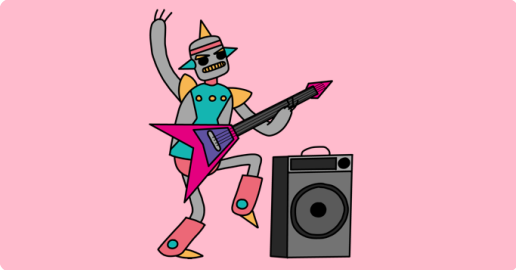
Binary hero

Score points by playing the notes of a song as they scroll past

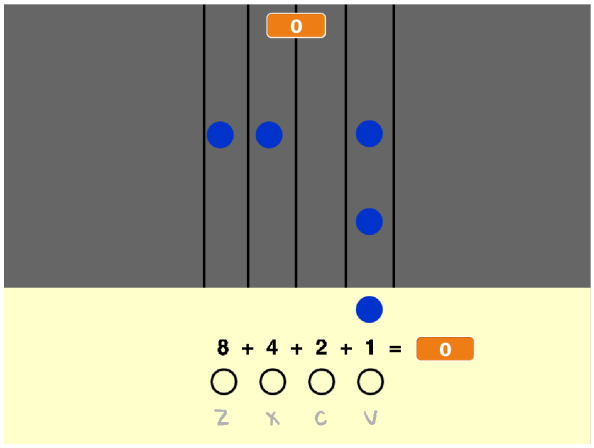


Step 1 Introduction

In this project you will make a game in which you play the notes of a song as they scroll down the Stage.

What you will make

The notes will fall from above, and you will have to press keys to “catch” and play the notes.



What you will learn

- How to use lists to store sequences of notes and timings
- How to use custom blocks with inputs

What you will need

Hardware

- A computer capable of running Scratch 3

Software

- Scratch 3 (either **online** (http://rpf.io/scratchon) or **offline** (http://rpf.io/scratchoff))

Downloads

- **Offline starter project** (http://rpf.io/p/en/binary-hero-go)

Additional notes for educators

You can find the **completed project here** (http://rpf.io/p/en/binary-hero-get).

Step 2 Key presses

How many notes can you play with four keys? It might be more than you think!

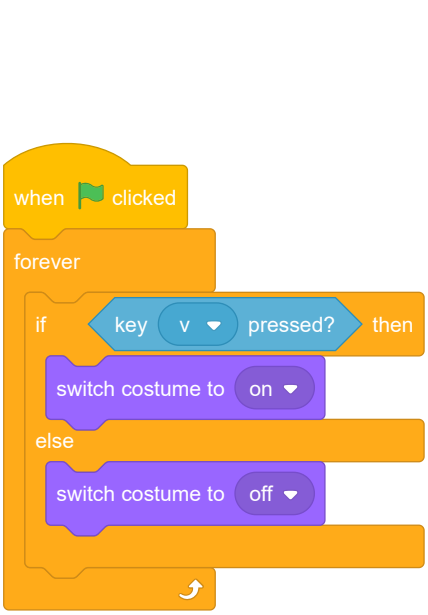
Open the 'Binary hero' Scratch starter project.

Online: open the starter project at **rpf.io/binary-hero-on** (http://rpf.io/binary-hero-on). If you have a Scratch account, you can click on **Remix** in the top right-hand corner to save a copy of the project.

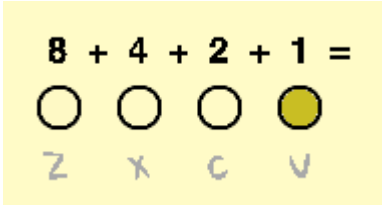
Offline: open the **starter project** (http://rpf.io/p/en/binary-hero-go)in the offline editor. If you need to download and install the Scratch offline editor, you can find it at **rpf.io/scratchoff** (http://rpf.io/scratchoff).

Start by showing which key is being pressed.

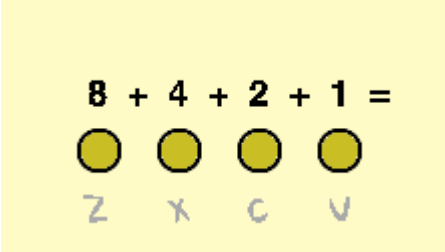
Click on the sprite called '1', and add code to change the sprite's costume if the `v` key is pressed.



When you test your code by pressing the `v` key, the sprite should light up.



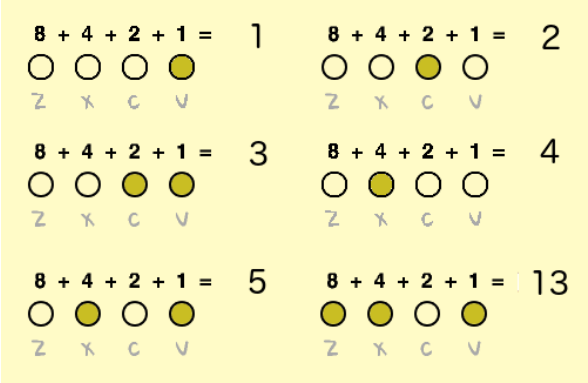
Do the same for the other three sprites so that they light up if the `z`, `x`, or `c` keys are pressed.



Step 3 Binary numbers

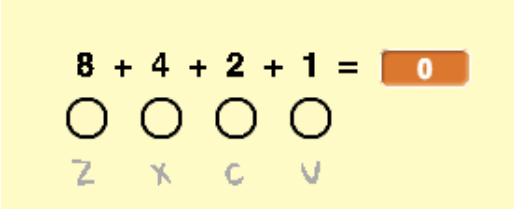
You will use different combinations of pressing the four keys to play different notes. Each of the keys is either on (pressed) or off (not pressed). This means that you can think of each combination of keys as a **binary number**.

Moving from right to left the keys double in value: `1`, `2`, `4`, and `8`. By adding up the numbers above the keys that are pressed, you can work out the value of the note.



There are 2^4 = **16 combinations** of pressing the four keys. This means that you can play 15 different notes, as `0` will mean that no note plays.

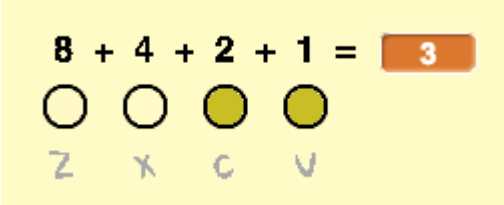
Create a new variable called `note`, and drag it next to the four note sprites.



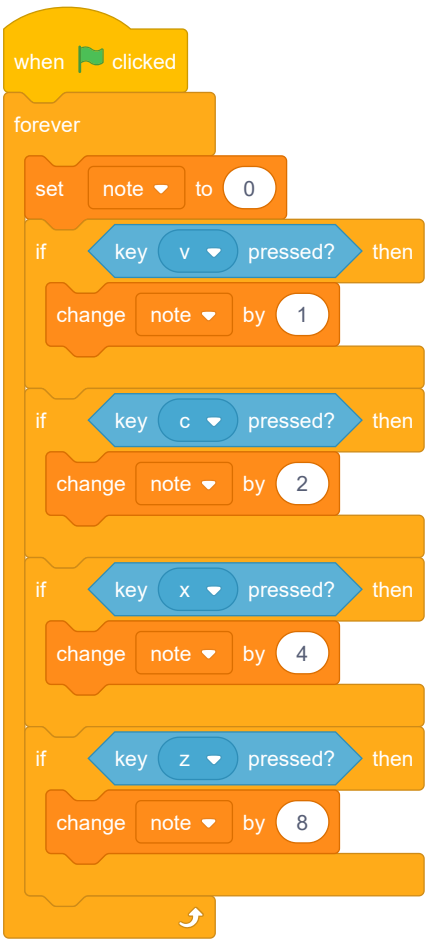
`note` will store the value of the note that should be played.

Add code to the Stage to use the combination of pressed keys to calculate the value of `note`.

For example, when `c` and `v` are pressed, the value of `note` should be `3`.



This is what your code should look like:

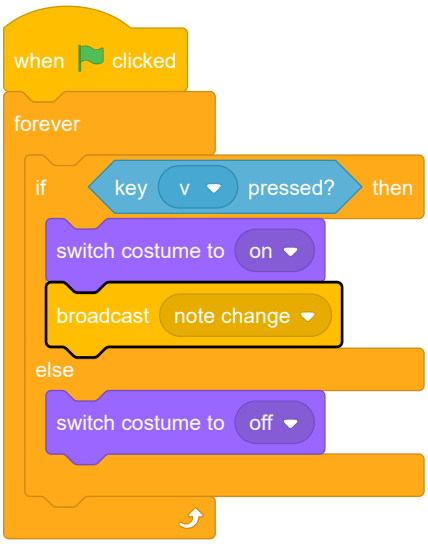


Step 4 Play notes

Play notes when the keys are pressed.

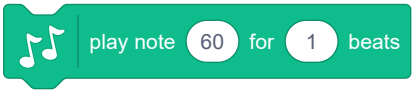
Add the Music extension to your project.

Broadcast a 'note change' message whenever **any of the four keys** is pressed.

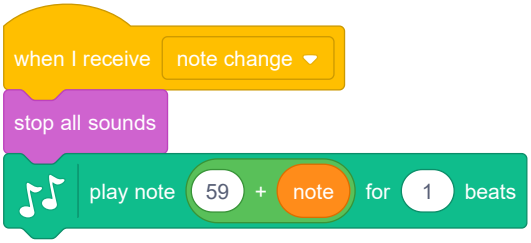


Add code to the Stage to play a note when a combination of keys is pressed.

Your notes should start at middle C, which is note 60.

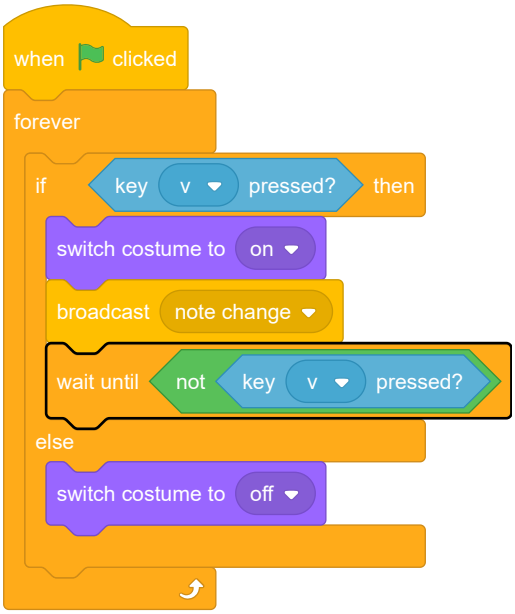


This is what your code should look like:



Test your code. Can you hear that a note is repeatedly played when you hold down a key?

Add code so that the **all** the key sprites only play a note **once** when a key is held down?



Step 5 Scrolling notes

You need to make notes scroll down the Stage so that the player knows which keys to press and when to press them.

Create two lists called `notes` and `times`.

Add the following numbers to your `notes` and `times` lists. Note: make sure to **add these exact numbers in the right order**.

Here's how songs are stored in your game:

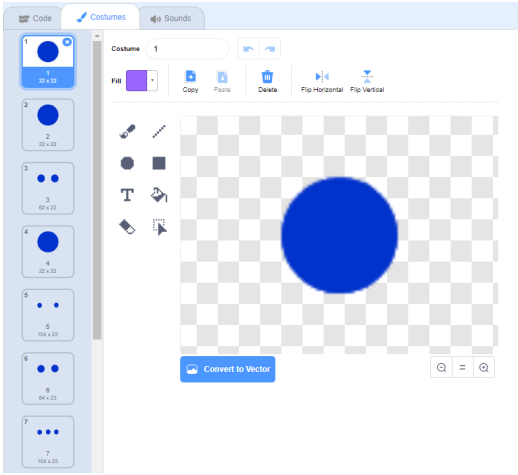
- The `notes` list stores the notes of the song (from 1 to 15), in order
- The `times` list stores the times when the notes should be played in the song

So with the two new lists:

- Note 1 (middle C) should be played at 5 seconds
- Note 1 should be played again at 5.5 seconds
- Note 3 should be played at 6 seconds
- etc…

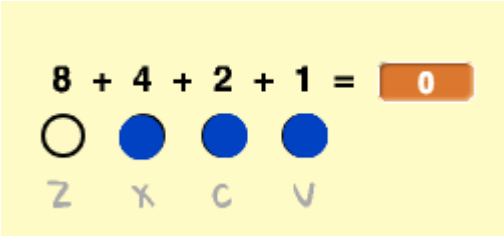
Click on the ‘note’ sprite and then click on **show**.

Then click on **Costumes**.

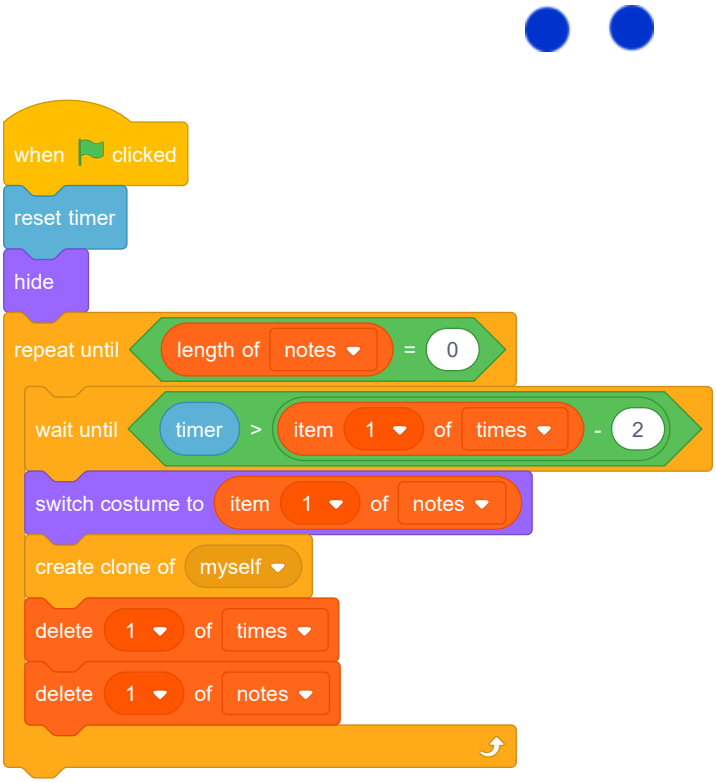


You should see that the ‘note’ sprite has 15 different costume, one for each different note from 1 to 15.

Add code to create a 'note' sprite clone for every note stored in `notes`. Each clone should be created at the correct time stored in `times`. Each clone should be created two seconds before its note needs to be played. This gives the clone two seconds to move down the screen. You'll create the code to move your clones in a little bit!

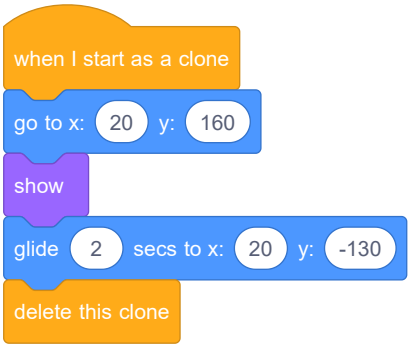


This is what your code should look like:



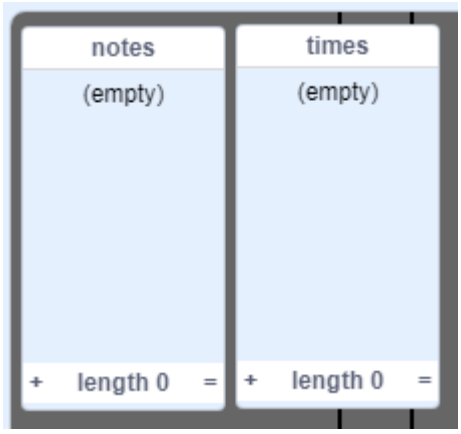
When you test your code now, nothing seems to happen, because the 'note' sprite is hidden. If you show (or don't hide) the sprite, then you should see clones being created on top of each other.

Add code to make each ‘note’ clone glide from the top to the bottom of the Stage before being deleted.

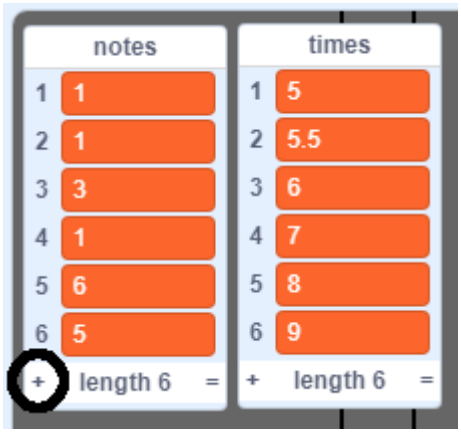


Step 6 Store your song

At the moment, notes are removed from the lists after being played, so you're left with empty lists:

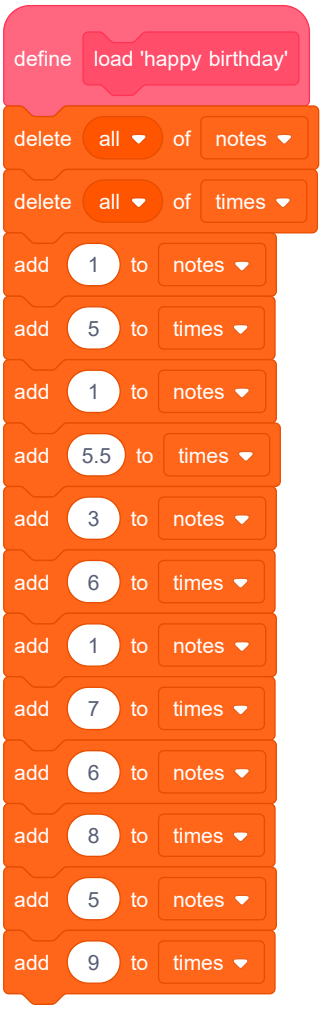


You're now going to add code to store songs in your project, so that you don't have to add to your lists each time.

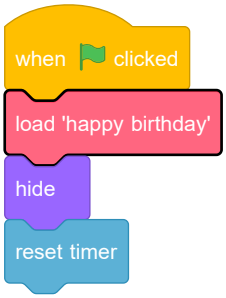


Make a new block called `load 'happy birthday'` that clears both the `notes` and `times` lists, and then adds the correct numbers back into both lists.

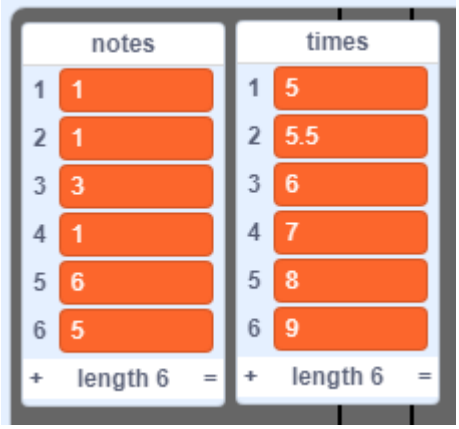
This is what your code should look like:



Test your new block by running it at the start of your project.



Each of your lists should now contain six numbers.

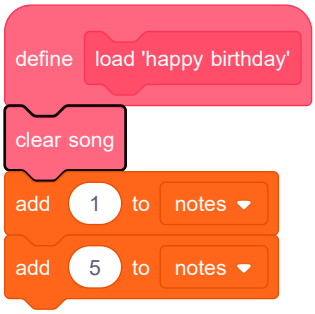


Step 7 More custom blocks

The newest section of code is difficult to read, so you're going to use more custom blocks to make it simpler.

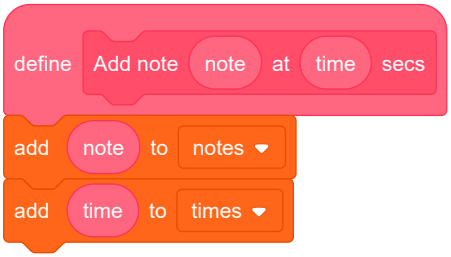
Make another block called `clear song` that deletes all items from both lists. Use this block before adding numbers back intothe lists.

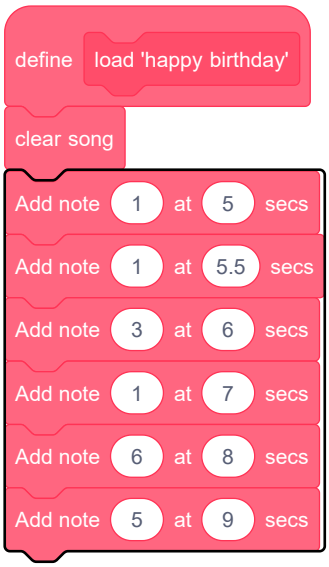
When you test your code, it should work just as it did before.



So that your code is even easier to read, make another block that allows you to specify a note to be played and a time to play the note at.

This is what your code should look like:

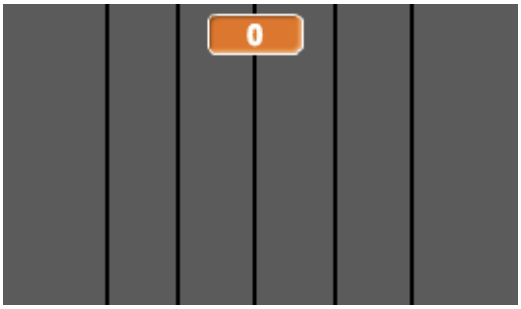




Step 8 Keep a score

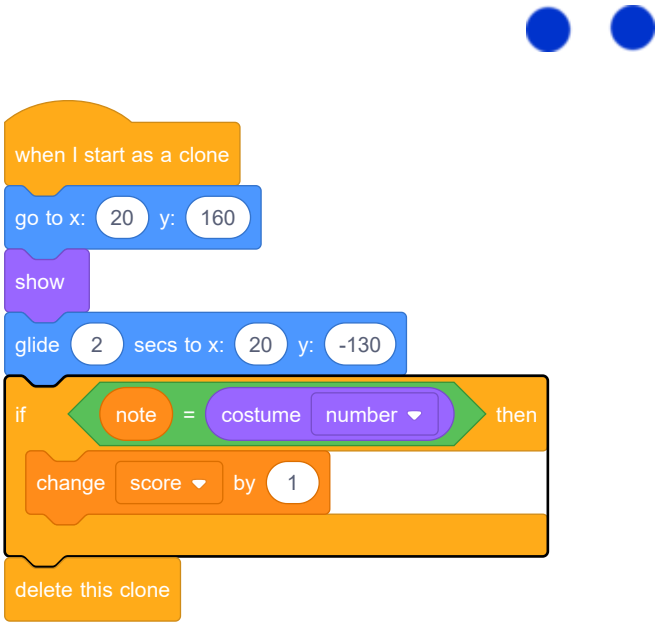
Improve your game by giving the player points for playing the correct note.

Create a new variable called `score`, and place it at the top of your Stage.

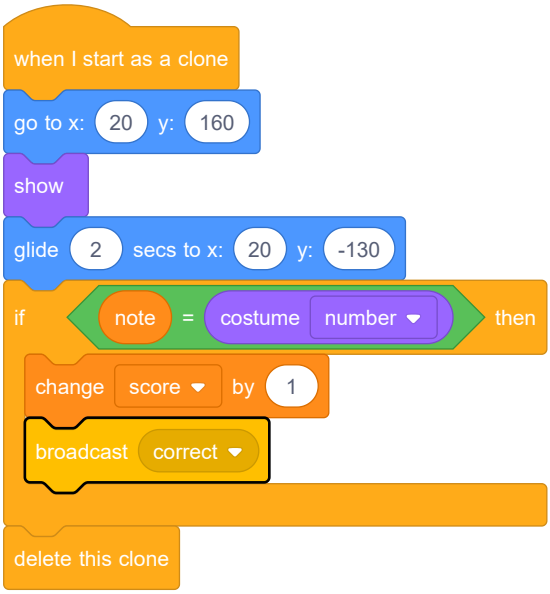


Add to `score` whenever the player plays the correct note at the correct time. Remember to set `score` to `0` at the start of the game.

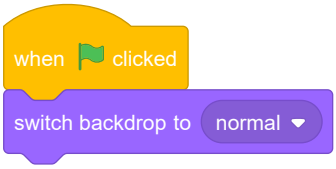
This is what your code should look like:

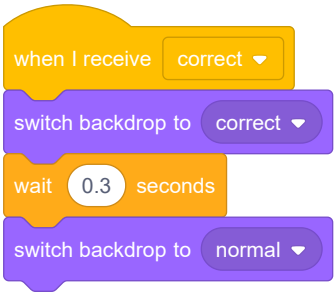


Broadcast a message called ‘correct’ when the correct note is played.



Add code to your Stage to briefly change the backdrop when the player plays the correct note. The project already contains a second backdrop for this.



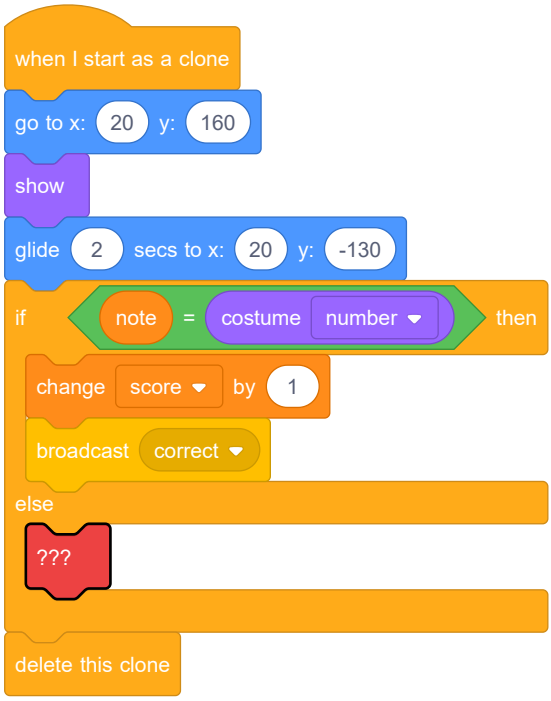


Challenge!

Challenge: take it further

Your game is done now, but there are a few things you can do to make it even better if you want to!

For example, can you add code to change how the Stage looks if the correct note is not played?



To do this, you need to add code that's very similar to the code that changes the backdrop when the correct note is played. The project contains another backdrop you can use.

Step 9 What next?

Try these other projects to build you knowledge of other programming languages.

- **About me** (https://projects.raspberrypi.org/en/projects/about-me?utm_source=pathway&utm_medium=whatnext&utm_campaign=projects) is a great introduction to Python.
- **Happy birthday** (https://projects.raspberrypi.org/en/projects/happy-birthday?utm_source=pathway&utm_medium=whatnext&utm_campaign=projects) introduces HTML and CSS.
