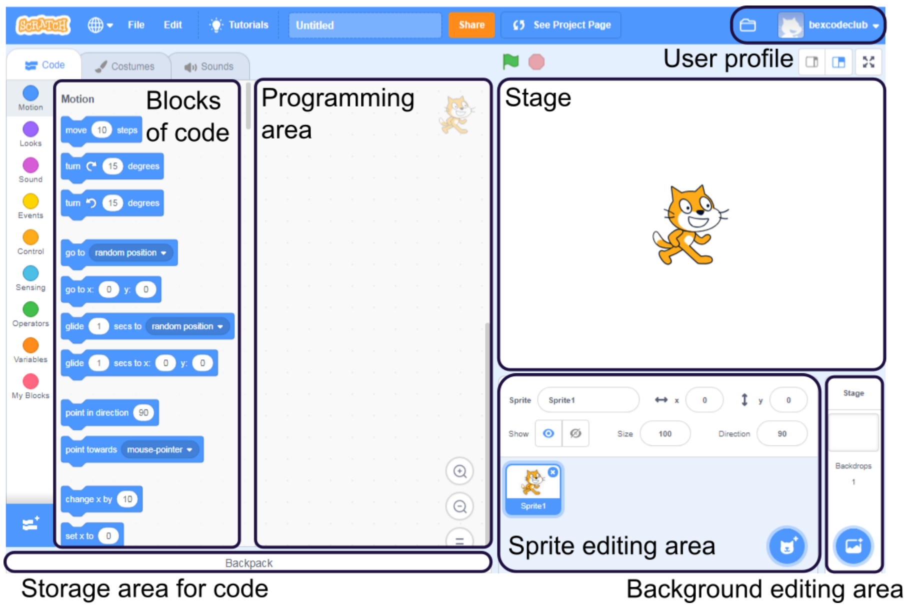
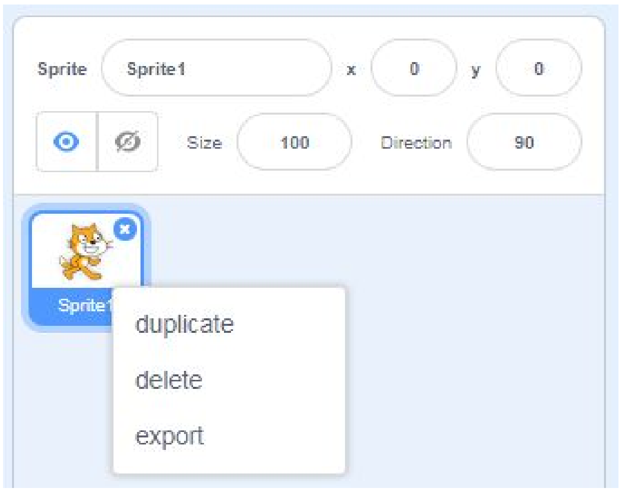
Matariki

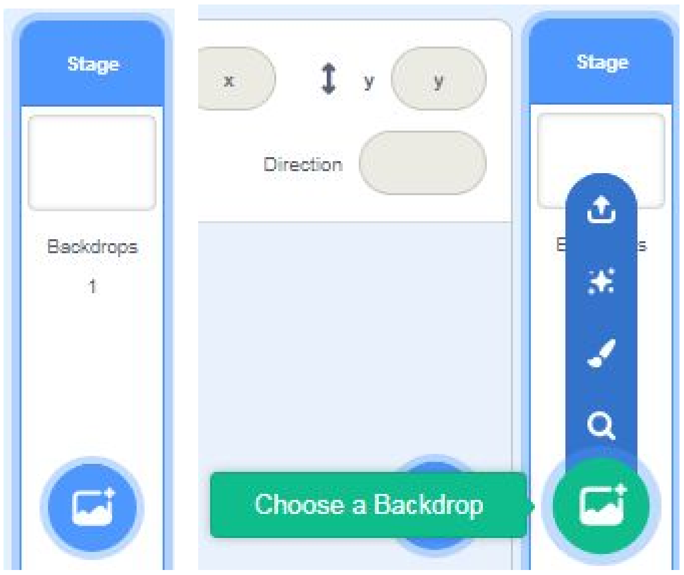
Session One

Head to scratch.mit.edu and create a new project. The cat is called a ‘sprite’. This is a little character that we will will be able to control with our coding. The coding blocks are in the centre and they can be dragged and dropped into the grey area.



The first thing we want to do is change the sprite. We do this by right clicking and selecting delete.

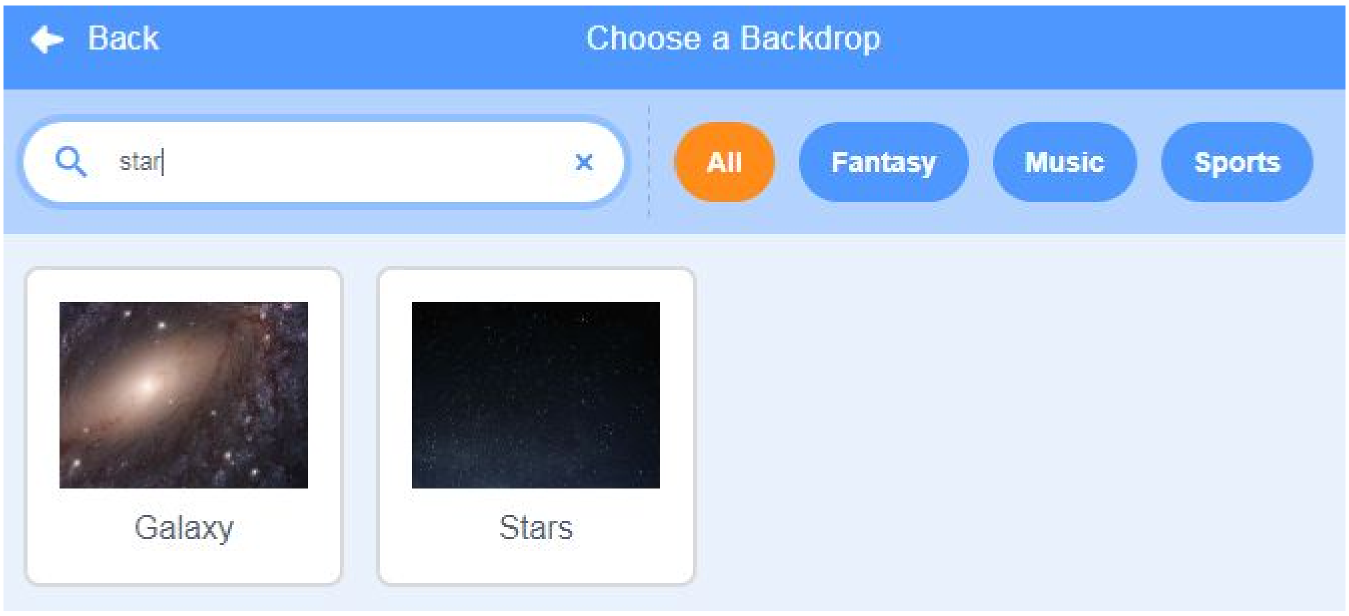




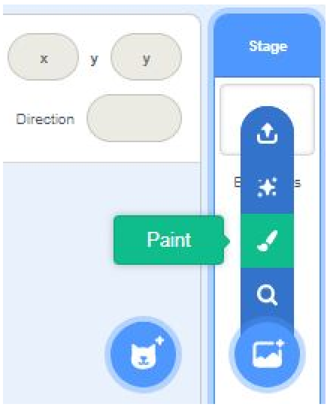
We need to add a starry backdrop for our Matariki stars. We can:

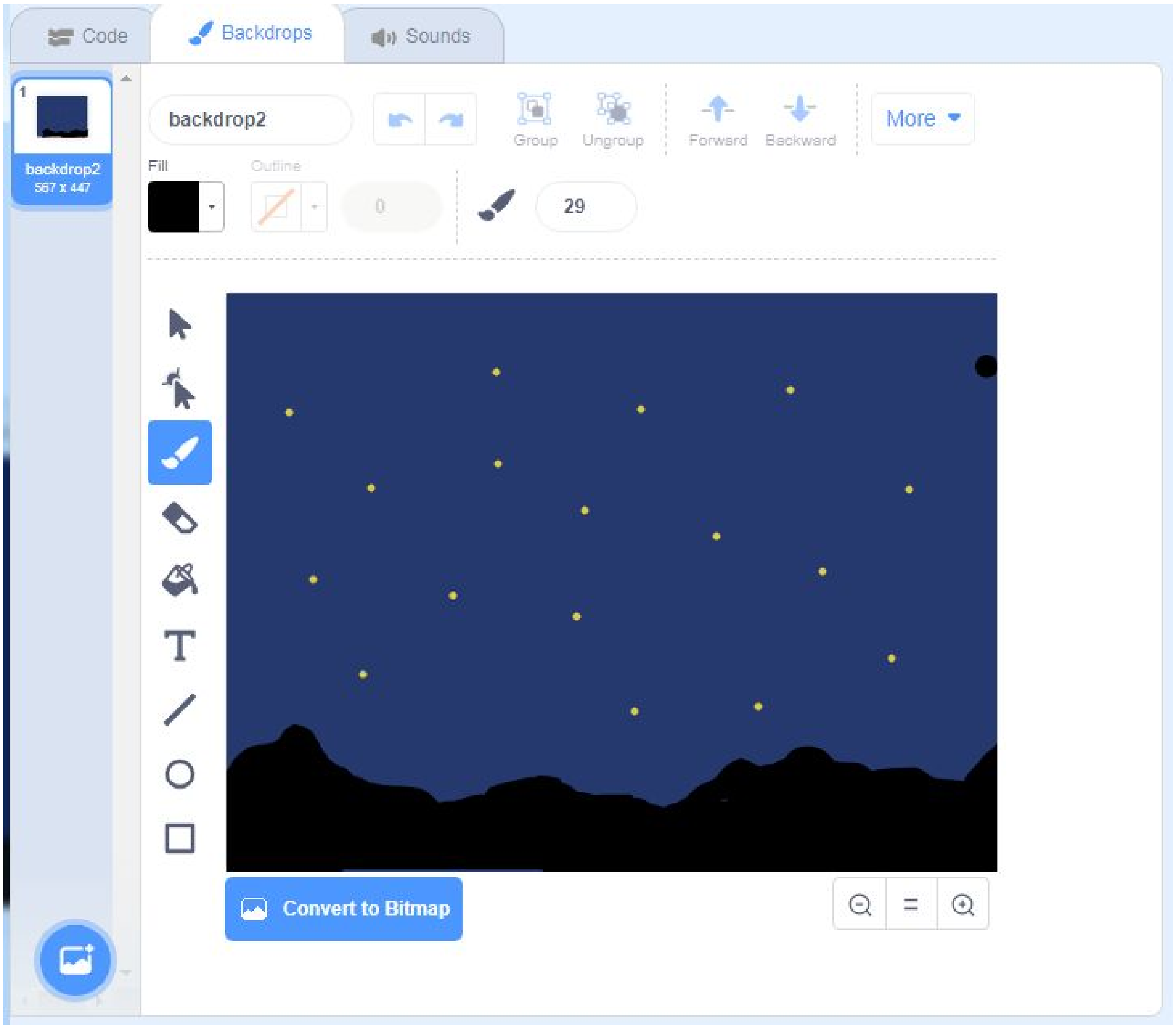
- Use a preset background from Scratch
- Draw a background in Scratch
- Uploading an image from your computer

Using a backdrop from Scratch



Drawing a backdrop in Scratch

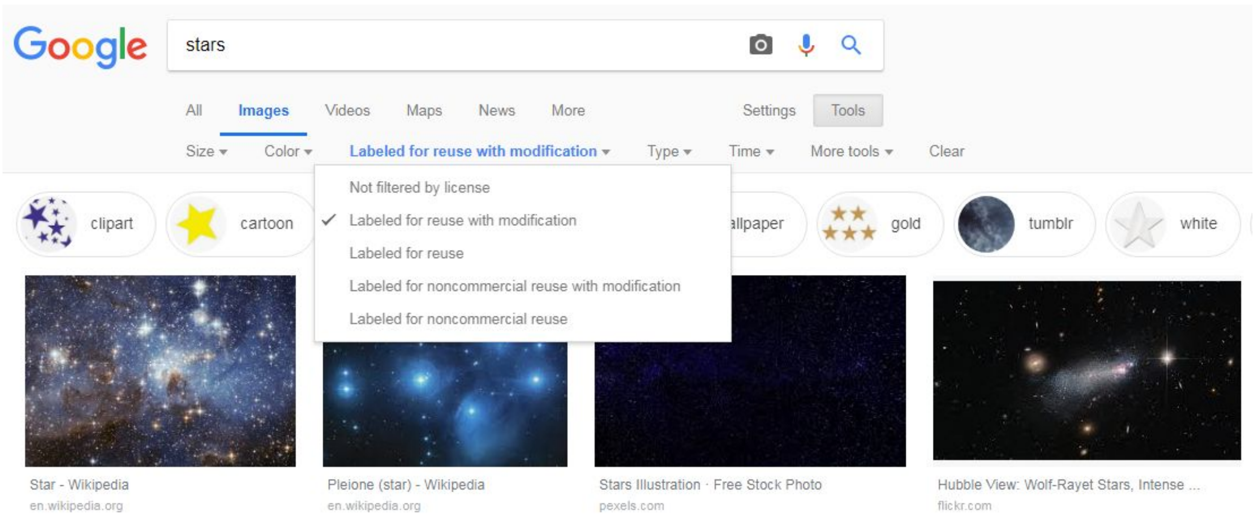




Uploading from your computer

We can search for a suitable backdrop online. When using images from the internet, it is important to check the usage rights of the image. Some images are copyrighted and cannot be used without the owner's permission. Think about how you would feel if someone took an image that you created and used it without asking you?

Some search engines have tools that let us search for images by their usage rights. It is important to always check the usage rights by clicking through to the website.

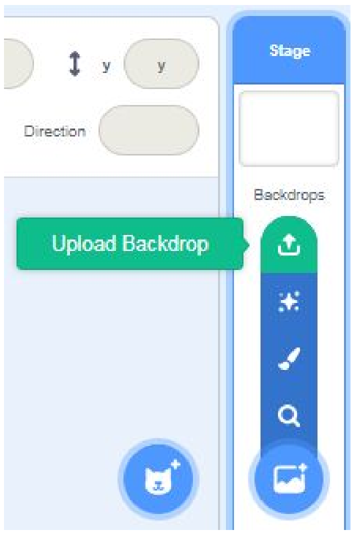

Google
stars
All
Images
Videos
Maps
News
More
Settings
Tools
Size
Color
Labeled for reuse with modification
Type
Time
More tools
Clear
Not filtered by license
Labeled for reuse with modification
Labeled for reuse
Labeled for noncommercial reuse with modification
Labeled for noncommercial reuse
clipart
cartoon
allpaper
gold
tumblr
white
Star - Wikipedia
en.wikipedia.org
Pleione (star) - Wikipedia
en.wikipedia.org
Stars Illustration · Free Stock Photo
pexels.com
Hubble View: Wolf-Rayet Stars, Intense ...
flickr.com

Alternatively, there are websites that are provide copyright-free images for use in education.

- Wikicommons has a mix of copyrighted and copyright free images
- NASA allows all of it's photographs to be used for free for educational purposes
- Unsplash.com and pixabay.com are both search engines for free to use images

Once you have found a suitable picture, save the image to your computer (for most machines, right click then click save as). The image should appear in your computer's downloads folder or where ever you have saved it.

To upload it to Scratch, click on the upload icon and navigate to the folder where the image saved. Find the image and click open. You can edit your image using the paint tools in Scratch.

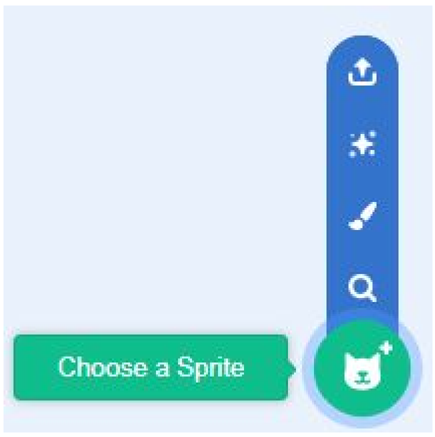


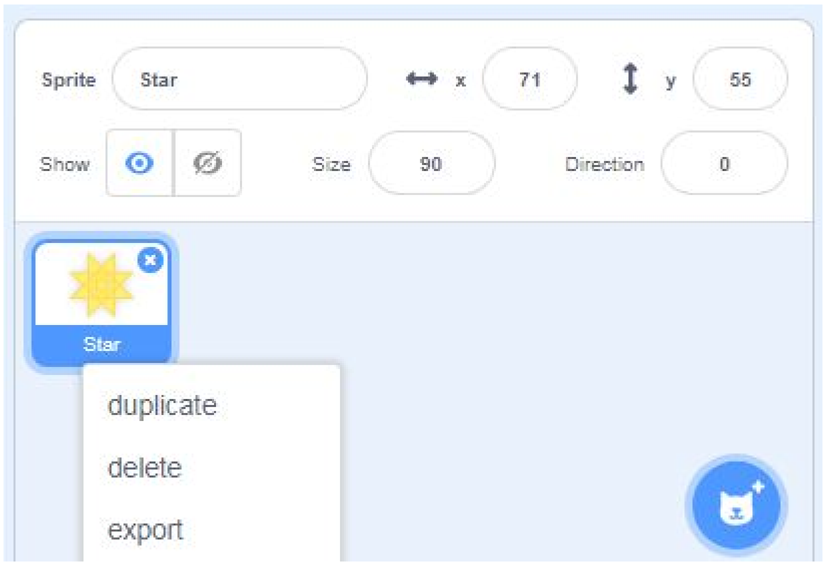
Creating Sprites

As with the backdrops, there are three main ways to get a sprite:

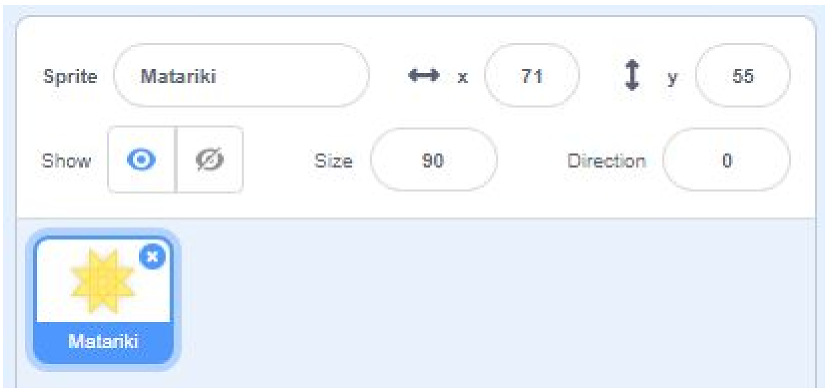
- Choose one from the library
- Draw one in Scratch
- Upload an image

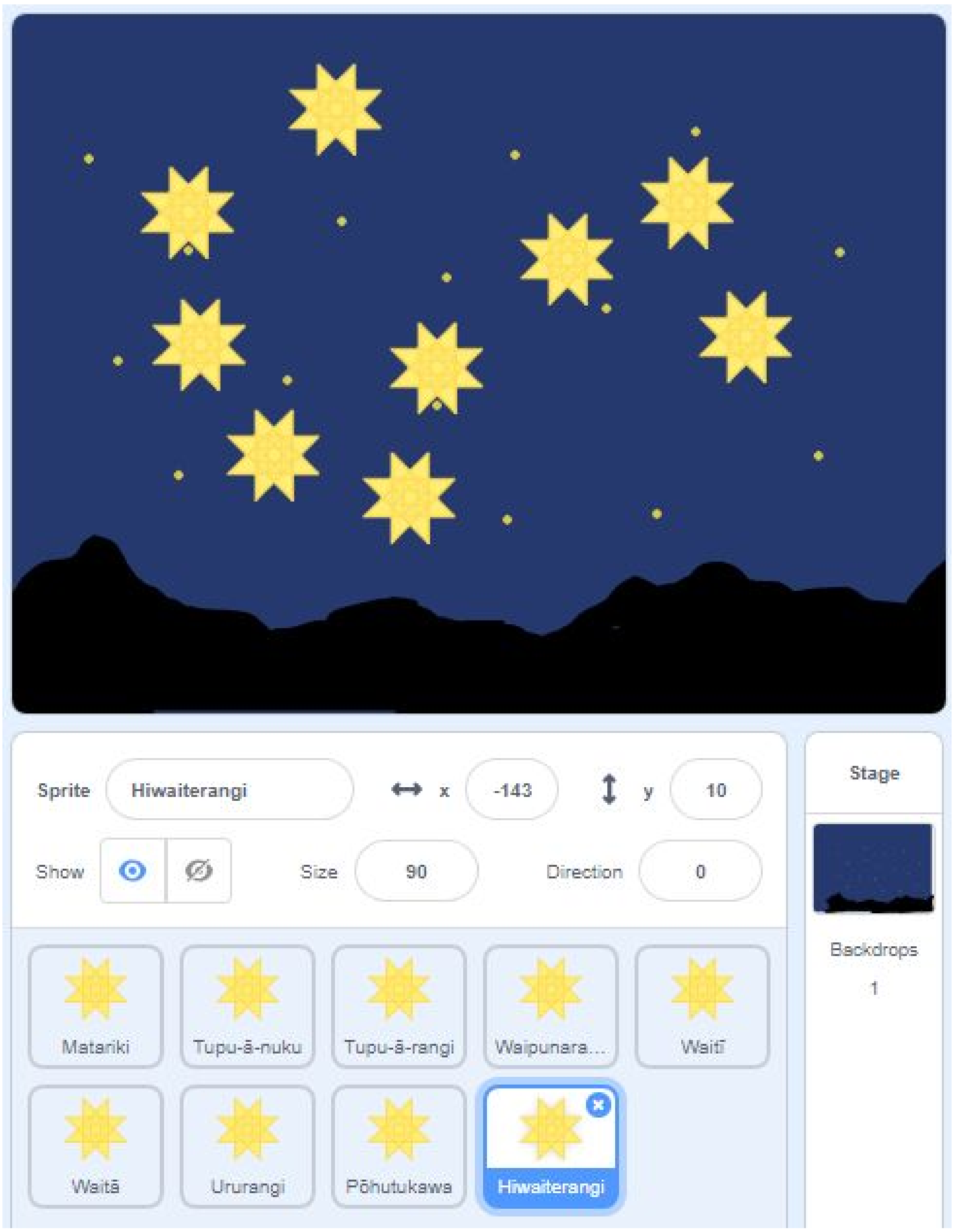
Once you have create your star sprite, you can duplicate it by right clicking and selecting duplicate. As there are nine main stars in the Matariki cluster, create nine star sprites.





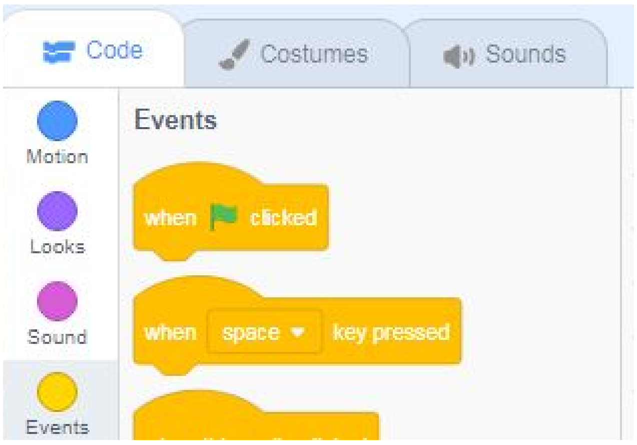
Rename your star sprites to the names of the stars and rearrange the sprites to form the constellation on your starry backdrop.



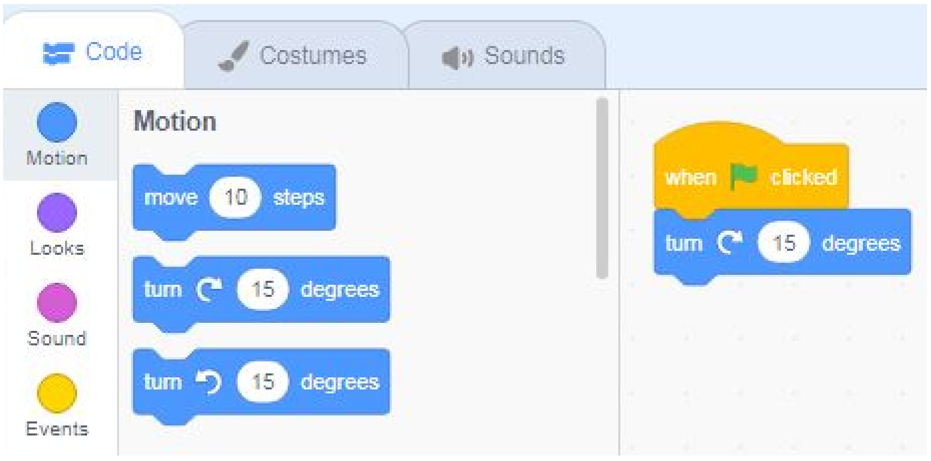


We're now ready to start programming! We're going to write a program that makes the stars twinkle. Click on the star you want to start with. We need to use a block to signify when our program will start. These can be found in **Events**.

From **Events**, find the 'when flag clicked' block. Drag and drop the block into the programming area.

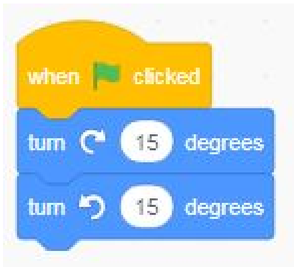


Now head to **Motion** and find a ‘turn right __ degrees’ block. Drag and drop the block under the ‘when flag clicked’ block. A white line will appear under the ‘when this sprite clicked’ block when the ‘turn right __ degrees’ block is close enough to snap together.



We’ve now written our first program! Click on the green flag to test your program out. You should see one of your stars turn a little.

We make the star turn back and forth by using a second block. Drag a ‘turn left __ degrees’ block and attach it under the ‘turn right __ degrees’ block.



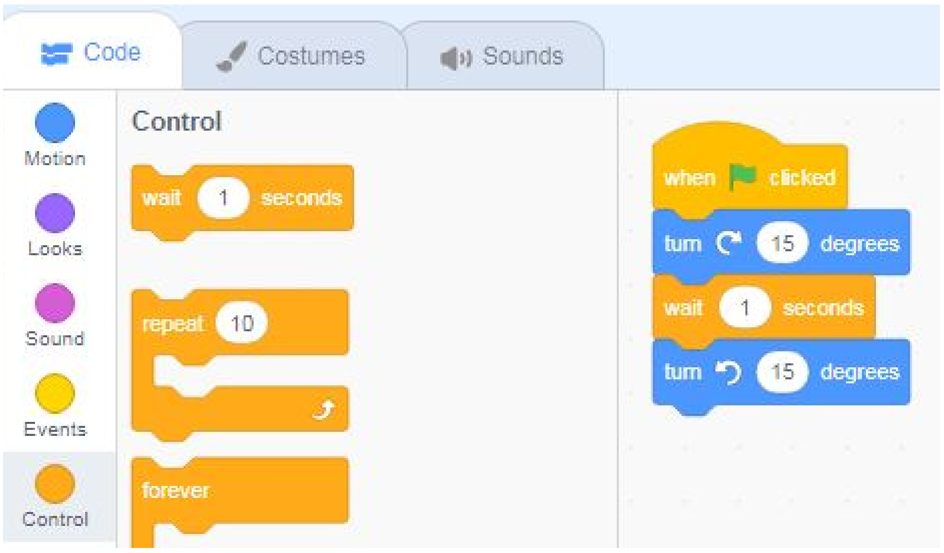
What happens now when you click the green flag?

Debugging

When a program doesn't do what we wanted it to do, we need to debug it. We do this by asking three questions:

- What did we want the program to do?
- What did the program do?
- What did we tell the program to do?

Computers do exactly what we tell them to do, and they often do it very fast. In this case, Scratch will be rotating the star to the right and then to the left, however it will be doing it faster than we can process! We can slow it down by using a 'wait' block from **Control**.



Sequence

As Scratch can only do exactly what we have told it to do, we need to make sure our instructions are in the correct order. Try changing the order of your blocks to see how it affects the sprite's movement.

Repetition

Sometimes we want to repeat a set of instructions. For example, we might want the star to keep rotating one way and then the other for a few seconds. We could do this by adding extra 'turn right' and 'turn left' blocks, but this would quickly get time consuming and messy.

Luckily, the computer has an easy way to repeat sections. In **Control**, there is a block called 'repeat'. Other blocks can be put inside this

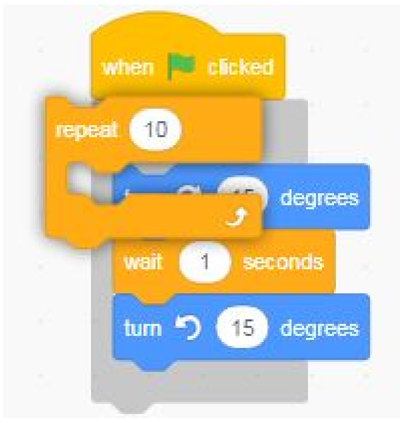
To delete blocks of code in Scratch, you can either right click on the block you want to remove and click delete, or drag and drop the blocks back into the centre code bar.

Our loop contains a sequence of three instructions. Loops usually contain sequences and sequences can also contain loops!

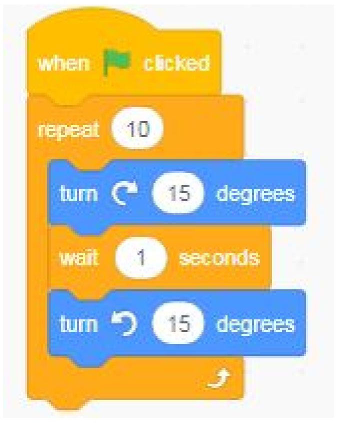
block, and will repeat the instructions for the number of time in the white area. This is called a loop.

A loop is a section of instructions that are repeated until they are told to stop. The signal to stop can vary, for example if something has repeated a certain number of time or if a certain condition has been met.

From **Control**, find the 'repeat __' block. Drag it to you program. The loop will automatically expand to contain your existing code.



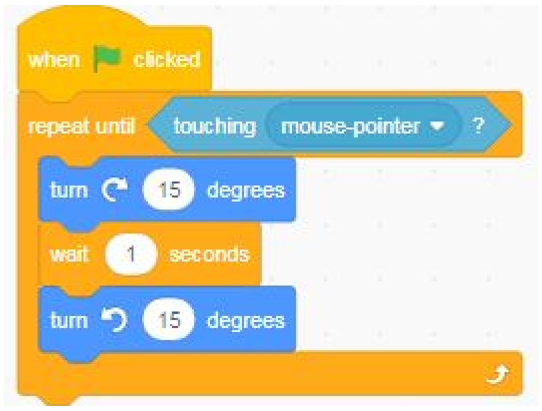
```
when 🏳 clicked
repeat 10
degrees
wait 1 seconds
turn ⟲ 15 degrees
```



```
when 🏳 clicked
repeat 10
    turn ⟳ 15 degrees
    wait 1 seconds
    turn ⟲ 15 degrees
```

Extra: The star's movement is not very smooth. Can you add a second wait block to improve the look of the movement?

The 'repeat __' block is great if we want our program to repeat the code for a set number of times. Sometimes we want to be able to give the program a signal that tells it to stop instead. The 'repeat until' block requires us to provide an extra piece of information, which is when we want it to stop repeating. We can tell Scratch to continue repeating the instruction until we give it a signal by going into **Sensing** and selecting the 'touching ____?' block. Drag the 'touching ____?' block into the space in the 'repeat until ____' block. Click on the small downwards arrow to open the drop down menu and select mouse-pointer.



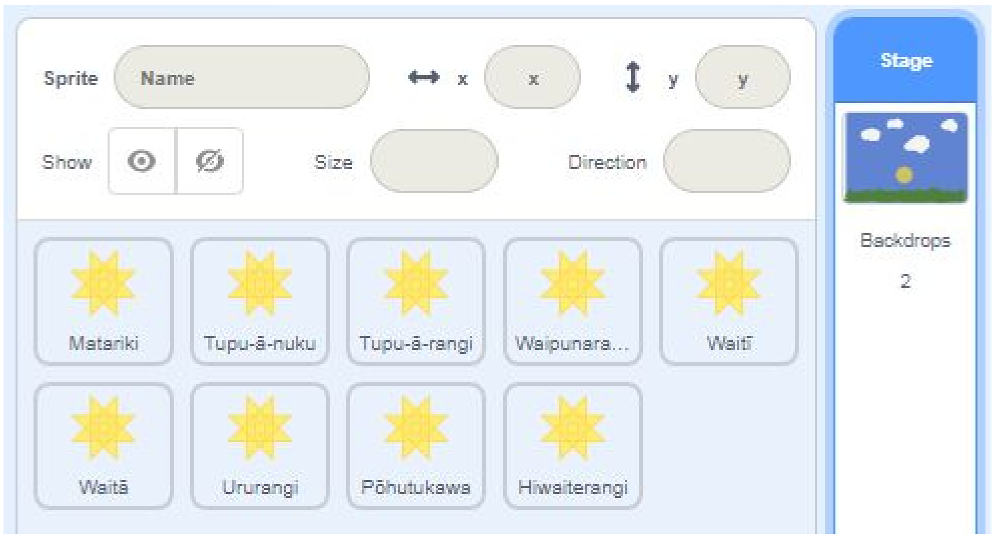
```
when 🏳 clicked
repeat until touching mouse-pointer ▾ ?
    turn ⟳ 15 degrees
    wait 1 seconds
    turn ⟲ 15 degrees
```

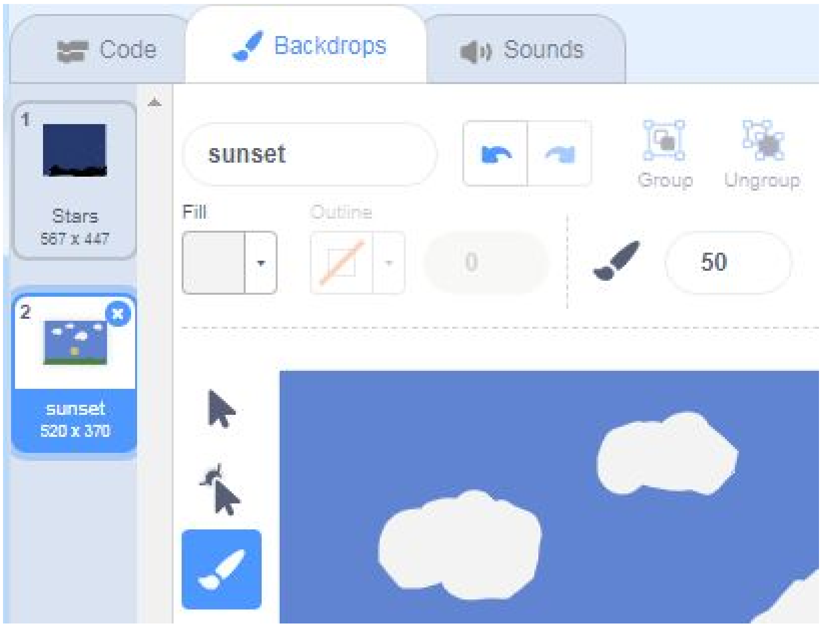
Extra: Explore the blocks of code in motion and looks. Can you make each star twinkle in a different way?

Session Two

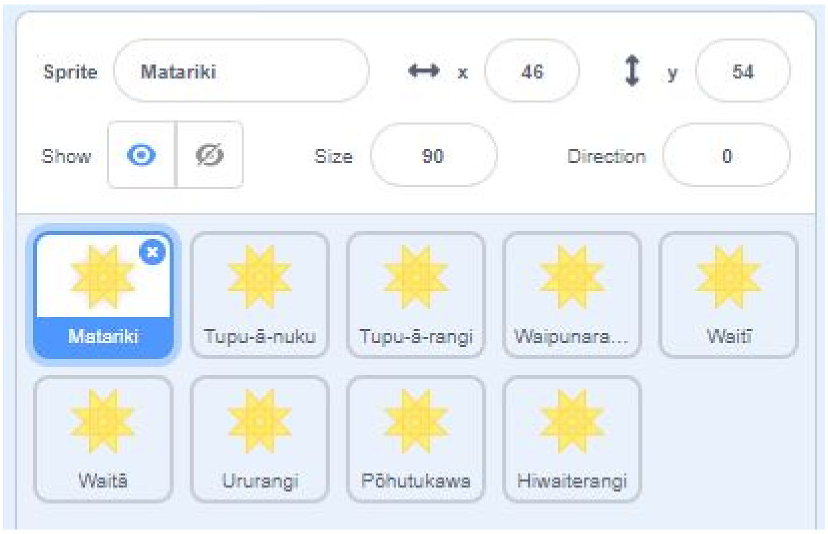
Broadcasting

We want to start telling a story using our Scratch project. Before we start telling the story, we are going to change the backdrop and hide the stars. First we want to create or upload a second backdrop that looks like twilight or sunset.

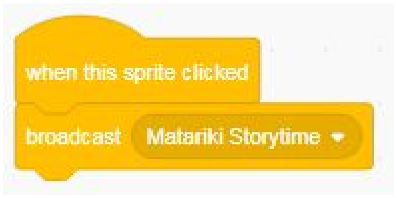




We want to be able to send a message between the sprites and backdrops. In Scratch, we can broadcast a message to all the sprites and backgrounds in use and can be used as a signal for something to start or stop happening. To use broadcasting, we need something to trigger sending a message. We will use Matariki being clicked to signal sending the message. Click on Matariki in your sprite menu.

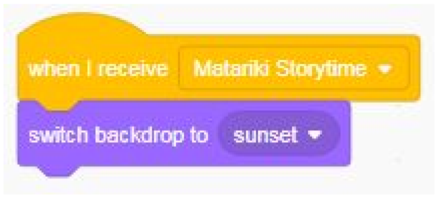


From **Events**, find the 'when this sprite clicked' block and the 'broadcast _____' block and snap them together in the programming area. Click on the drop down arrow and select new message. Choose a name for your message. You can pick any name and naming the messages xyz will work, but it could make it difficult for someone else to read and use your code.

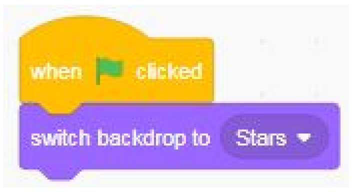


```
when this sprite clicked
broadcast [Matariki Storytime ▾]
```

Now click on the backdrop. We can give the backdrop code in the same way that we give the sprites code. From **Events**, find the 'when I receive ___' block and drag it into the programming area. From **Looks**, use a 'switch backdrop to ___'. We also want to be able to put the backdrop back to the star scene, so find a 'when flag clicked' block and another 'switch backdrop to ___' block.

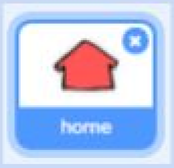


```
when I receive [Matariki Storytime ▾]
switch backdrop to [sunset ▾]
```



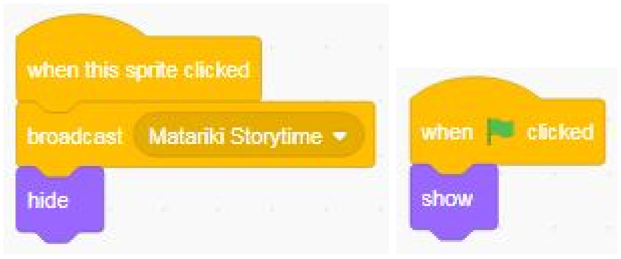
```
when ⚑ clicked
switch backdrop to [Stars ▾]
```

Extra: can you add another Sprite to make a home button that takes you back to the night sky when you click on it?



Backpack

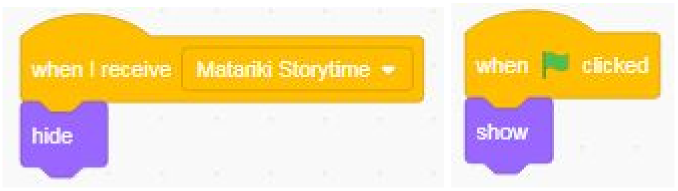
Now that we are able to change the backdrop, we also want to hide the stars. Add a 'hide' block under the broadcast block in the Matariki sprite's programming area. We want the star to reappear when we go back to the starry night sky, so add a 'show' under a 'when flag clicked' block.



```
when this sprite clicked
broadcast [Matariki Storytime ▾]
hide
```

```
when 🏴 clicked
show
```

We need to hide all nine of the stars individually. Click on the next star and add a 'when _____ received' block followed by a 'hide' block, and a 'when flag clicked' followed by a show block.

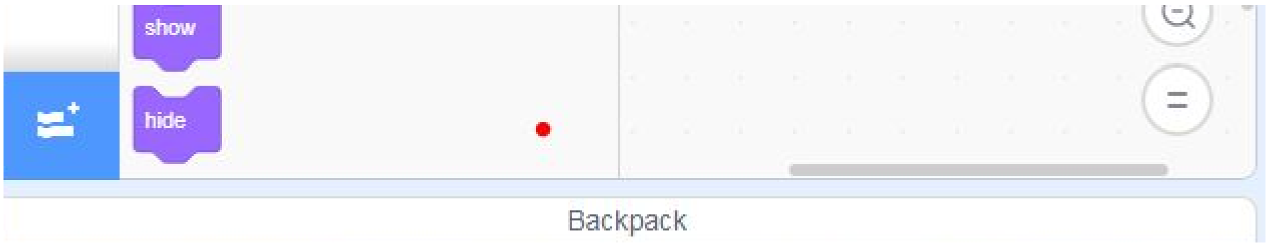


```
when I receive [Matariki Storytime ▾]
hide
```

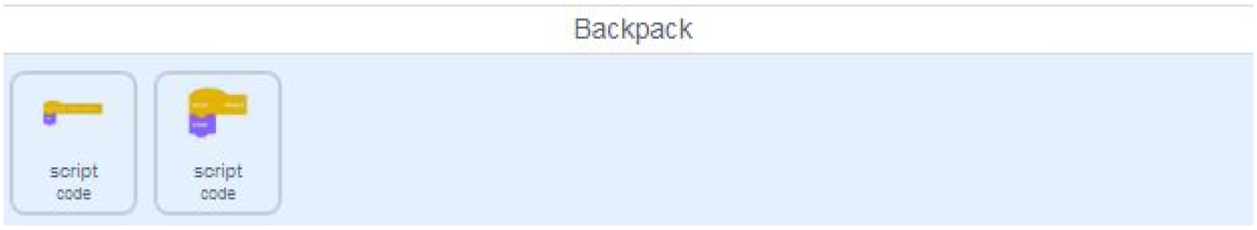
```
when 🏴 clicked
show
```

We could go through and recreate these two blocks of code, or we could use the Backpack feature to copy the code into the other areas.

Click on the arrow at the bottom of the programming are to open the Backpack.



Drag and drop the whole code block you want to copy into the backpack (don't worry, it will reappear in your programming area!).

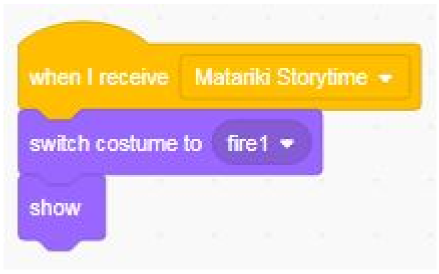


Click on the sprite you want to copy to the code to and drag the block of code out into the programming area. If you are signed in to Scratch, any code in your backpack will be accessible to any Sprite in any program you have written or are remixing. To remove move something from your backpack, right-click on the block and click delete.

Copy the two blocks of code into the programming areas of every star sprite so that all the stars disappear when the Matariki sprite is clicked and reappear when the green flag is clicked.

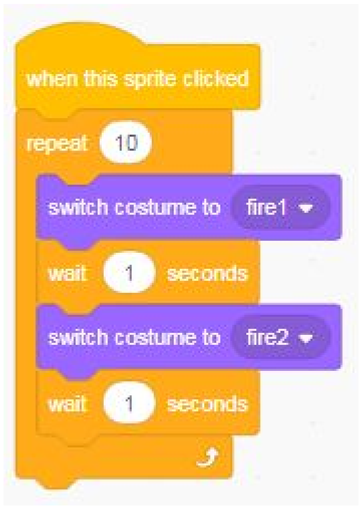
In our blank backdrop, we are going to create a Matariki celebration scene.

Create a new sprite by uploading these pictures of a campfire or drawing your own, fire1, fire2, fire3. We want our campfire sprite to appear when the Matariki storytime message is broadcast. Make sure that the fire isn't lit when it first appears!



```
when I receive [Matariki Storytime ▼]
switch costume to [fire1 ▼]
show
```

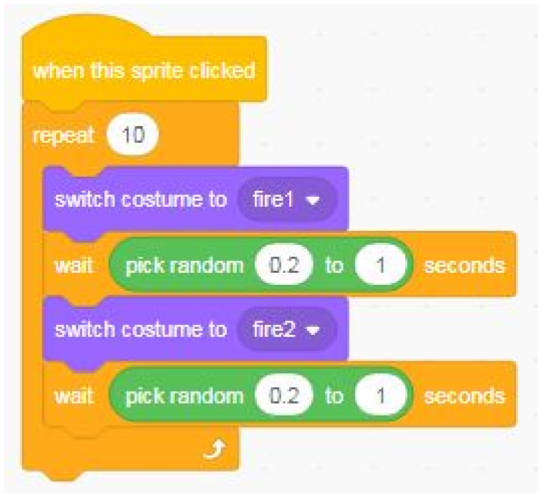
We want the fire to light and flicker between costumes when we click the sprite.



```
when this sprite clicked
repeat (10)
    switch costume to [fire1 ▼]
    wait (1) seconds
    switch costume to [fire2 ▼]
    wait (1) seconds
```

Random number generators

It is possible to ask the computer to choose a random number for you. We can use a random number generator to make the fire flickering more realistic. In Operations, find the 'pick random __ to __' block and drag it into the number space in the 'wait' blocks. Change the numbers to reflect the fastest and slowest that you would want your fire to flicker.



```
when this sprite clicked
repeat 10
    switch costume to fire1
    wait pick random 0.2 to 1 seconds
    switch costume to fire2
    wait pick random 0.2 to 1 seconds
```

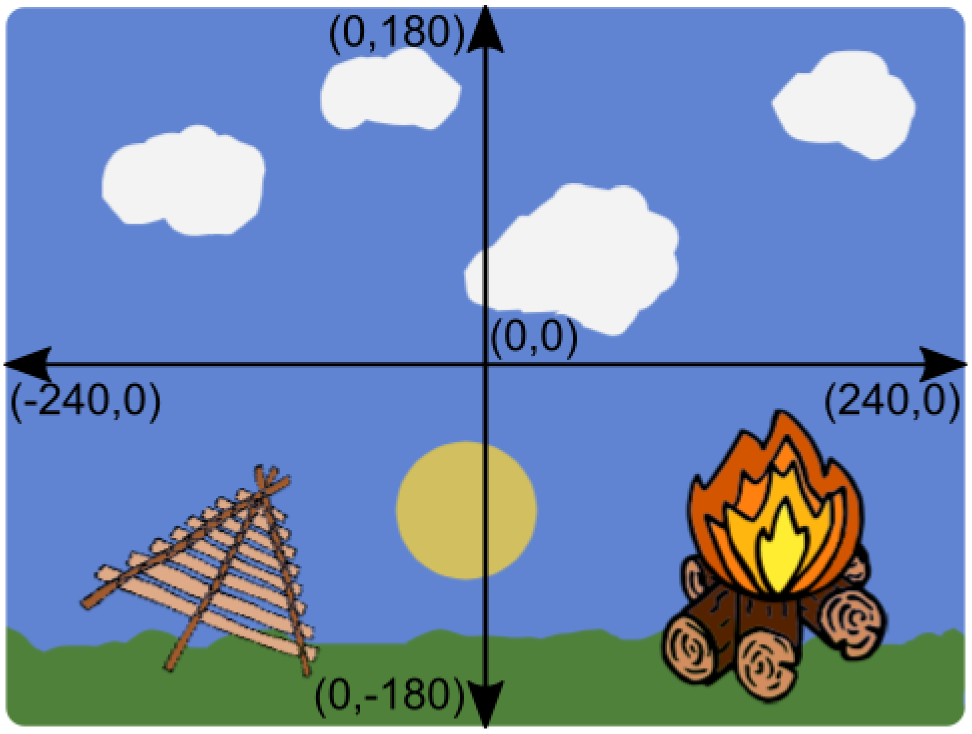
Move your campfire sprite to the bottom corner of your stage and upload this kite sprite to your program.

We are going to get the kite to fly by exploring the coordinates of the stage area. In Motion, there are two ways we can make a sprite move across the screen, 'move __ steps' and 'change x by __'.

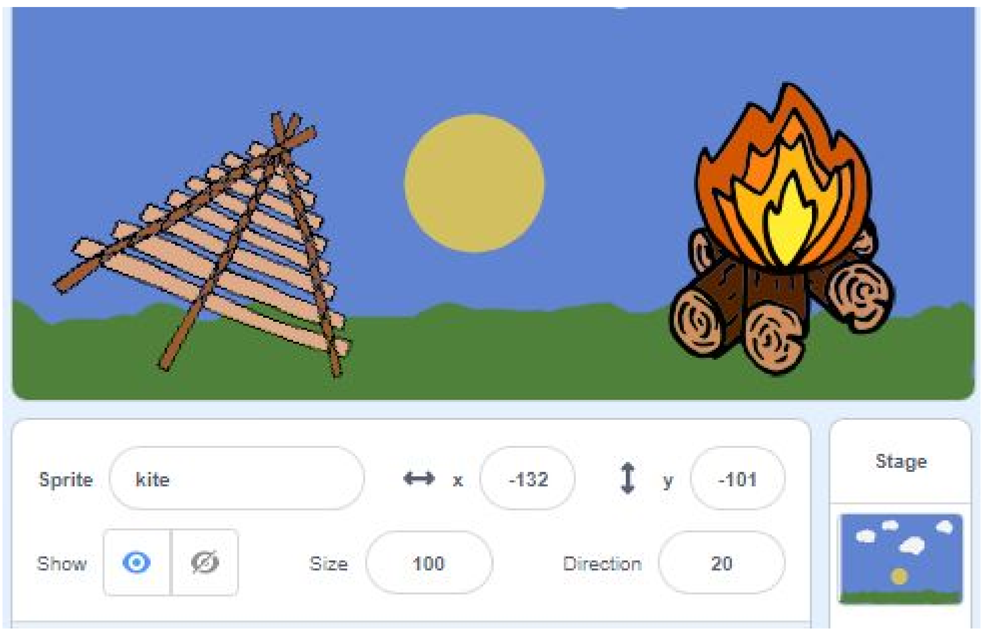
The 'move __ steps' block will moves the sprite half the number of specified pixels in the direction that the sprite is facing (each step is half a pixel), while 'change x by __'

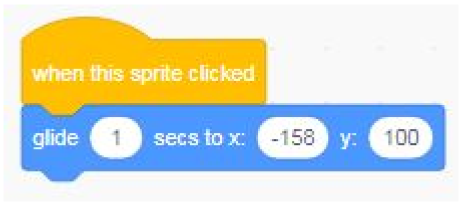
What is a pixel? A pixel is the smallest unit of a digital display. An image displayed on a computer screen is made up of grid of tiny squares. Each square will be a different colour and together this makes up an image. In the scratch, the display area is 480x360 pixels.

will move the sprite the specified number of spaces along the x axis. (the x axis ranges from -240 to 240, the y axis ranges -180 to 180).



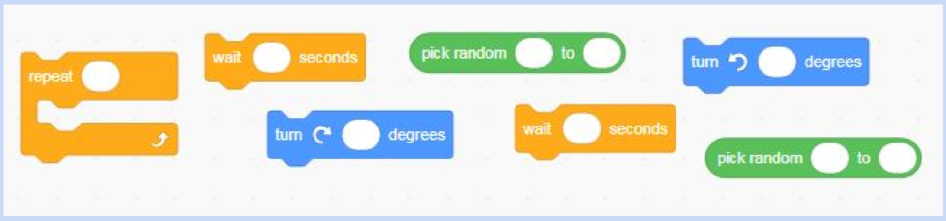
We want the kite to move from the ground into the sky. To do this, we can use the ‘glide’ block from Motion . We can estimate where we want the kite to travel to from the grid, use the coordinates of the mouse cursor that are displayed to the bottom right of the stage, or move the sprite to the right location and use the coordinates given in the sprite area.





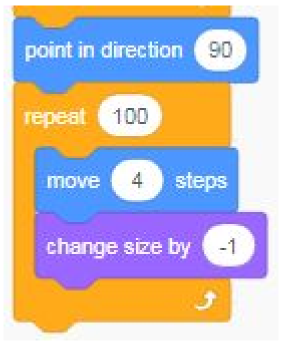
```
when this sprite clicked
glide (1) secs to x: (-158) y: (100)
```

Extra: using these blocks, can you make the kite look like it is fluttering in the breeze?



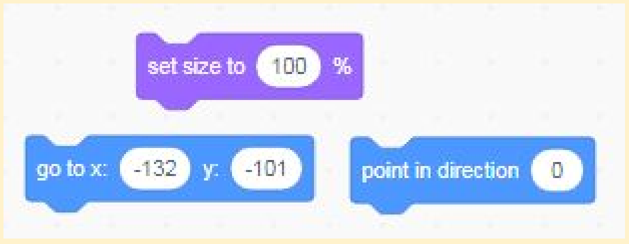
```
repeat ( )
wait ( ) seconds
pick random ( ) to ( )
turn ↺ ( ) degrees
turn ↻ ( ) degrees
wait ( ) seconds
pick random ( ) to ( )
```

We are now going to get the kite to fly off into the corner and disappear into the sky! We first need to change the direction that the kite is pointing in. Next, we're going to use a repeat loop and alternate between a 'move __ steps' block and a 'change size by __' block from **Looks**. The change size block grows or reduces the sprite's size. To make the sprite smaller, we will need to use a negative number.



```
point in direction (90)
repeat (100)
    move (4) steps
    change size by (-1)
```

What happens when you restart your program? Have you got a bug? You might need to reset the kite's position and size

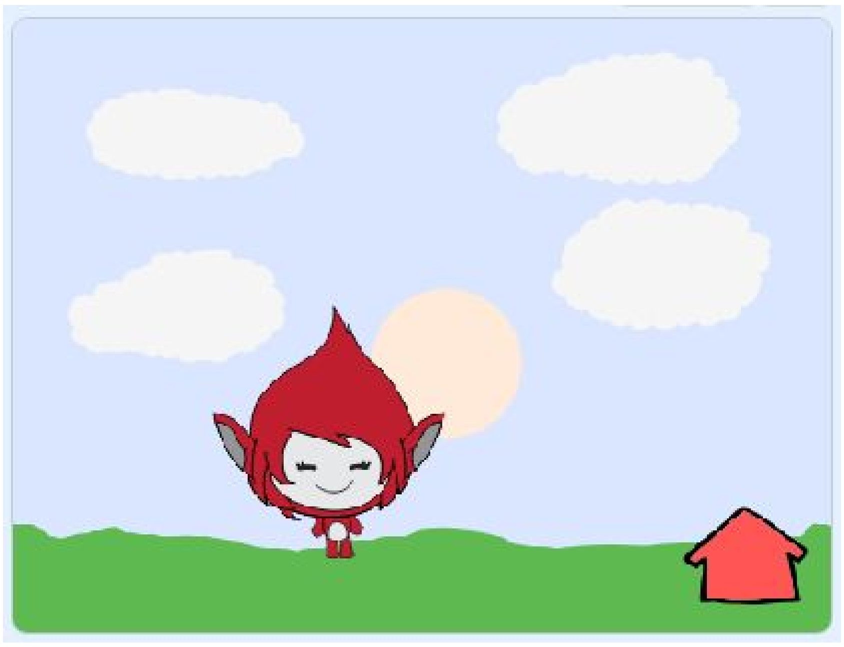


```
set size to (100) %
go to x: (-132) y: (-101)
point in direction (0)
```

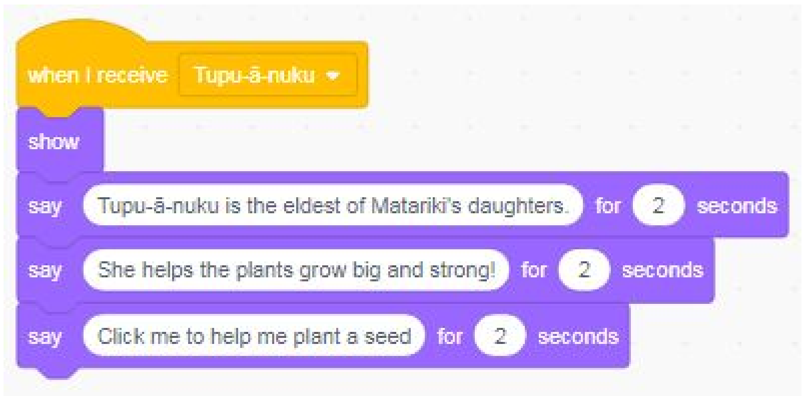
Session Three

Each star in the Matariki cluster has its own story, with different versions existing in different parts of Aotearoa. As an example, we are using the story of Tupu-ā-nuku as told by Te Papa.

Create a new backdrop and use ‘broadcasting’ to move to the backdrop and hide the stars when Tupu-ā-nuku is clicked. Add a new sprite to be your storyteller that appears with your new backdrop.

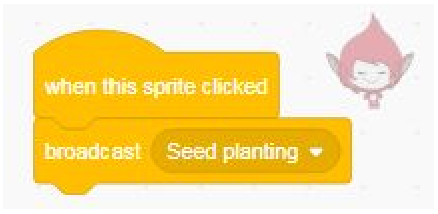


We’re going to get our sprite to tell us a little bit about Tupu-ā-nuku, and then plant a seed which grows into a seedling. When the storyteller sprite receives the Tupu-ā-nuku message, use blocks from **Looks**, to show the sprite and get it to say the following messages:

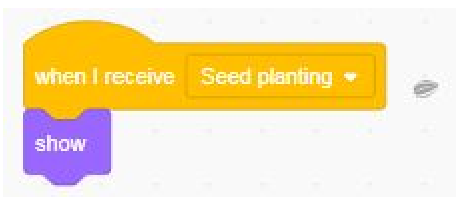


```
when I receive [Tupu-ā-nuku ▾]
show
say [Tupu-ā-nuku is the eldest of Matariki's daughters.] for (2) seconds
say [She helps the plants grow big and strong!] for (2) seconds
say [Click me to help me plant a seed] for (2) seconds
```

Upload these seedling images as a new sprite with five costumes, seed1, seed2, seed3, seed4, seed5. When the storyteller is clicked, they should broadcast a message to the seed to appear.



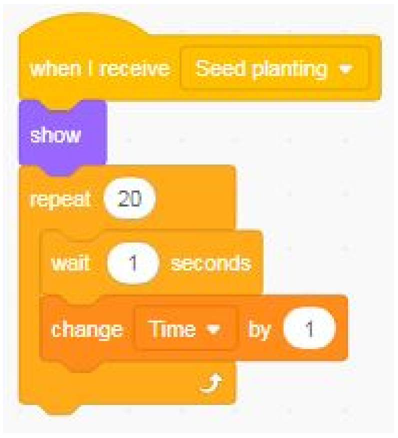
```
when this sprite clicked
broadcast [Seed planting ▾]
```



```
when I receive [Seed planting ▾]
show
```

Variables

A variable is a way for the computer to store small chunks of information like a name or a number. In Scratch, the default variable type is a number. We want the seed to change costume automatically, for example after a set number of seconds have passed. We will do this by creating a timer using a variable. Head into **Data** and click make a Variable. Name your new variable Time. A counter should appear in the top left corner of the stage and it will now count up once per second.



```
when I receive [Seed planting ▾]
show
repeat (20)
    wait (1) seconds
    change [Time ▾] by (1)
↺
```

You'll need to reset your timer. Add a 'set Time to 0' block to your 'when flag clicked block'

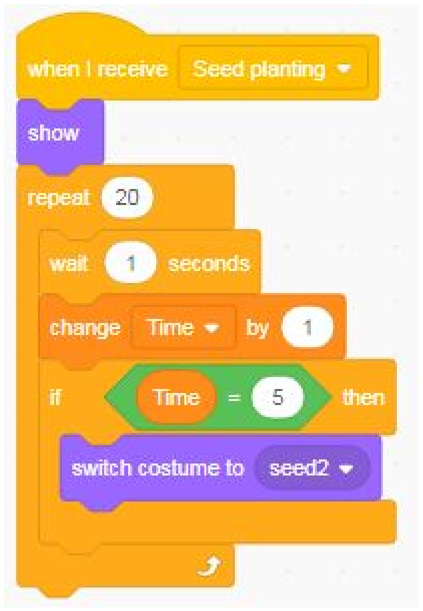
Conditional Statements

A conditional statement is a statement that a computer uses to help it make a decision. We are going to create some code that causes the seed to grow depending on how much time has passed.

From **Control**, find a 'if ____ then' block and add it inside your repeat. An if block is an example of a conditional statement. We want the seed to change its costume if 5 seconds have passed. From **Operations**, find the '__ = __' block and drag it into the gap in the 'if ____ then' block.

We use conditional statements all the time, for example "if you pick up all your toys, then you can watch tv" or "if you eat your broccoli, then you can have an ice cream or else you'll have to have an apple".

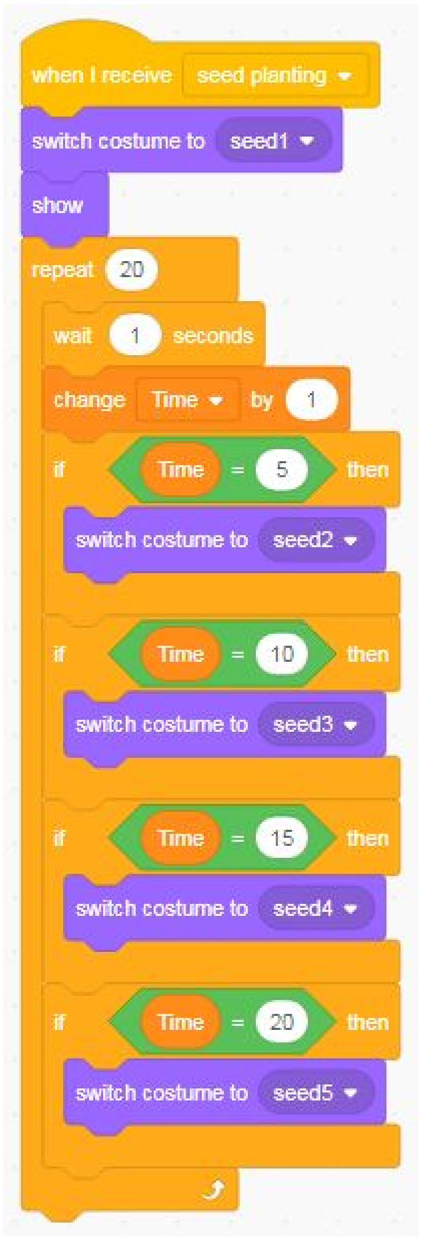
From **Data**, drag a 'Time' block into the first white space in the '__ = __' block, and type 5 in the second white space. From **Looks**, find a 'switch costume to ____' block and drag it inside the 'if ____ then' loop. Select 'seed2' from the drop down list.



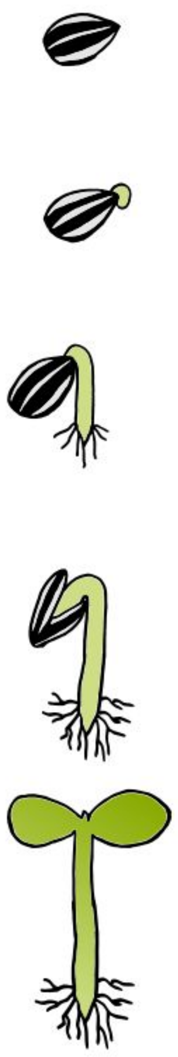
```
when I receive [Seed planting ▾]
show
repeat (20)
    wait (1) seconds
    change [Time ▾] by (1)
    if <(Time) = (5)> then
        switch costume to [seed2 ▾]
```

Now run your program. Does your seed change costume? Add in three more 'if ____ then' blocks to make the seed change costume every 5 seconds. Does it look like your seed is growing?

Extra: Do we need to use a variable to change the costume? Why do you think we are using a variable instead of using an alternative method?



```
when I receive [seed planting ▾]
switch costume to [seed1 ▾]
show
repeat (20)
    wait (1) seconds
    change [Time ▾] by (1)
    if <(Time) = (5)> then
        switch costume to [seed2 ▾]
    if <(Time) = (10)> then
        switch costume to [seed3 ▾]
    if <(Time) = (15)> then
        switch costume to [seed4 ▾]
    if <(Time) = (20)> then
        switch costume to [seed5 ▾]
```



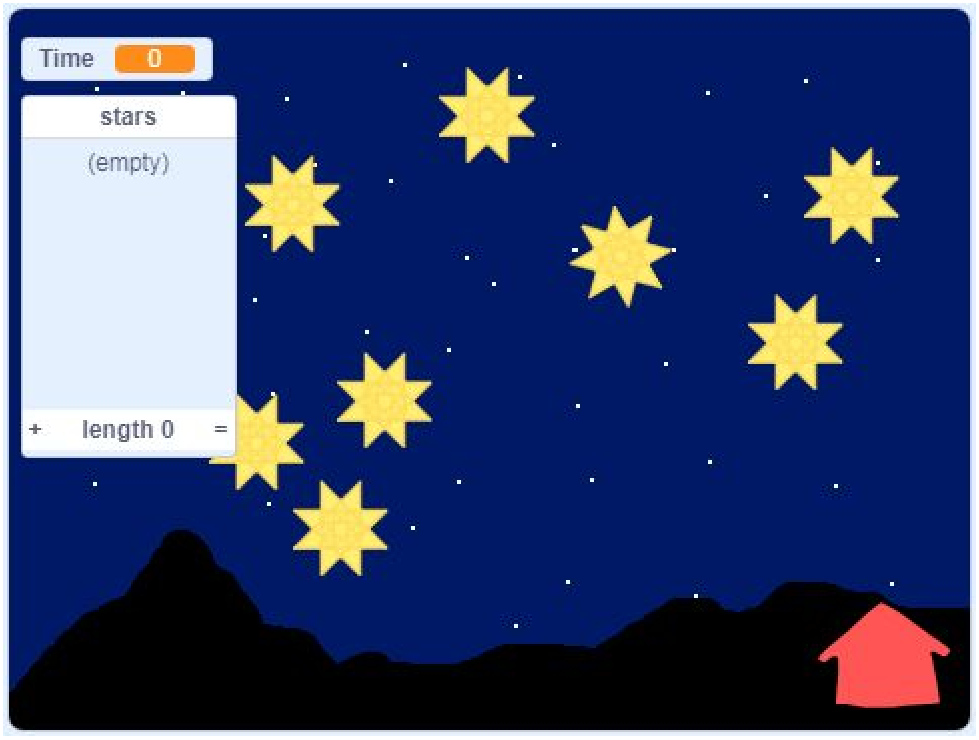
Session Four

Lists

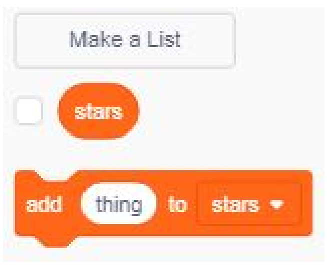
We are going to work on improving the user experience of our interactive Matariki animation. We will do this by keeping a record of which stars the user has visited. We will do this using a list.

A list (also known as an array) is a way of storing lots of pieces of information of the same type together.

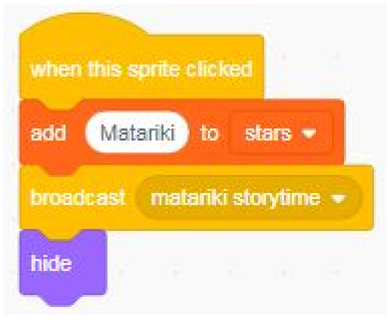
Head to **Variables**, and click 'make a list'. Give your new list a name, for example 'Stars'. You'll notice that the list appears in the display area. If you don't want to see it, right-click and hide the list.



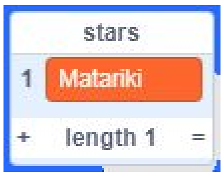
When we click on a star, we can use the 'add ____ to ____' block to record the name of the star we have just visited in our 'Stars' list.



```
Make a List
stars
add thing to stars
```



```
when this sprite clicked
add Matariki to stars
broadcast matariki storytime
hide
```

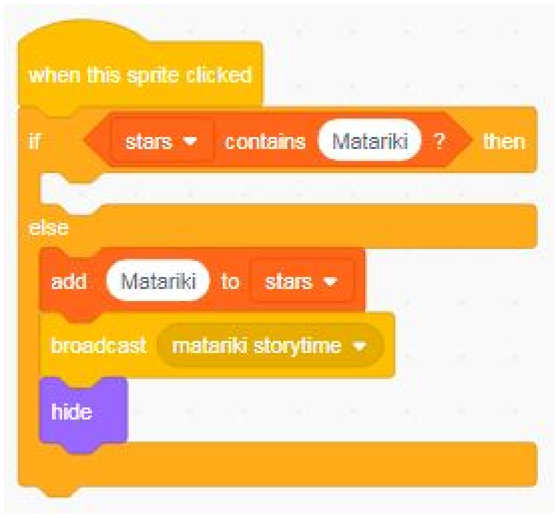


```
stars
1 Matariki
+ length 1 =
```

Questions and Answers

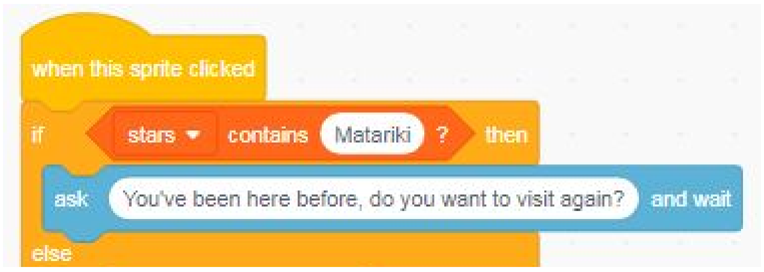
To check whether or not the user has visited a star, we can use a conditional statement with a question. We are going to create some code that causes something to happen depending on the answer given.

From Control, find the 'if ___ then else' block and drag it into the programming area. We want the program to check if our list of stars contains the star we have just clicked on. A block for this task exists in Variables. If our list does not contain the name of the star, we want the program to proceed as before.



```
when this sprite clicked
if <stars contains Matariki ?> then
else
    add Matariki to stars
    broadcast matariki storytime
    hide
```

If the list does contain the name of the star, we want to highlight this to the user. There are several ways we can do this, including using an 'ask' block from Sensing. the 'ask' block asks the user a question, and stores their response as a variable. We can write our program so that it responds to the answer. Type a question into the 'ask' block, asking the user if they want to revisit a star if they have already seen it.



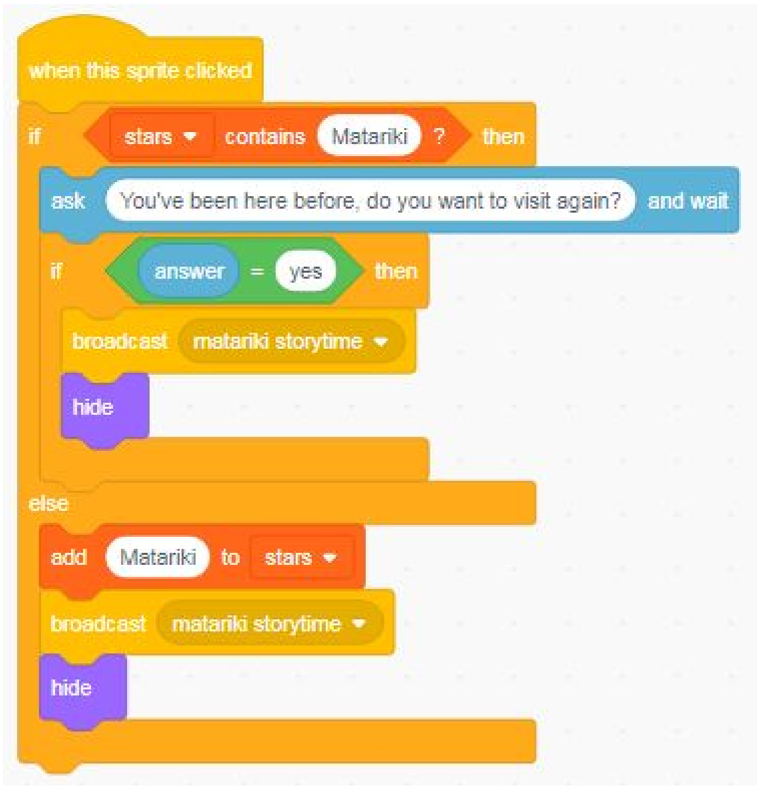
```
when this sprite clicked
if <stars contains Matariki ?> then
    ask You've been here before, do you want to visit again? and wait
else
```

Nested Statements

We can use a second conditional statement to respond if the answer is yes. From **Control**, find the 'if ____ then' block and put it inside your first if statement. This then becomes a nested statement.

We want the program to take the user to the star's page if they answered 'yes' to the question. Head into **Operations**, find the '__ = __' block and slot it into the space in the 'if ____ then else' block. From **Sensing**, find the 'answer' block. Drag it into the first space in the '__ = __' block and type "yes" into the second space. Inside the 'if' block, add a 'broadcast' and a 'hide' block.

What will happen if the user types no?



```
when this sprite clicked
if <[stars v] contains [Matariki] ?> then
    ask [You've been here before, do you want to visit again?] and wait
    if <(answer) = [yes]> then
        broadcast [matariki storytime v]
        hide
    end
else
    add [Matariki] to [stars v]
    broadcast [matariki storytime v]
    hide
end
```

Add this code to all of your stars using the Backpack. Don't forget to change the names you are adding to the list and checking the list for!