SCRATCH: INTRODUCING KIDS TO PROGRAMMING

Kyle Snowden
ksnowden@email.arizona.edu

Dylan Fox
dylanfox@email.arizona.edu

April 21, 2020

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim 2justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt.

1 Introduction

```
main() {
}
```

2 History

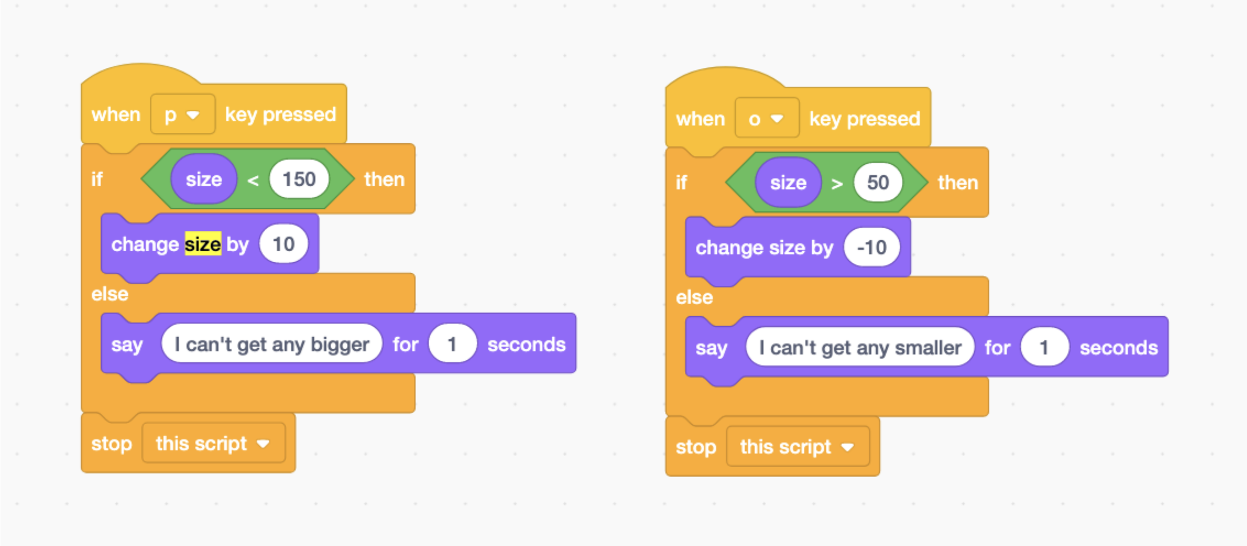
Scratch is not your traditional programming language, it's what's called a block-based visual programming language that was developed by the MIT Media Lab. It's primary purpose is to educate it's users of concepts and skills that can then be applied to other languages such as Java or C. The language first appeared in 2003 with the first desktop version of the language was developed, however it wasn't until 2007 when it was released to the public. The goal of the project was to teach young children to code in an easy fun and interactive way. [1]

Today Scratch is currently on version 3.0, released in 2019, replacing it's predecessor Scratch 2.0 which was released on May 9, 2013. Today Scratch is used in many places across the globe and has been translated into 70+ languages. It is very prevalent in classrooms in all age ranges, scratch was developed in close coordination with a young audience at "Computer Clubhouses to maximize it's ease of use and educational effectiveness.

Scratch aims to simplify creating animations, games, and interactive stories, and simulations. Scratch 3.0 has it's own self contained paint editor and sound editor allowing users to create assets all within one suite. Scratch targets kids within the age range of 8-16 years old, often giving the kids a brief glimpse into Computer Science for the first time.

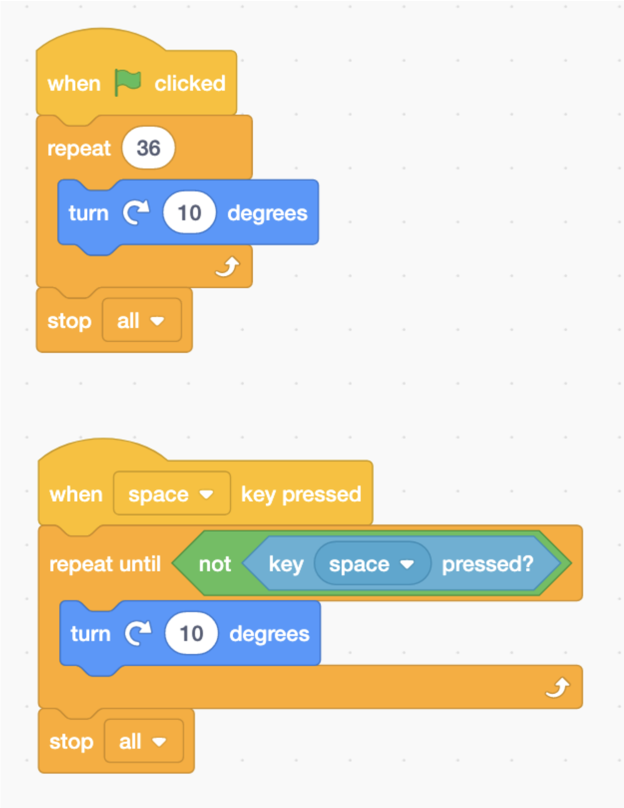
3 Control Structures

The first thing to realize when it comes to control structures is that Scratch is structured around 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, and 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:



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.

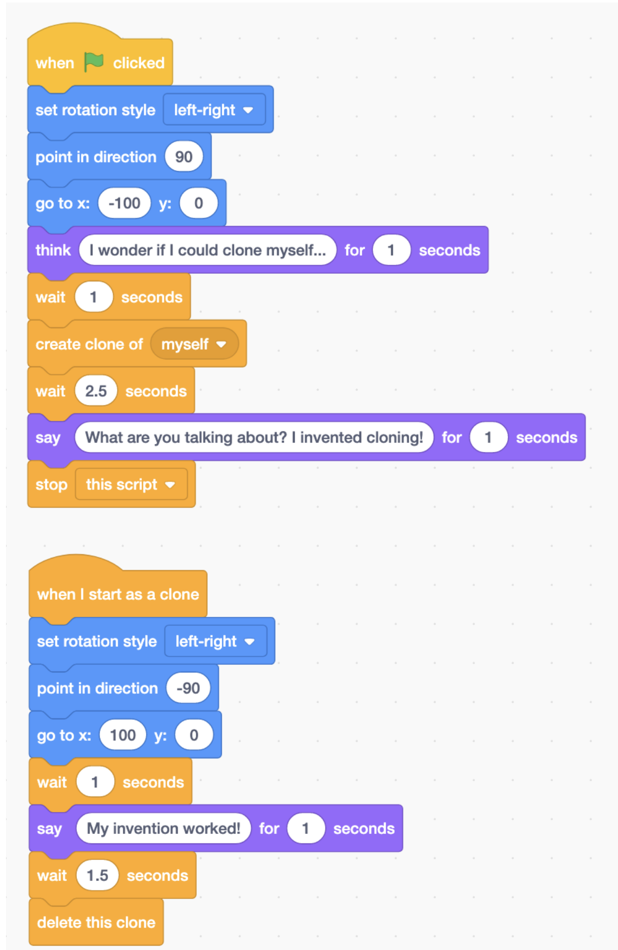


In the example to the left, both handlers begin with user input. The top script begins when user clicks on the flag—the typical way to start the program—at which point a 'repeat' block will turn the sprite around clockwise. In the bottom script, the sprite will rotate clockwise while the spacebar is 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.

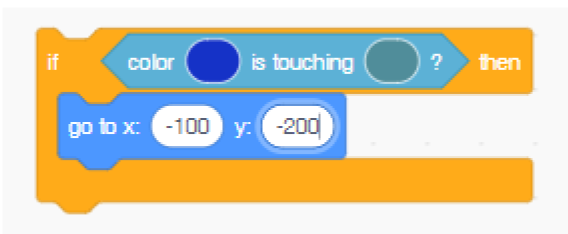
Another block in this family is the 'forever' loop that will not escape until the 'stop' block is called. This is functionally the same as the 'repeat until <boolean>' block seen the left.

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.



There are further control features which stem of 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.



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 as minimal as possible. 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 forever keep rotating 10 degrees every tenth of a second and x can be observed increasing quickly. At a given point x becomes to large for scratch to store and perform operations on and is displayed as 'Infinity.' Regardless of this the sprite will continue to rotate 10 degrees every tenth of a second, skipping the computation performed on x.

Scratch has just three primitive data types: numbers, strings, and Boolean. 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 come up. 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: trade flexibility for simplicity every time, in every way.

Scratch handles operations on strings unlike most languages. If you look at the following example you might expect an error to be given. The variable 'someString' is equal to 'Hello' then three lines down the integer 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 instead just takes on the number value of 4. It is possible to append an integer to a string and visa versa with the join operator. On the second to the last line the variable 'someString', which now equals 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 based visually, 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 if you look at the code below on line 13 'x_state' is set equal to 'New York' then on the next line added to the list 'States'. The value of the variable is then changed to 'Florida' however this is not reflected in the list as it was passed by value and not by reference.

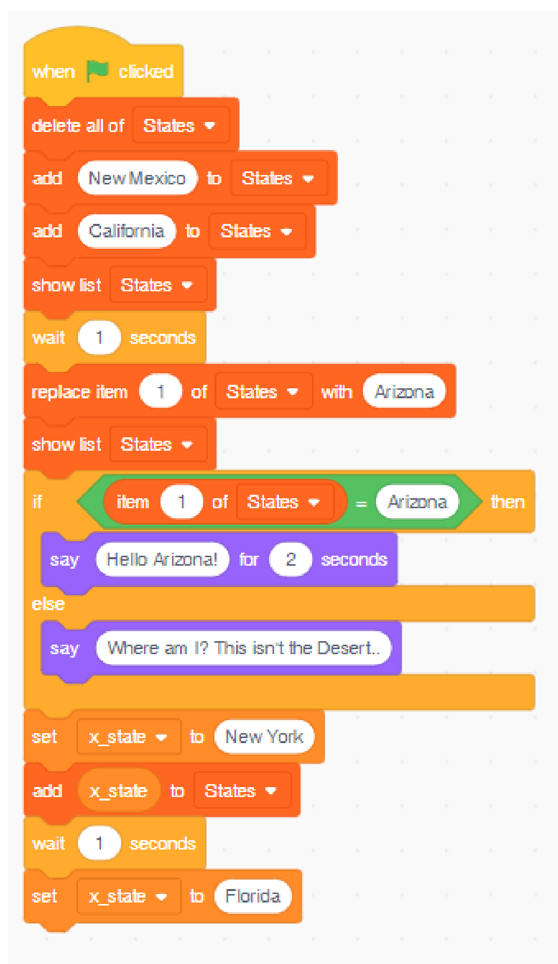
The final primitive data type is Boolean. In scratch, Boolean components have a unique horizontal hexagon shape which makes it clear to the user that those components always evaluate to true or false. These Boolean components can be the operators 'and', 'or', 'not', '=', '<', and '>', they can be the '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 and only fit in spaces that are the same shape (the 'if' statement,

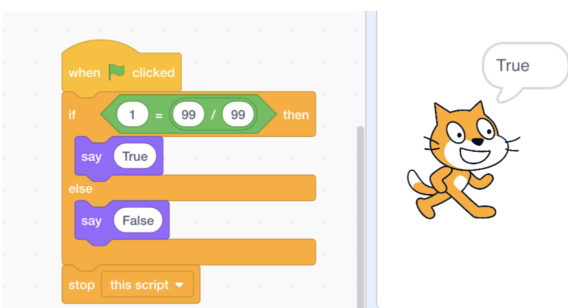


for example). Although Scratch tries to avoid errors as previously mentioned, the rest of the errors are dynamically checked at run-time.

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 out are laid out visually. Each script starts with an 'Event' block which defines the condition which triggers the script. Some of these blocks include 'when space key pressed', 'when this sprite clicked, '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 very easy to plan a program by having very modular scripts which each react to a distinct key press or change on screen.

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 whether the

6 Summary

References

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