Introduction to Computer Programming:

SCRATCH

1.

Getting Started

Computer programming may seem to be a magical skill out of reach, especially without the guidance of an instructor or classroom to work in.

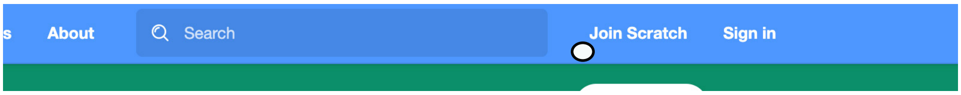
Today, the tools that have been created to teach you about programming have made it easy to do at home and at your own pace. In this booklet you will be guided through three assignments that will make up the beginning of your coding portfolio, starting from Scratch. Scratch is a software designed by Massachusetts Institute of Technology to help people learn the structures of C-based programming languages. C is a computer programming language that was developed in the 70's. As our needs and potentials with computers expanded, new languages had to be constructed from older ones to account for new capabilities. Languages based off of C that are commonly used today include: Java, Javascript, PHP, C+, and Python.

The drag and drop coding tiles that are used in the Scratch interface help the user focus on learning coding vocabulary and syntax without having to worry about memorizing lists of methods and elements. This characteristic is a powerful asset in averting frustration when learning to code, as forgetting elements can be a common occurrence for early coders, and can prove be a trouble to keep track of when learning the more structural parts of programming. The skills you learn in these assignments will be excellent building blocks for approaching any C-based language, as they will share similar syntax and structures.

Important to remember throughout these assignments, is that if you're doing this for the first time, it is more than okay to mess up a couple times. Failure is inevitable when trying new things. Simply make note of how you tried to solve the problem, and give it another go, cross-referencing when you get it right to see where you went wrong. Errors are even common by career coders, (think of all those updates you get all the time on your phone apps) so boot up your computer, and get to experimenting with a new skill!

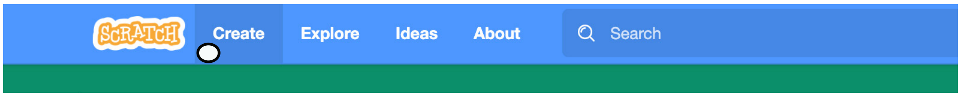
Let's take a look at the web-application:

We can get started by going to https://scratch.mit.edu/. Create an account with Scratch by clicking the "Join Scratch" button on the right side of the top toolbar. After following the prompts in the dialog, your account will be created.

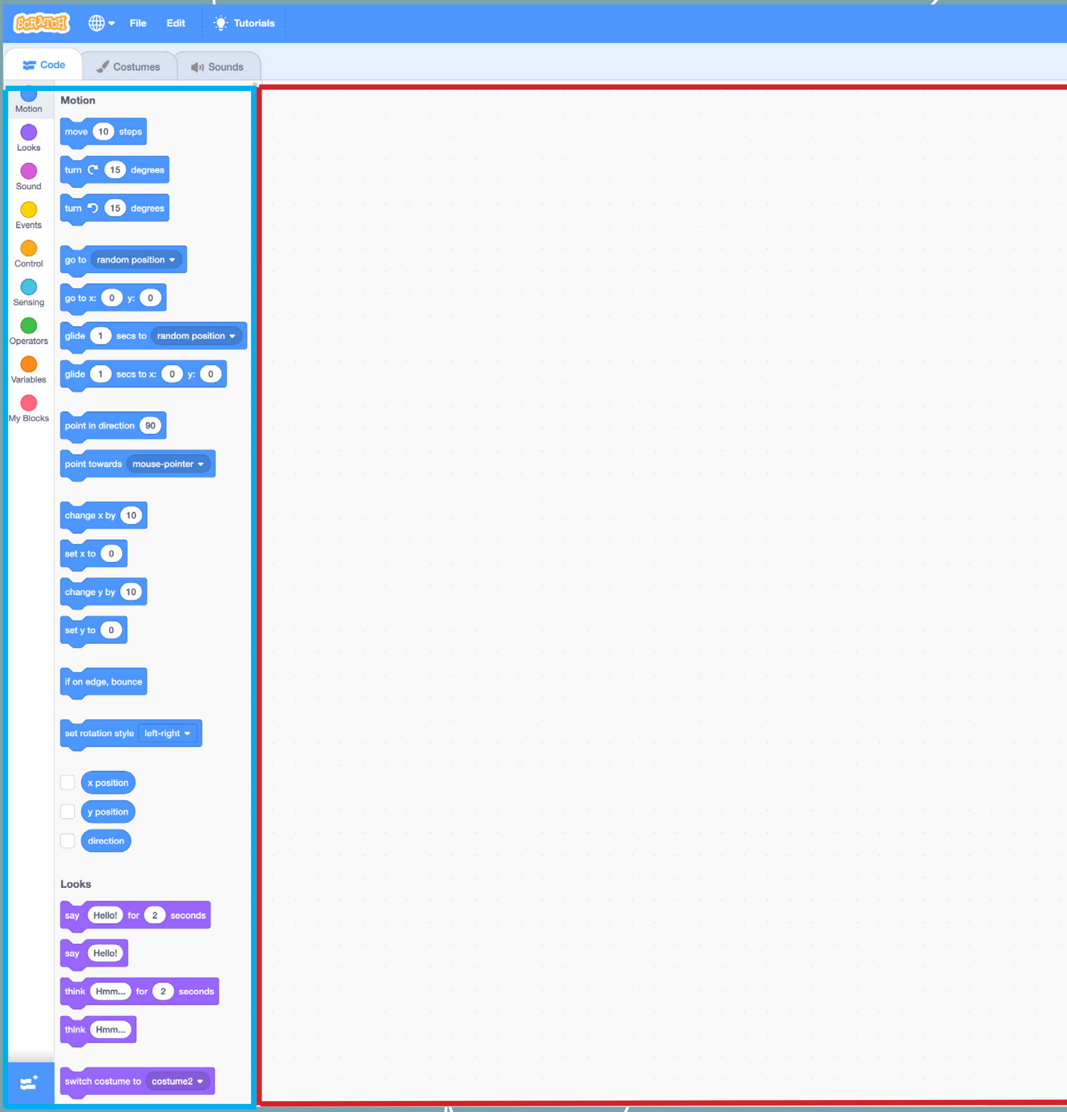


Click here to sign up for your free Scratch account!!

Signing in will bring you to a new screen, where you can click the "Create" button in the top toolbar to begin your first project.



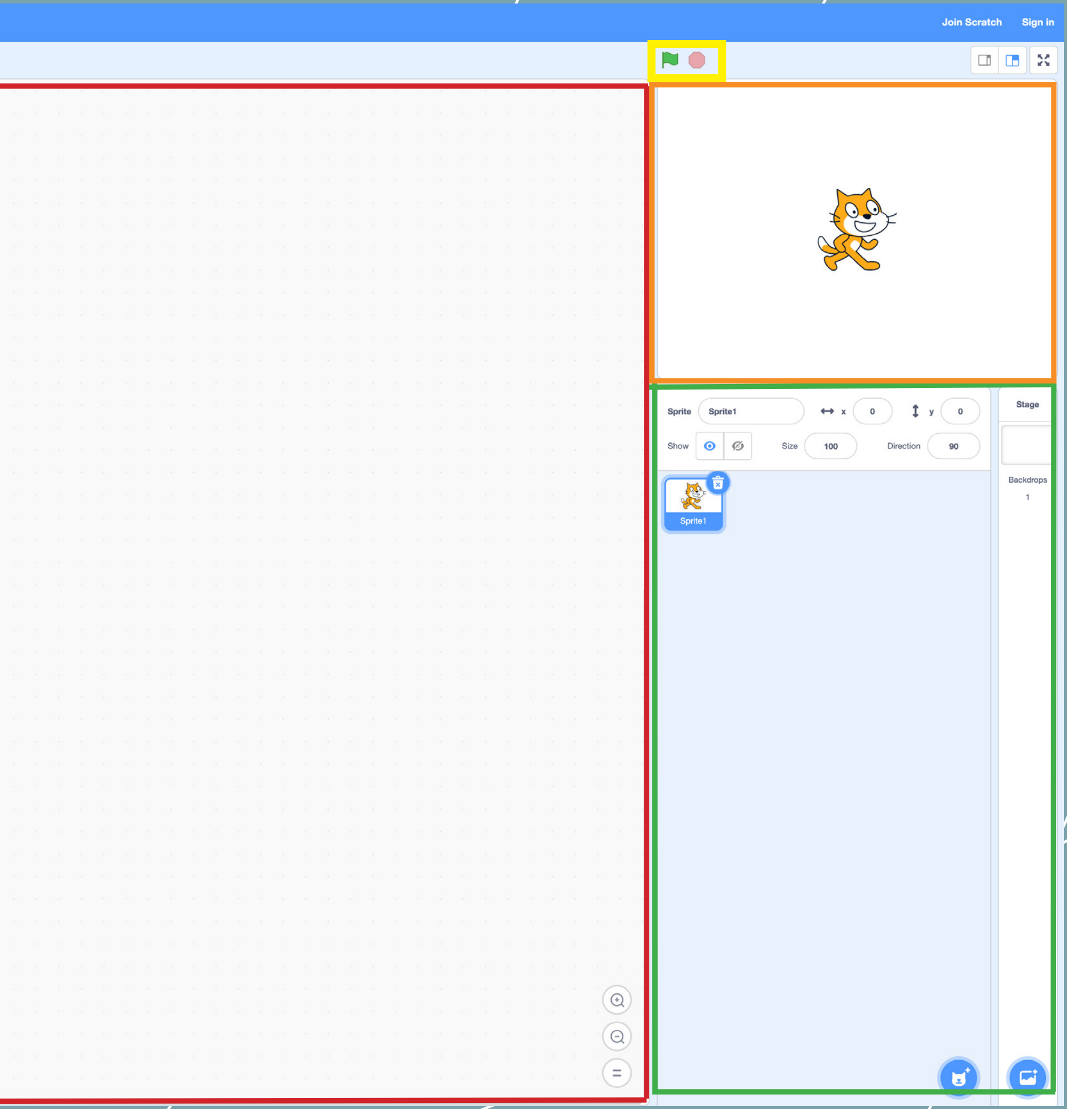
Click on the "Create" tab to begin coding!



Inside the “Create” page in Scratch, you will be presented with various panels that help to manage your workflow.

Starting from the left (outlined in blue) we have our “Tile” panel. In this section, you can find the collection of code snippets that Scratch equips users with to create their own programs. These code snippets are color-coded in their abilities, such as green tiles being used for mathematical operation, and yellow panels for events. These tiles can be dragged and dropped into our next panel, the “Developer” panel.

The Developer panel (outlined in red) will be where the coding will take place. By dragging code snippets this panel, they become active in the program file. These snippets will clip together, like magnets, and will activate if given trigger that calls them.



Sprite Sprite1 x 0 y 0
Show Size 100 Direction 90

The green box is surrounding the "Sprite" panel. Sprites are objects that interact with code by either being trigging, or being modified by code snippets. In the panel, you can add new Sprites by clicking the "Choose a Sprite" button at the bottom of this panel, or edit properties of existing Sprites, such as color, name, and position.

Right above the Sprite panel is the viewer panel (bordered in orange). Sprites can be positioned by dragging them in this panel, but primarily, this is where you will see your code run after you run it. This panel can be expanded to full screen.

Lastly, the yellow box above the viewer panel signifies the Start and Stop buttons. These buttons will run your code, or stop its running respectively.

2.

Your First Program

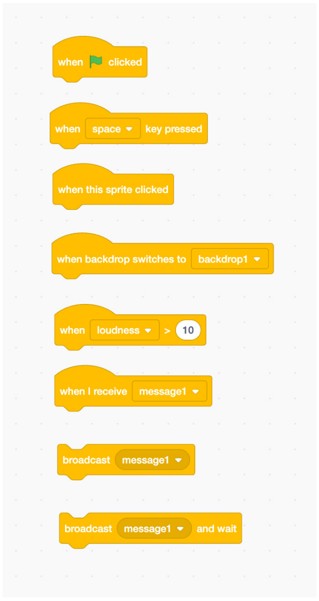
In your first program you will be getting familiar with the Scratch interface as well hopefully develop an understanding of the different types of code snippets available in Scratch.

Our first assignment will trigger a sound upon running the program.

The primary goal of this assignment is to help you to grow comfortable with the Scratch interface and its different components. After completing this assignment, feel free to return to this file and try-out other code snippets after the trigger, it will help with understanding what each one does.

In order to get started, start up your web browser and login to your Scratch account and open a new "Create" file in the top navigation bar on the web page. Once you get this pulled up, you can start the coding!

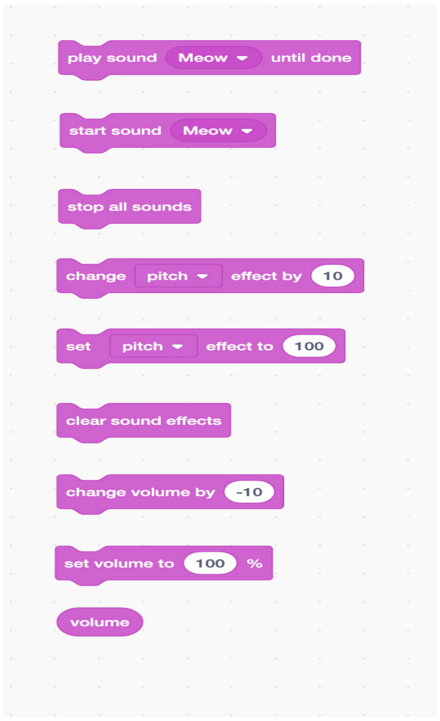
As a little pre-activity lesson, we will go over the snippets that you wil be using in this assignment:



```
when [green flag] clicked
when [space ▾] key pressed
when this sprite clicked
when backdrop switches to [backdrop1 ▾]
when [loudness ▾] > (10)
when I receive [message1 ▾]
broadcast (message1 ▾)
broadcast (message1 ▾) and wait
```

The first color category that we will be taking code snippets from will be the yellow "Events" category. Events serve as triggers for the lines of code. This could be a button press, a page refresh, or even the program start-up. In this program, we will use the first one in the list, "When [green flag] clicked" which will trigger our code to run at the start of the program.

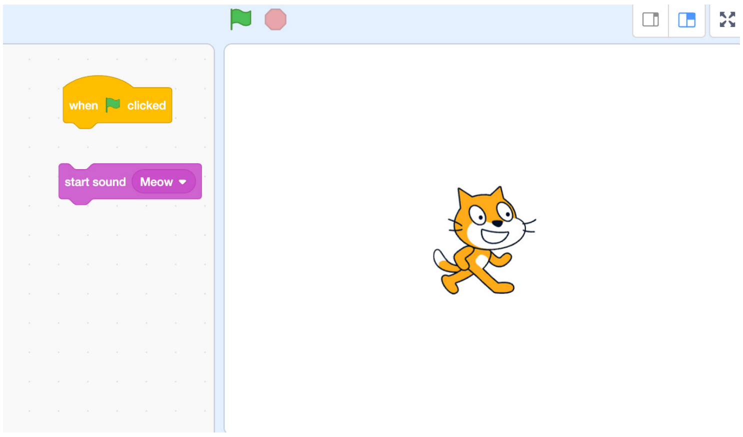
For the second category we will go over in this assignment, we will take a look at the magenta "Sound" category. These code snippets deal with causing sounds to play. By using a sound code snippet you can make buttons that change the volume of the sound in the application, as well as buttons that trigger, or silence sounds. In this application we will be using the "Start sound" button, which will come preloaded with the "Meow" sound file. You can record your own sound files for your Scratch application by clicking on the dropdown in the sound code snippets.



```
play sound (Meow ▾) until done
start sound (Meow ▾)
stop all sounds
change [pitch ▾] effect by (10)
set [pitch ▾] effect to (100)
clear sound effects
change volume by (-10)
set volume to (100) %
(volume)
```

Now that you have the basics of these tools, let's put them to work:

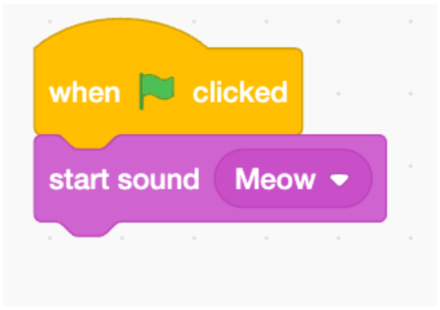
All you need to do now, is drag both of the blocks mentioned on the last page from the Tile panel to the Developer panel.



Now click the start.

Right now, with both of these snippets floating around in the Developer panel, neither snippet is doing anything. One is a trigger without anything to trigger, and the other is an output, with nothing to trigger it. In order to make our code do something, we will need to attach the pieces.

To connect the code snippets, drag the sound block's open groove toward the matching notch on the trigger block. When close enough, a shadow will appear between the snippets, and allow them to snap together by unclicking the block that you are dragging.



When we connect code snippets in this way, we get what is called a “block” of code. A block of code is a chunk that serves a specific purpose and function, and is usually sectioned off from other code by a commment section.

Now let’s click start again.

“Meow!”

Now with our trigger and code connected in a block, we get our desired outcome when we run our program. Now that you have the basics of computer programming under your belt, use the same event trigger to see what the other code snippets in the Tile panel do. Scratch provides tons of different ways you can affect your program by combining different snippets into block combinations. Give it a try before moving on to the next program!

Another tip: once you connect multiple snippets in a block of code, you can move the whole block as one piece by clicking and dropping the top tile.

3.

A New Step Forward

In our last program, we made our computer “Meow” by running the program that we wrote. This project only had one code snippet run in it, and it only ran when we started the program. By using websites and applications as much as we do these days, we know that computers are doing things even when we aren’t clicking them. In this project, we will take a look at loop structures.

Loops are used when developers want the computer to run one block of code multiple times, perhaps even infinitely. They are built by “nesting”, or placing other lines of code inside the bounds of the loop. In order to do this, loop statements have a starting and closing marker to signify which code is inside the loop, and which is outside.

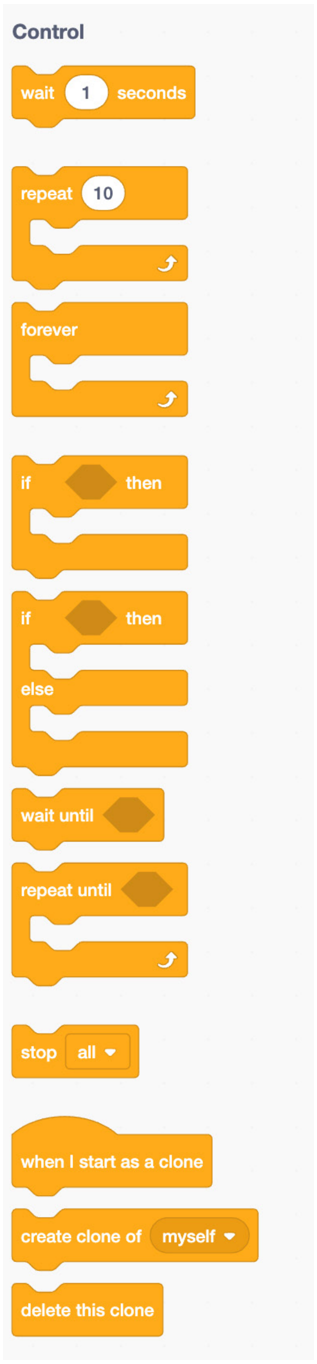
Let’s give it a shot!

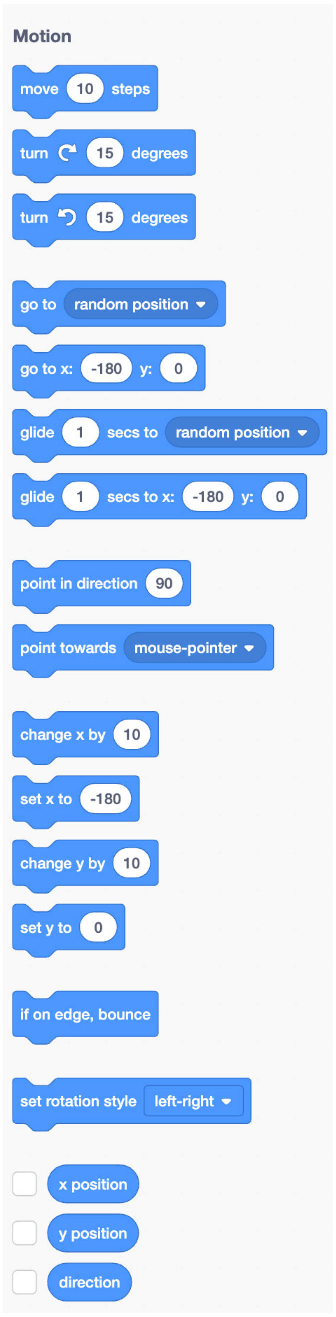
Before we get started with this assignment, let's go over a couple more Tile categories:

Controls

The orange code snippets in the Tile panel signify "controls". These controls include our loop structres, as well as other orders that can delay, interupt, or even duplicate code and objects.

In this assignment we will be using the "forever" loop, as well as the "wait" function.





Motion

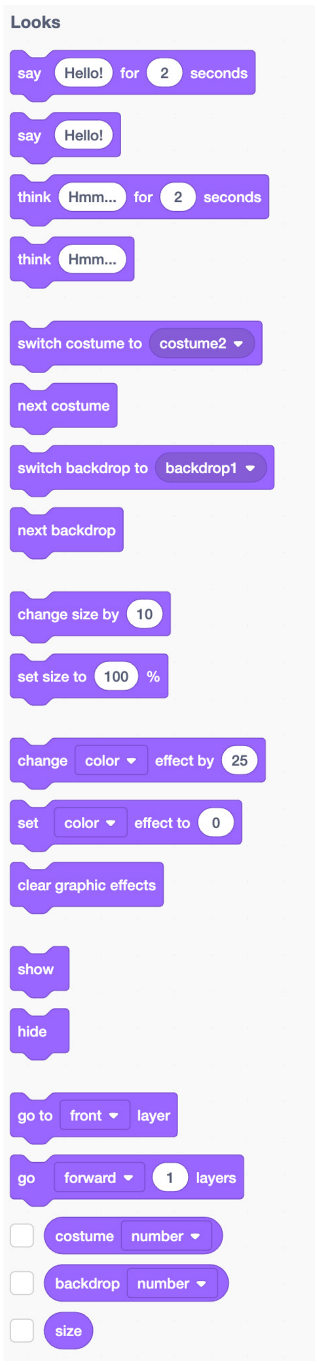
The dark blue tiles are the “motion” category. These tiles deal with moving and orienting objects on the developer’s panel as a result of code. This can be used to rotate
something on screen with a button press, or to move around a character in a video game.

In this assignment we will be using the “move 10 steps” action.

Looks

The purple code snippets relate to the looks of the objects in the display panel. We can use these functions to activate speech bubbles, switch appearances of our objects, or change their size and color.

In this assignment we will be using the "next costume" function.



```
Looks
say Hello! for 2 seconds
say Hello!
think Hmm... for 2 seconds
think Hmm...

switch costume to costume2 ▾
next costume
switch backdrop to backdrop1 ▾
next backdrop

change size by 10
set size to 100 %

change color ▾ effect by 25
set color ▾ effect to 0
clear graphic effects

show
hide

go to front ▾ layer
go forward ▾ 1 layers
☐ costume number ▾
☐ backdrop number ▾
☐ size
```

Back to coding!

Now that we have our new vocabulary figured out, we can get to putting our program together!

For our second project, we are going to code a program that will cause our cat Sprite to run across the screen. In order to do that, we will need to create a block that does a couple things:

- moves our Sprite
- changes its costume to make our Sprite look like it is running
- & loops these actions until the Sprite crosses the screen

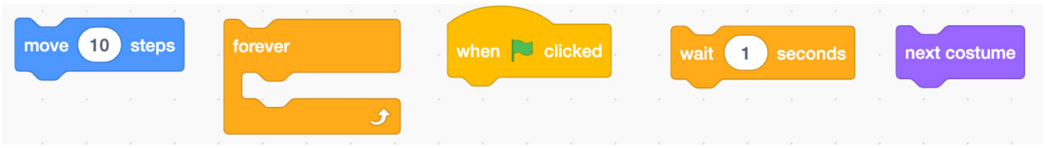
From the resources before this assignment, we know which pieces we will be using for this code:

- [MOTION] Move 10 Steps
- [CONTROL] Forever loop
- [CONTROL] Wait
- [LOOKS] Next costume

We also know that in order for things to be repeated, they must be nested inside the loop structure.

Using the shape of the blocks as guidance, sketch out what you think this block of code will look like in the blank page next to this one, before we go through it together.

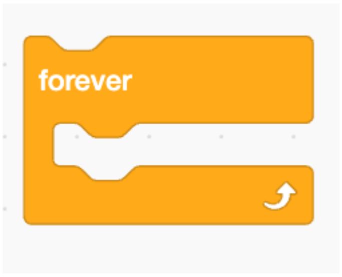
Use the blocks below and what you learned in the last section to draft the code for this assignment!

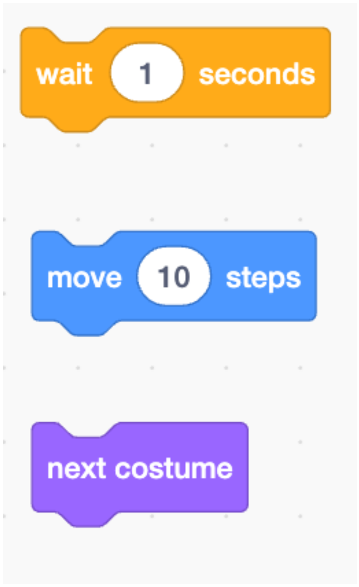


Continue the assignment on the next page...

Let's get started by building our loop:

Since we want all of our other tiles to repeat, we will start with the repeat forever tile. Nesting our other tiles inside this one will cause those tiles to repeat, while the condition for the loop (forever) is met.

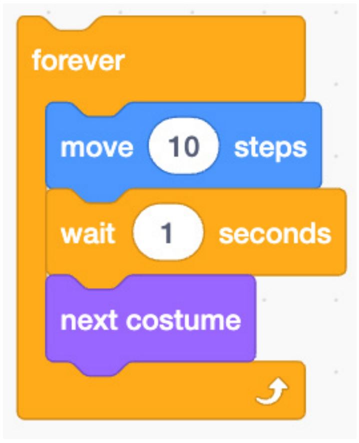

```
forever
```


```
wait (1) seconds

move (10) steps

next costume
```

Now the only problem to solve is what order do we put our tiles in inside the loop? Does it matter?

In Scratch, the computer will read all of the code line by line and execute commands in functions. This forever loop is an example of a function that alters more than one part of the program. Usually, all of the code in a function would appear to happen silmutaneously, but the implementation of a command that says otherwise (like our wait tile) can cause certain parts of the code to be executed at different times. Knowing this, the placement of the wait tile becomes important.

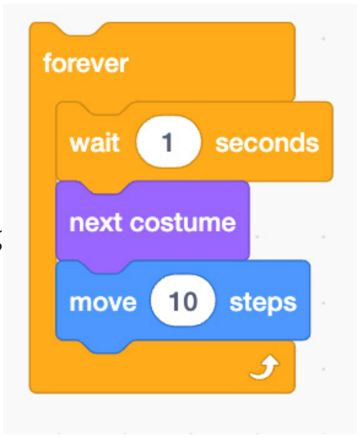
Let's take a look at the potentials:



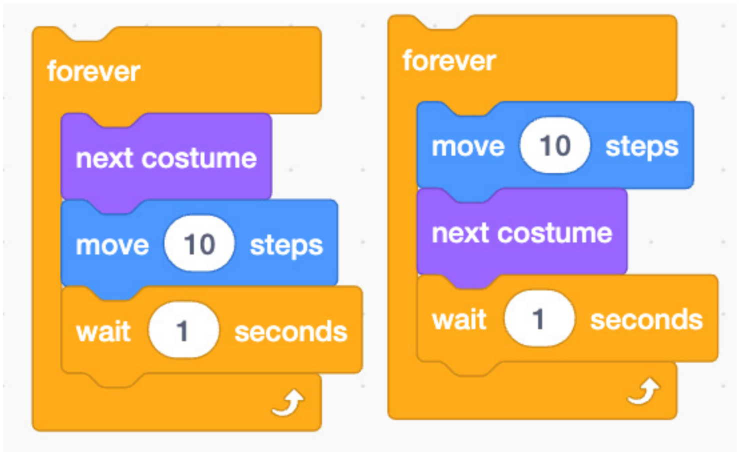
```
forever
    move 10 steps
    wait 1 seconds
    next costume
```

By looking at this instance of our loop function, we can read it from top to bottom as "forever, our Sprite will move 10 steps, wait, and then switch costumes". With wait in the middle of the two other tiles, the next costume tile will not execute until a second after the move tile. Because the intended effect of switching the costume is to make our Sprite look like it is running when it moves, the first time this loop runs, the costume will not change until after our Sprite has moved failing to emulate the appearance of running.

In this instance of the loop function, we can read our code as "our Sprite will wait 1 second, and then switch costumes and move 10 steps, then repeat these actions forever". While this code works, it may not be the best option. By having the wait block at the beginning of the function, the program pauses between pressing the launch button and it appearing to begin, since no visible change occurs immediately. In order to avoid this, which may appear as a lag, we can move the wait tile to the bottom of our function.



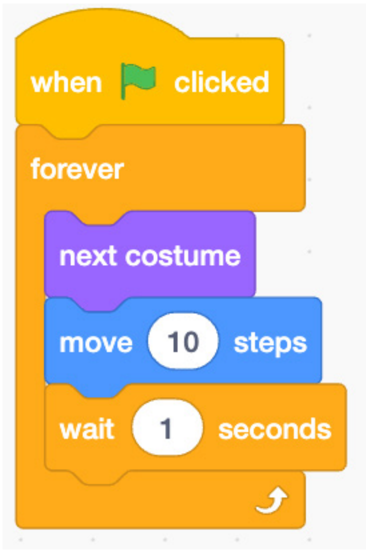
```
forever
    wait 1 seconds
    next costume
    move 10 steps
```



```
forever
    next costume
    move (10) steps
    wait (1) seconds
```

```
forever
    move (10) steps
    next costume
    wait (1) seconds
```

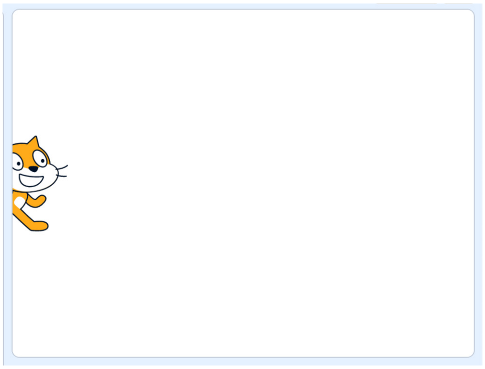
With all of that being said, either of the above functions would be optimal for this assignment. Because code will occur silmutaneously unless programmed otherwise, the next costume and move tiles will run at the same time, and then our Sprite will pause for 1 second before repeating forever.

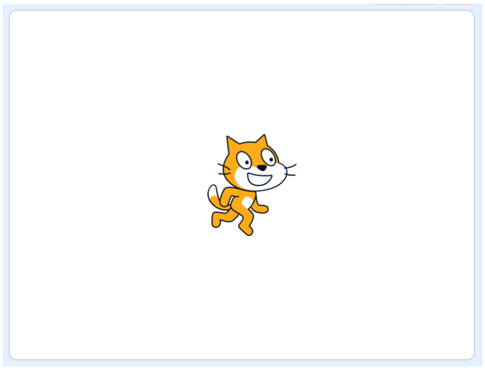


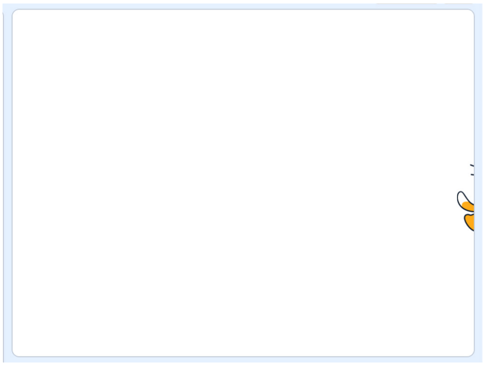
```
when 🏁 clicked
forever
    next costume
    move (10) steps
    wait (1) seconds
```

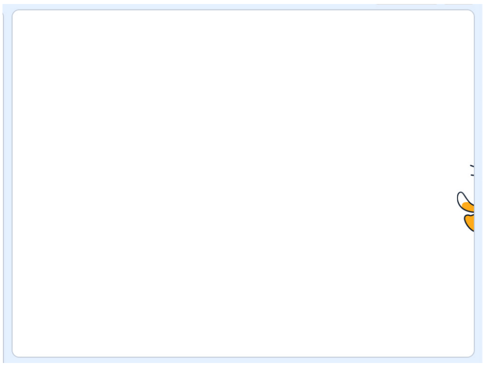
Now all that is left is to attach the event that triggers our function!

Once you do this, you can position your Sprite to the left of the viewer panel and then press Start to watch your Sprite move across the screen!









As you can see, once our Sprite makes it to the right side of the screen, It does not understand that it cannot go any further, and continues to try to run off of the edge of the viewer panel, but gets stuck. this is because in our viewer panel we have bounds that the objects must fit inside of. These physical bounds are determined by x and y coordinates, starting at 0 in the center of the panel, and going up to 272 when moving upward or to the right, or down to -272 when moving downard or to the left.

Our move block references the x number that controls the left and right position of our Sprite. By moving 10, we change our x value by 10, moving 10 points to the right. When we reference a number that stands for meaning outside of its amount, we are calling a variable. Variables can control things such as position on in the panel, as well as the value that we have in the wait block. By changing the variable in the wait block, we can alter how long the loop pauses before it runs another iteration.

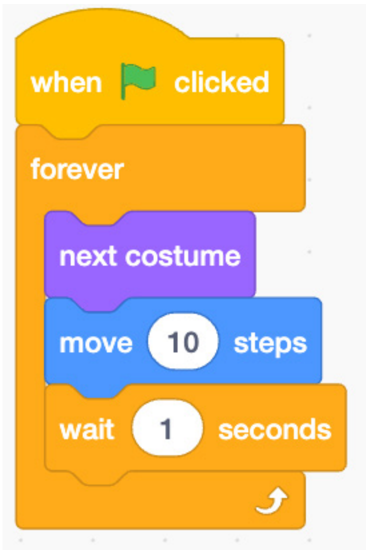
4.

Around the Block and Back Again

In the third assignment we will take what we have learned in previous projects and use them to develop a program that continues to run effectively and attractively without being reset every time we want it to run again. We will be using commands to set the location of the Sprite at the beginning of the program, as well as resetting it's position every time it reaches the edge of the screen so that the Sprite appears to run out of one side of the screen and return on the other.

In our last project, our Sprite crossed the screen and ran into the right wall of the frame. In order to keep this from happening, we will need to create a conditional statement that permits our Sprite to perform seperate action in case of certain parameters. In order to do this, we will repurpose our code from the last chapter and edit it to achieve our program's desired objectives.

If/then statements are a common type of conditional statement used to run a separate line of code if a certain condition is met (or unmet). By altering our code from the last section, we can create a program that does something else with our Sprite once it reaches the edge of the viewer panel by checking our Sprite's location as a condition. We will start by setting up our if/then statement, then you'll be left open to how you wish to code the end your program.



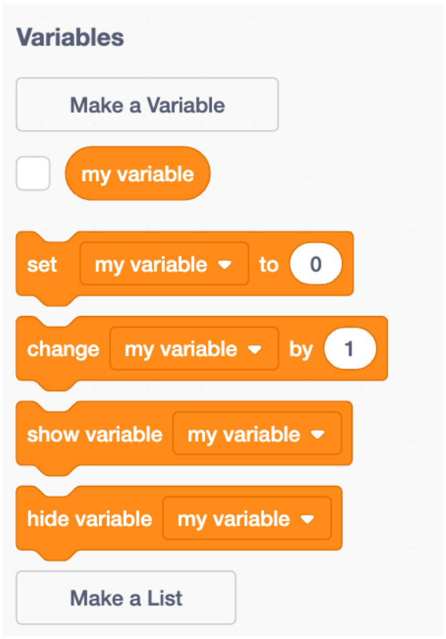
Starting with where we finished in the last assignment, we can look start to look at what needs to change in order to complete the current project. We want our Sprite to continue with the same actions, however, we do not want it to continue running into the right wall once it reaches the end of the viewer panel.

The first step is to create a conditional that checks whether or not the Sprite is running into the wall. For this objective, we will need the new if statement tile from the control section, and we will also be learning about how to use the green operator tiles and variables.

Operator tiles are used for things such as adding and multiplying numbers, comparing values, or picking random items from a list. We will use an operator in our if statement to check a number against the x-position of our Sprite to verify when our Sprite has reached the rightward bound of the viewer panel.

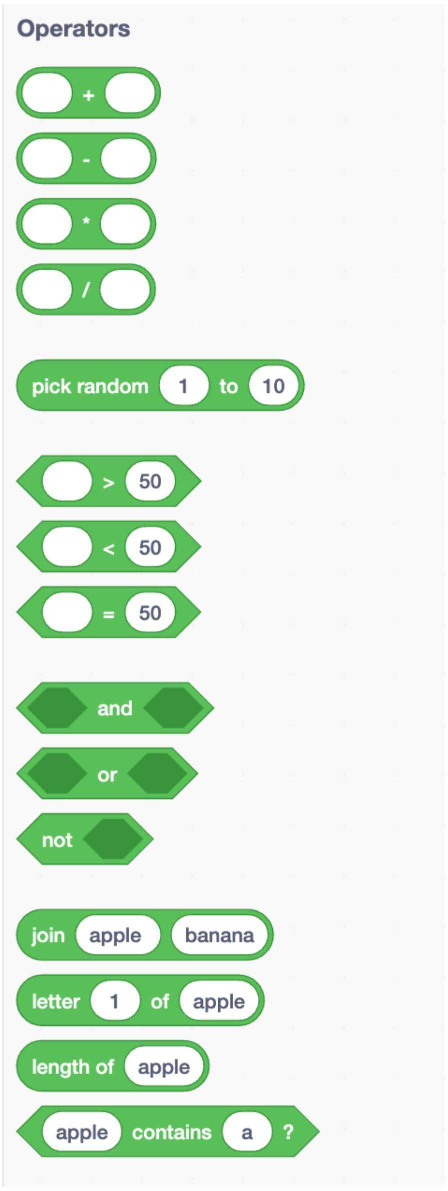
Variables

Variables are used as references for values in a program. Scratch includes some variables in other categories if they pertain to the tiles in that section, but in the dark orange Variables section you are given tools to alter or call custom variables. Calling a variable simply means referring to it to retrieve its value.

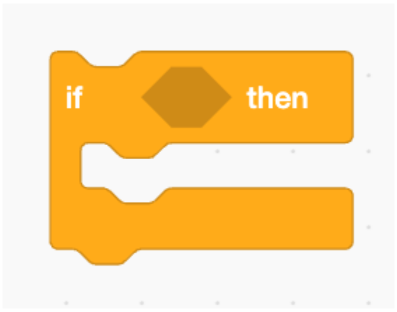


Operators

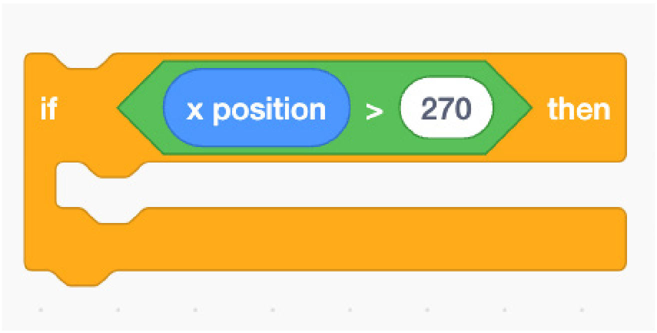
Operator tiles are used for things such as adding and multiplying numbers, comparing values, or picking random items from a list. We will use an operator in our if statement to check a number against the x-position of our Sprite to verify when our Sprite has reached the rightward bound of the viewer panel.



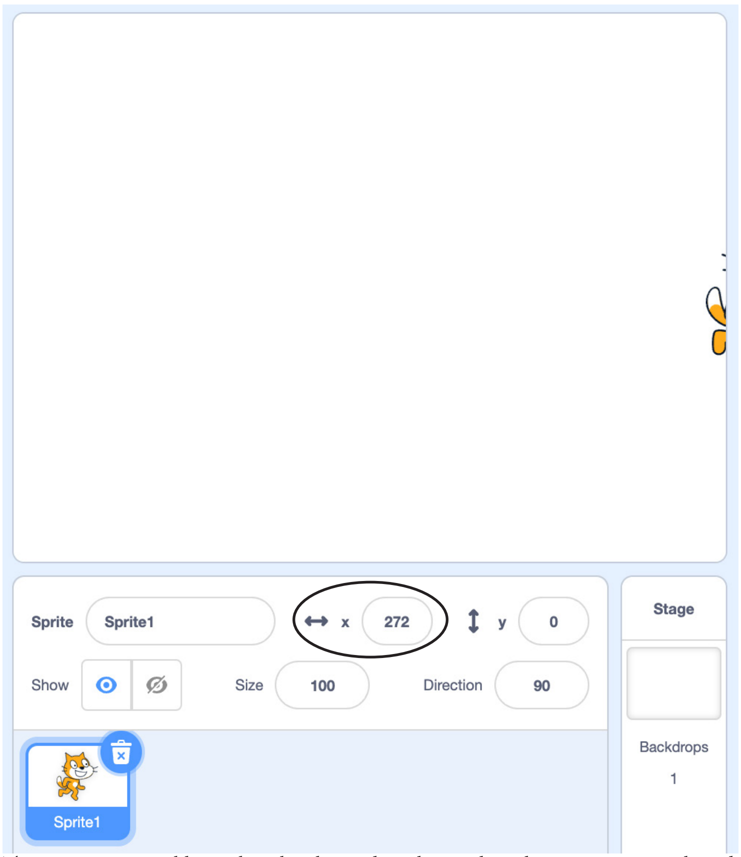
If statment tiles in Scratch come with a gap that fits an operator tile. In order for an if statement to work, there must be a condition to check it against. Not just any operator can be put in this spot, it must be checking for a result that will trigger the condition. The operator tiles have different shapes for this reason. The diamond shaped operator tiles compare two values and then determine if the condition is true.



The two values checked against each other must be of the same form. If we are comparing a text variable (known as a string) we will need to compare it to another body of text.



In our program, we will be using the greater than tile. This will allow us to check whether our x-position is greater than the rightside-limit of the viewer panel by comparing integers. By entering a number slightly below the rightside-limit we can in the right side of our equation, we can then put our x-position tile from the movements panel into the left. In this equation, we use a value slightly below our limit, because if we set this equation up to be“if x-position>272” our condition would never trigger, as our x-position could never be higher than our limit.


Sprite: Sprite1

x: 272

y: 0

Show

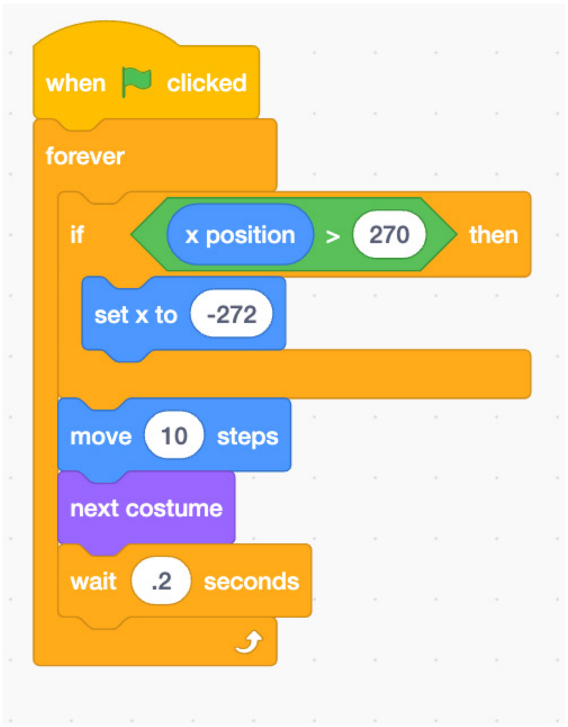
Size: 100

Direction: 90

The x-position variable can be taken by reading the x-value when our sprite is selected

We will want to run our conditional before we move the Sprite, because if our Sprite has reached it's bounds, the movement will not complete unless our x-position has been reset. In order to do this, we want to position our conditional statement in the forever loop before our existing code. This way, before our Sprite moves, it will reset it's position if it cannot move further towards the right!

Once you have your if statement in the correct place, set a separate path! You can use a couple different tiles to achieve the desired effect for this project, in the figure below, we used the set x tile. This allows us to immediately reposition our Sprite at the far left side of the viewer panel once our condition is met, and our Sprite can repeat their journey across the world! Other potential solutions include rotating the Sprite, reversing the move direction, or you could have used the go to (x,y) block, the potentials keep on going!



```
when [flag] clicked
forever
    if <(x position) > (270)> then
        set x to (-272)
    move (10) steps
    next costume
    wait (.2) seconds
```

Keep on giving this program a shot with the other solutions, and see if you can figure them out on your own.

Key Concepts

Block: a group of code that works together for a common end

Path: a block of code that occurs when a certain condition is true

Loops: a statement that groups together a block of code and allows for it to recur

Break: a line or block of code that orders a loop to end when a certain condition is met

To nest; nesting: to place a line or block of code inside a path of a controlling statement

Conditional statements: a type of statement that is used to branch code into different paths, allowing it to make different decisions based on the inputted value

Operations: mathematical and truth operations used to check values against each other or to change values

Variables: placeholders for a value that can be acted upon or referenced by a program