <boolean>' block seen the left.

Another interesting control structure is the sprite cloning capability of Scratch. Scratch has extensive support for 'sprites', which are bitmap graphics used along with 'backgrounds' in the construction of a scene. Sprites can be either static or dynamic, and there are unique features which relate to sprites. One of these features is showcased in the following code snippet, which clones a sprite.

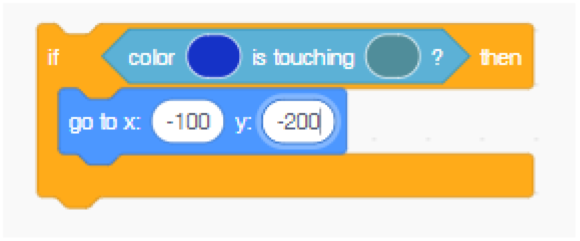
In the program to the right, the program starts when the user clicks on the green flag. After some commands which position the sprite correctly (the commands in blue) the original sprite visually displays the text "I wonder if I could clone myself...". The real magic of cloning happens at the command 'create clone of myself', at which point the second block begins executing. In Scratch, 'cloning' copies only the visual aspect of a sprite, but executes a different script. Continuing this example, the clone says "My invention worked!", after which the original sprite responds, "What are you talking about? I invented cloning!" Both sprites are coordinated so that neither is speaking at the same time.



```
when [flag] clicked
set rotation style [left-right ▾]
point in direction (90)
go to x: (-100) y: (0)
think [I wonder if I could clone myself...] for (1) seconds
wait (1) seconds
create clone of [myself ▾]
wait (2.5) seconds
say [What are you talking about? I invented cloning!] for (1) seconds
stop [this script ▾]

when I start as a clone
set rotation style [left-right ▾]
point in direction (-90)
go to x: (100) y: (0)
wait (1) seconds
say [My invention worked!] for (1) seconds
wait (1.5) seconds
delete this clone
```

There are further control features which stem from Scratch's visual focus. One example of this is a sprite's ability to detect if it is touching another sprite, background, or a specific color. In the example below, the Boolean expression 'color <color1> touching <color2>?' is used as part of an 'if-then' control block.

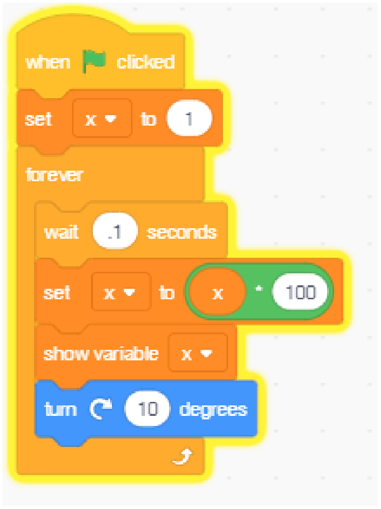


```
if <color (●) is touching (●) ?> then
    go to x: (-100) y: (-200)
```

4 Data Types

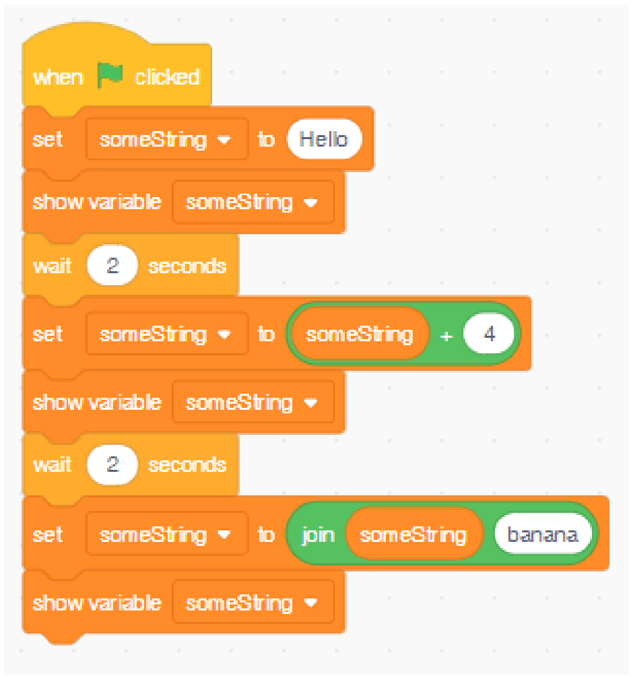
Scratch is an incredibly forgiving language when it comes to data types, and has extensive type inference. Scratch intentionally avoids the complexities of data types by always trying to make the user's code work. Even when data types would normally produce errors, Scratch gets creative to make type errors invisible to the user. For example, take the numeric overflow exception that occurs when a number is too big to be reasonably stored in memory. Scratch will display this number as 'Infinity'. It will not halt the execution of the program, but it will also not preform any more computation on this value and will not preform any operations contingent upon it.

In the program on the right, when the user executes the program with the green flag the sprite will begin to rotate 10 degrees every tenth of a second while x increases quickly. When x overflows it gets the value 'Infinity', and all operations involving it will be skipped. The spite will continue to rotate 10 degrees every tenth of a second since that operation does not rely on the value of x.



```
when [flag] clicked
set [x ▾] to (1)
forever
    wait (.1) seconds
    set [x ▾] to ((x) * (100))
    show variable [x ▾]
    turn ↻ (10) degrees
```

Scratch has just three primitive data types: numbers, strings, and Booleans. When it comes to numbers, Scratch makes no differentiation between 'Int', 'Long', 'Double', or any other number types. Instead Scratch considers all numbers to be be just that: numbers. It will mix decimals with integers and present no problem. Scratch supports this data type with simple mathematical operators, random number generation, and many other more complicated functions. In trying to present all of these data types as the same, there are sometimes little intricacies that arise. As an example, the random function. If you supply two integers as the lower/upper limits, the function with return an integer. However, if you supply two decimal values, then the function will return a decimal value—but curiously, it will never return more than two decimal places of precision. These little oddities come up a lot when using Scratch, and reveals a choice the programmers made when designing Scratch: to always trade flexibility for simplicity.



```
when 🏁 clicked
set someString ▾ to Hello
show variable someString ▾
wait 2 seconds
set someString ▾ to (someString + 4)
show variable someString ▾
wait 2 seconds
set someString ▾ to (join someString banana)
show variable someString ▾
```

Scratch handles operations on strings unlike most languages. If you look at the example to the left you might expect an error to be thrown. The variable 'someString' is set equal to 'Hello', then three lines down the number 4 is added to this string variable. This is an operation that many languages would fail to compute, and, although Scratch cannot perform an addition with a string and number, the variable defaults to number value 4. It is also possible to append a number to a string with the join operator. On the second to the last line the variable 'someString', now set to 4, is joined to the string 'banana' resulting in the value '4banana'. As you can see the language is extremely forgiving and doesn't behave like many other traditional languages.

Scratch supports arrays for both strings and numbers called 'Lists'. Since most programs in this language are heavily visual, Lists see less use than in other languages. Lists support indexing, insertion, deletion, and querying, and are not typed. Any list can contain a mix of strings and numbers. It's interesting to note that Scratch doesn't store values in a list as references to a variable. For example, in the code below, 'x_state' is set equal to 'New York' and added to the List 'States'. The value of the variable is then changed to 'Florida', but this change is not reflected in the List since it was passed by value.

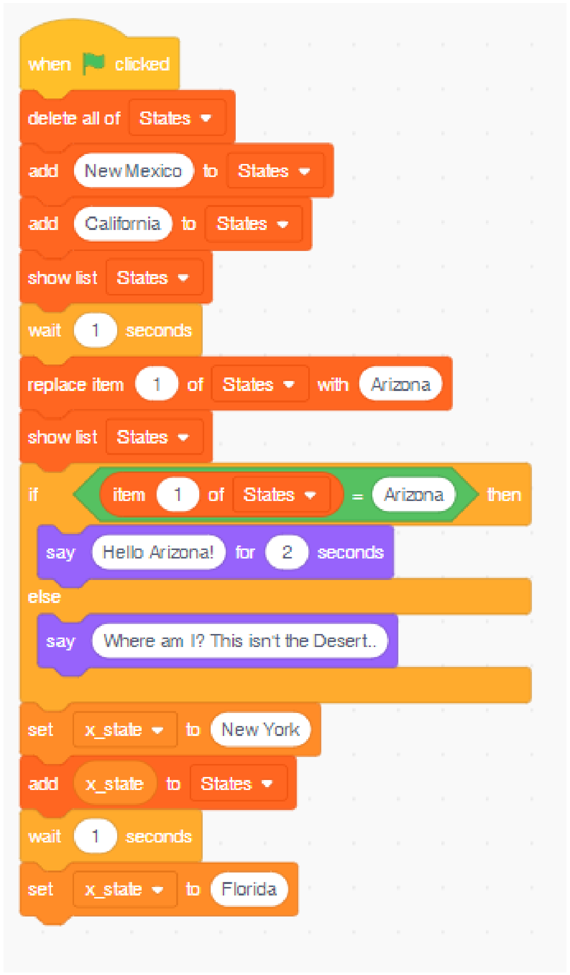
The final primitive data type is Boolean. In Scratch, Boolean components have a unique horizontal hexagon shape which makes it clear to the user that those components always evaluate to True or False. These Boolean components can be the operators 'and', 'or', 'not', '=', '<', and '>', they can be the 'in' function which checks whether an element is in a list, or can be the 'key pressed' function which helps programs evaluate user input.

Scratch has many other data types, they just aren't presented that way. It supports a color data type, date data type, sound data type, and many graphic data types. Color and Date data are expressed through a few predefined functions and cannot be manually created, stored, or changed. And while sound and graphic data types can be manipulated extensively to the user, they function more like pre-created instances of objects and less like true data types.

Scratch has no mechanism for allowing the user to define data types. Since lists can mix data types they may be loosely used as custom data types, but there is no way to enforce this in the code.

It's clear that Scratch is sufficiently different from most programming languages that most ways of analysing data types don't apply or aren't really helpful in understanding the language. Still, it's worth considering how data type concepts apply to Scratch.

Firstly, Scratch has a mix of static and dynamic type checking. Since Scratch code isn't written by typing and instead uses blocks, it was ingeniously designed to enforce static type checking with the shape of each component. Boolean components, for example, are contained in elongated hexagon components



```
when 🏁 clicked
delete all of States ▾
add New Mexico to States ▾
add California to States ▾
show list States ▾
wait 1 seconds
replace item 1 of States ▾ with Arizona
show list States ▾
if <(item 1 of States ▾) = Arizona> then
    say Hello Arizona! for 2 seconds
else
    say Where am I? This isn't the Desert..
set x_state ▾ to New York
add x_state to States ▾
wait 1 seconds
set x_state ▾ to Florida
```

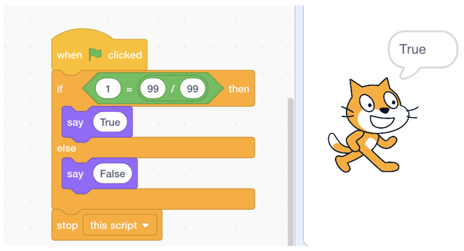
and only fit in spaces that are the same shape (the 'if' statement, for example). Although Scratch tries to avoid errors as previously mentioned, the rest of the errors are dynamically checked at run-time.

Secondly, equivalence. Scratch uses a loose version of structural equivalence in most cases. Since Scratch was designed to be simple enough for children, equality is always whatever makes the most intuitive sense to people unfamiliar with variables, references, and the inner workings of a computer. If two strings have the same characters in the same order, they're equal. If two numbers have the same value, they're equal. By this logic, 1 == 1.00 == 99/99.

Thirdly, Scratch uses polymorphism extensively. Polymorphism is what allows all numeric types to be interchangeable despite them likely being stored differently within the computer. Scratch also makes extensive use of parametric polymorphism by allowing most functions to accept both numbers and strings. Similarly to equivalence, Scratch tries to make polymorphism 'just make sense' to somebody with no programming experience. It doesn't always make sense for a function to take a number and string as an argument, but Scratch tries to use polymorphism intuitively. For example, joining a number and a string will convert the number to a string and then join them, while preforming arithmetic will default a string to 0 and proceed from there.

5 Subprograms

Scratch, as in so many ways listed already, works very differently from other programming languages. Scratch programs are always a single file, and instead of being typed are laid out visually. Each script starts with an 'Event' block that defines the condition which triggers the script. Some of these blocks include 'when space key pressed', 'when this sprite clicked, and 'when backdrop switches to backdrop1'. To emulate a main method there is the event 'when flag clicked', with Scratch's green flag working like a traditional 'run' button. Scratch is organized this way because it is designed around user interaction. This setup makes it easy to plan a program by having modular scripts which each react to distinct key presses or changes on screen, essentially installing a handler for key press.



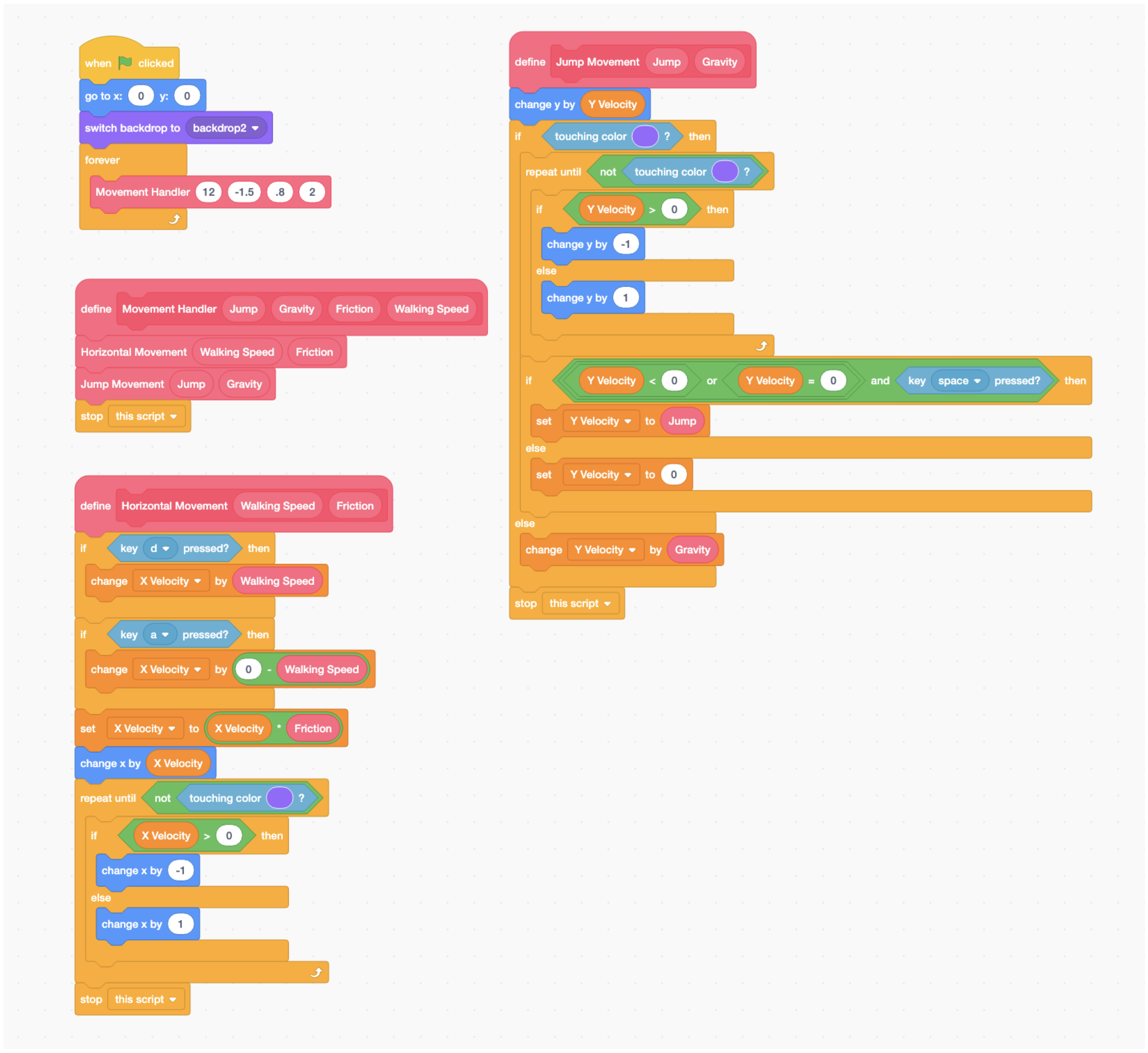
```
when [flag] clicked
if 1 = 99 / 99 then
    say True
else
    say False
stop this script
```

The other unique of subprograms is cloning, which was discussed above under Control Structures.

Scratch also has support for function calls. These are called 'Blocks', and allow users to create their own functions. The user is allowed to define the name of these functions as well as the type and order of the parameters. To simplify things for the user, there are only two types of parameters that are allowed to be passed to a function: Booleans, and a field which accepts numbers or strings. Although simplified, functions can still be combined complexly and preform recursion.

While most programming languages provide documentation detailing specifications on how subprograms like these work, Scratch does not. Since Scratch is committed to hiding implementation details behind a carefully thought out programming interface, this is all the level of detail that can be provided. It's safe to say, based on performance, that there is significant overhead involved in creating a function and especially in creating a clone. However, it's difficult to get exact details since Scratch doesn't provide diagnostic tools like exact-time printouts to track how long a section of code took to execute.

On the next page, you'll see a Scratch program which ties together most of the features discussed in this paper. Particularly, note how the visual layout of this somewhat complex code makes it easier to understand without formal understanding of coding principles.



```
when ⚑ clicked
go to x: 0 y: 0
switch backdrop to backdrop2
forever
    Movement Handler 12 -1.5 .8 2

define Movement Handler Jump Gravity Friction Walking Speed
Horizontal Movement Walking Speed Friction
Jump Movement Jump Gravity
stop this script

define Horizontal Movement Walking Speed Friction
if key d pressed? then
    change X Velocity by Walking Speed
if key a pressed? then
    change X Velocity by 0 - Walking Speed
set X Velocity to X Velocity * Friction
change x by X Velocity
repeat until not touching color ?
    if X Velocity > 0 then
        change x by -1
    else
        change x by 1
stop this script

define Jump Movement Jump Gravity
change y by Y Velocity
if touching color ? then
    repeat until not touching color ?
        if Y Velocity > 0 then
            change y by -1
        else
            change y by 1
    if Y Velocity < 0 or Y Velocity = 0 and key space pressed? then
        set Y Velocity to Jump
    else
        set Y Velocity to 0
else
    change Y Velocity by Gravity
stop this script
```

6 Summary

And that's it! In just five pages, the majority of what there is to learn about Scratch. This language was developed to be simple to explain and intuitive enough to program in with no experience. This development goal, different from any other programming language, leads to a unique set of features. Scratch is not fast, is inefficient in resource use, has minimal extendable library interactions, and is painstakingly slow to write in due to its drag-and-drop coding. And while this set of features makes it unusable for any real-world applications, it also helps it to be easy to learn and understand. It makes it simple to move a sprite on the screen, and have it react to user input. And it engages children while teaching them the basics of programming. Scratch isn't an amazing language because it's practical—it's an amazing language because it accomplishes its goals perfectly.

References

[1] Niels Bonderup Dohn. Students' interest in scratch coding in lower secondary mathematics. *BJET*, 51(1):71–83, 2020.

[2] MIT. Scratch website, 2020. The main hub for the Scratch language.

[3] Diana Pérez-Marín, Raquel Hijón-Neira, Adrián Bacelo, and Celeste Pizarro. Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children? *Comput. Hum. Behav.*, 105:105849, 2020.

[4] Tracy Gardner Rik Cross. *Book of Scratch*. Raspberry Pi Trading Ltd, 2018.

[5] Scratchers. Scratch wiki. https://en.scratch-wiki.info/wiki/, 12 2008.