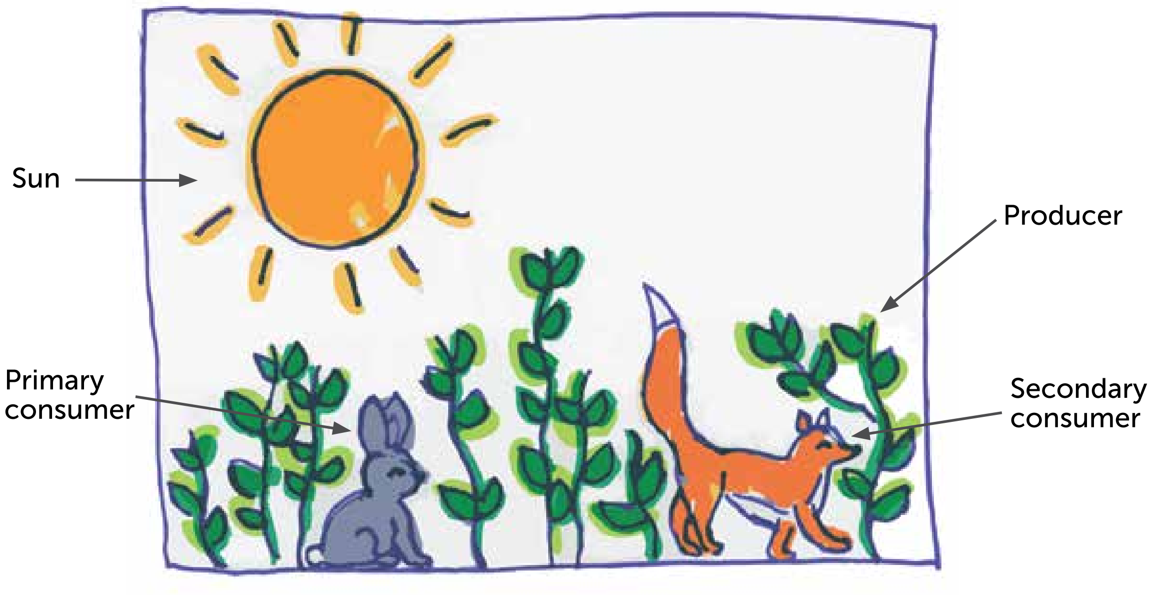
Creating Algorithms: Illustrate a Phenomenon

An algorithm is a repeatable process that delivers an expected result. In this assignment, you are going to create an algorithm to illustrate a scientific phenomenon. There are many predictable events that occur in the natural world, such as life cycles, chemical reactions, or Earth's processes. One way to illustrate these phenomena is by programming an algorithm. If your algorithm represents a mathematical relationship between parts of system, it may be a computational model. If there are no input/outputs or defined mathematical relationships, you are likely simply illustrating a scientific phenomenon. See resources for Creating Computational Models to learn more about how your illustration could be modified to become a computational model. In the example below, a student illustrated a simple food chain with a producer, herbivore and carnivore by programming in Scratch.

1 Part 1: Describe the Phenomenon

Sketch and **label** the phenomenon you will illustrate here:



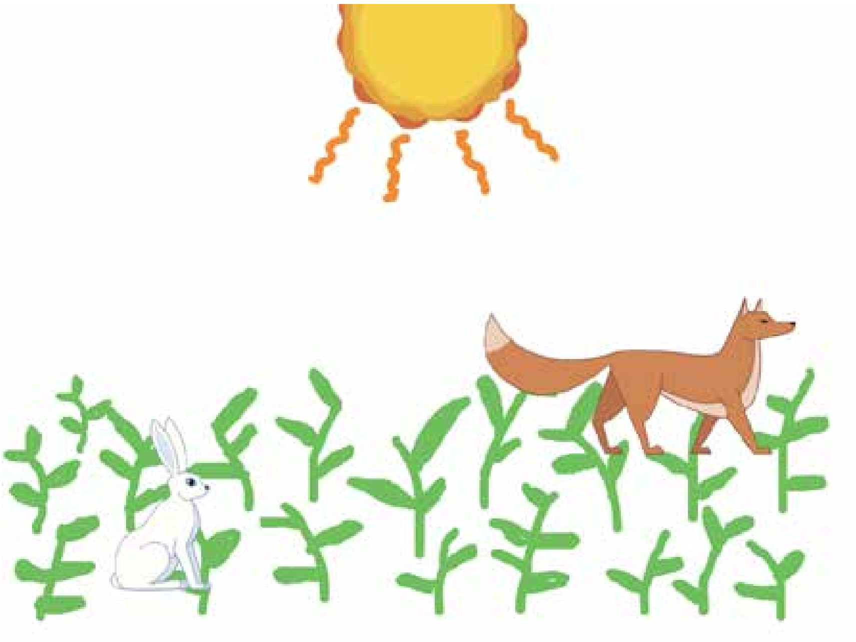
Think about the **parts** of the phenomenon you will illustrate, the **purpose** of each part, and if the part will perform an action in your program.

Part	Purpose	Action (yes/no). If yes, describe.
Sun	Provides energy	Appear and disappear on a timer to signal night and day
Producer	Converts sun's energy into food through photosynthesis	Grow/reproduce when exposed to the sun
Primary consumer (herbivore)	Eats producers	If herbivore is pushed, it eats the producer
Secondary consumer (carnivore)	Eats consumers	If carnivore is pushed, it eats the herbivore

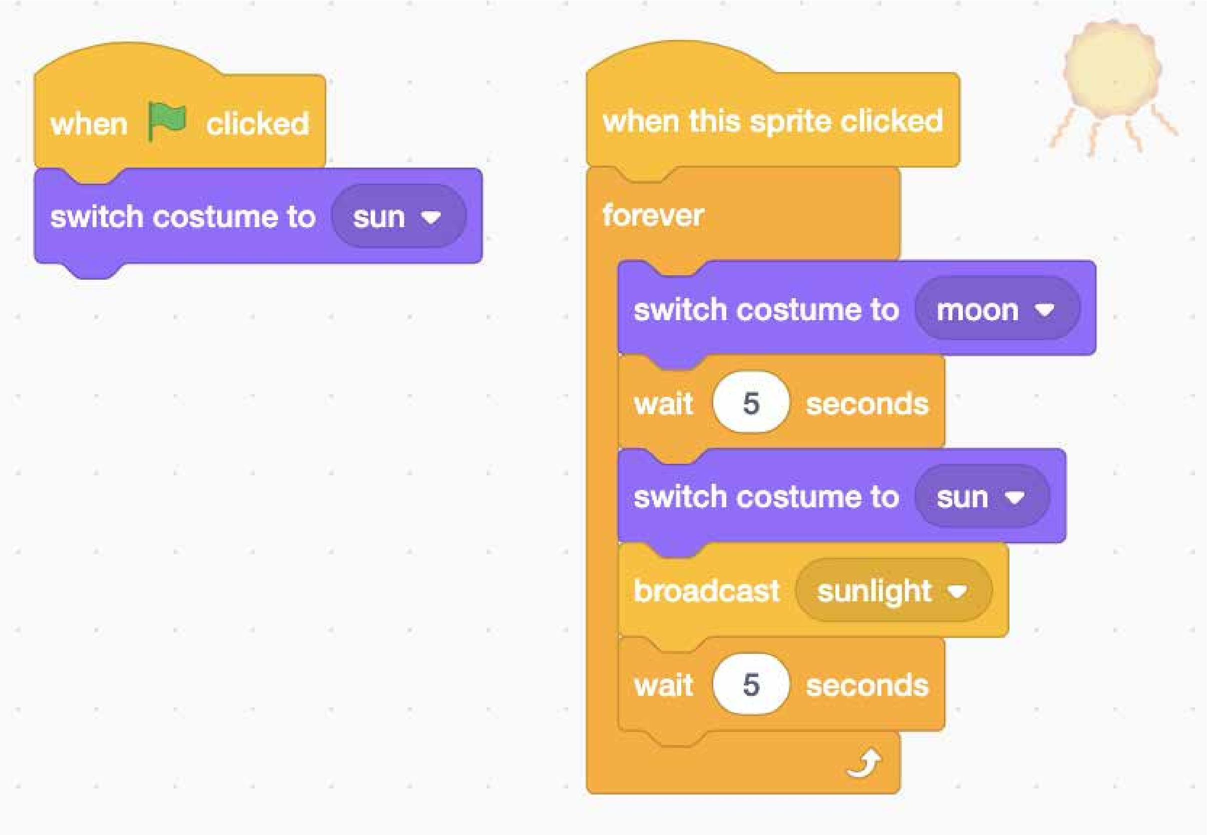
2 Part 2: Create Your Program

Now you will use a computational tool to create your algorithm. There are many tools available to program algorithms, such as coding platforms (e.g., Scratch, Snap, MakeCode) or computational making kits (e.g., Hummingbird Robots, Micro:bit, LegoWedo, Arduino, Raspberry Pi). Your teacher will tell you which tool(s) you may use for this assignment.

Link to Scratch project here



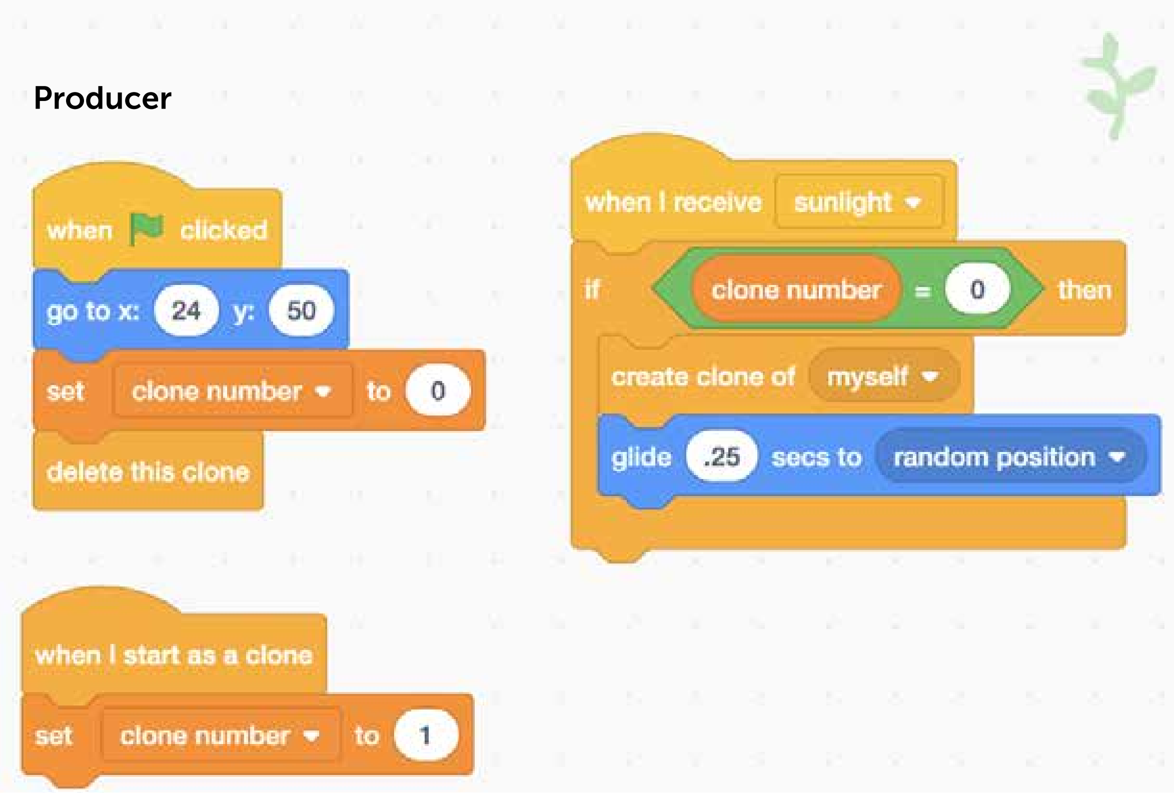
Sun



```
when [flag] clicked
switch costume to (sun ▾)

when this sprite clicked
forever
    switch costume to (moon ▾)
    wait (5) seconds
    switch costume to (sun ▾)
    broadcast (sunlight ▾)
    wait (5) seconds
↻
```

Producer

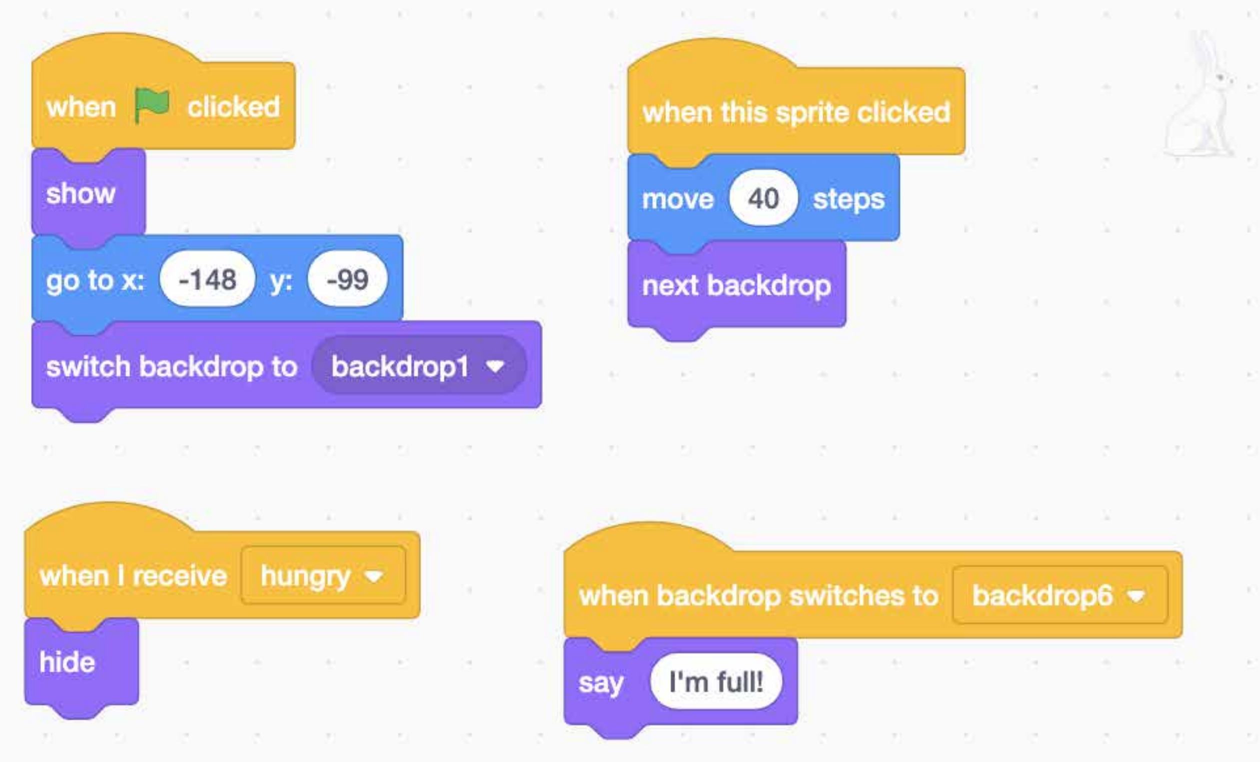


```
when [flag] clicked
go to x: (24) y: (50)
set [clone number ▾] to (0)
delete this clone

when I start as a clone
set [clone number ▾] to (1)

when I receive [sunlight ▾]
if <(clone number) = (0)> then
    create clone of (myself ▾)
    glide (.25) secs to (random position ▾)
```

Primary consumer (herbivore)



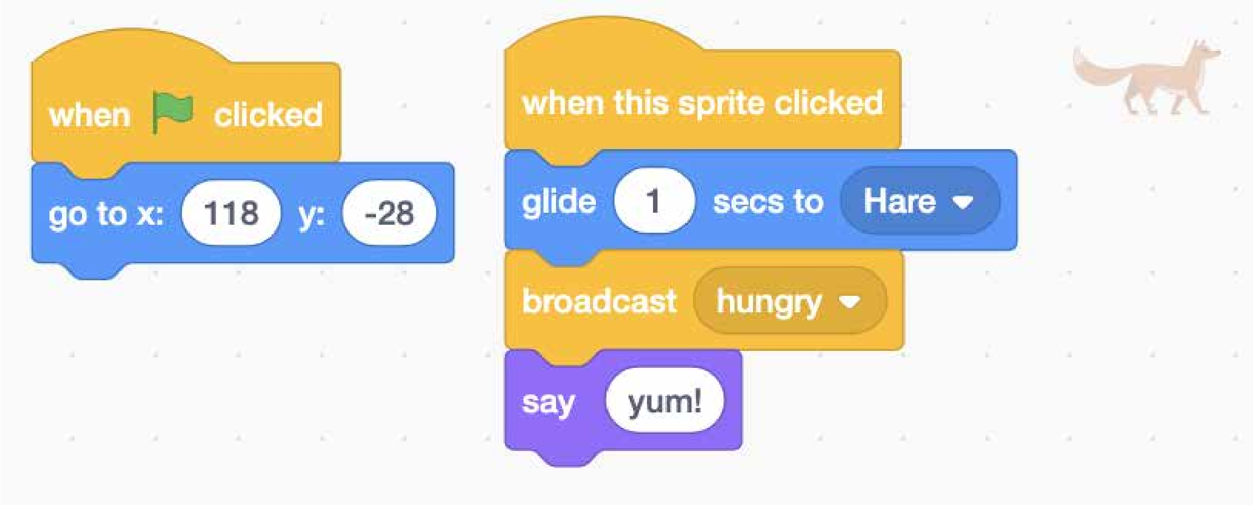
```
when ⚑ clicked
show
go to x: -148 y: -99
switch backdrop to backdrop1 ▾
```

```
when this sprite clicked
move 40 steps
next backdrop
```

```
when I receive hungry ▾
hide
```

```
when backdrop switches to backdrop6 ▾
say I'm full!
```

Secondary consumer (carnivore)

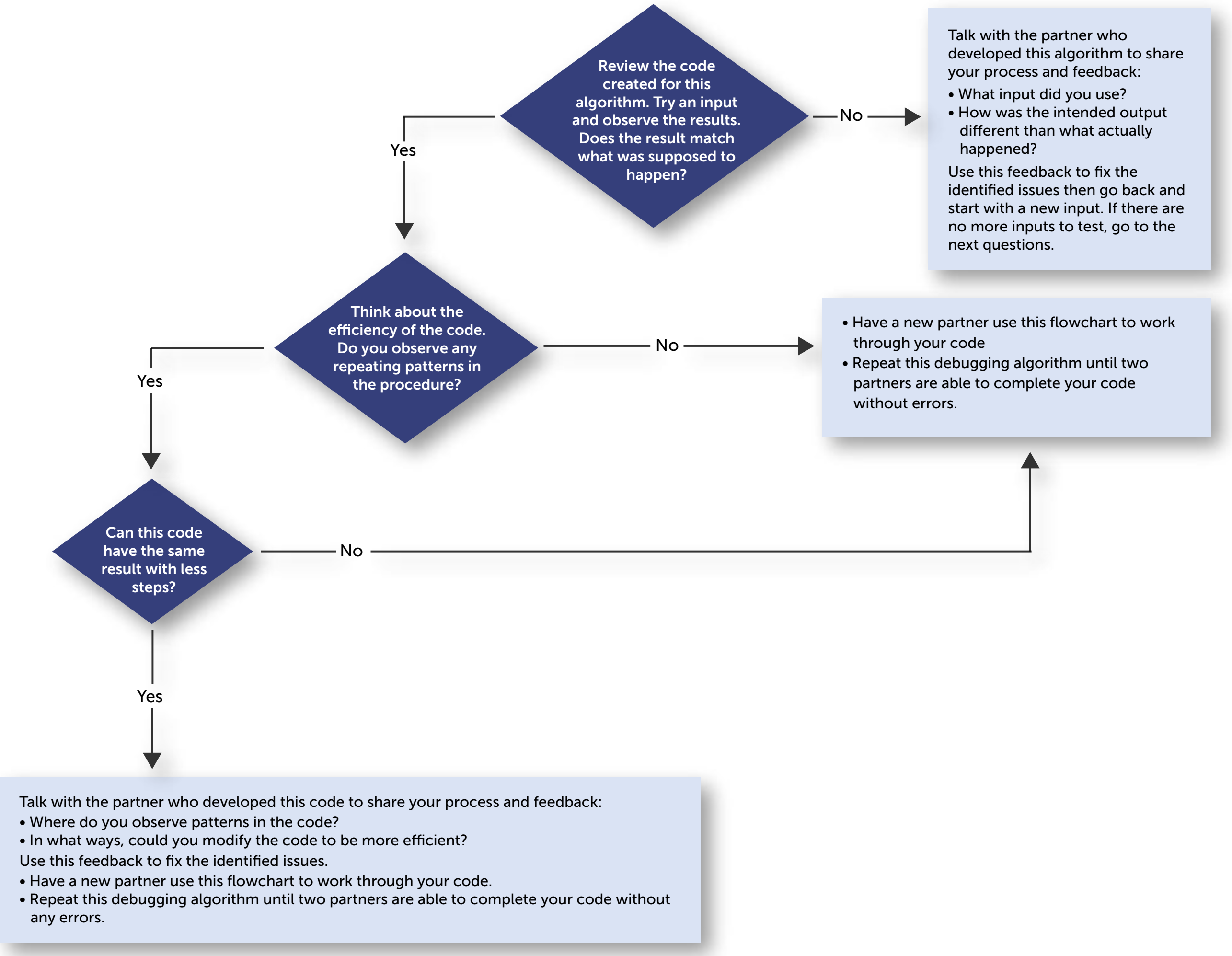


```
when ⚑ clicked
go to x: 118 y: -28
```

```
when this sprite clicked
glide 1 secs to Hare ▾
broadcast hungry ▾
say yum!
```

3 Part 3: Pair Debugging Algorithm

While completing your program, work with a partner to debug -- which is to find and fix errors -- and improve it:



Review the code created for this algorithm. Try an input and observe the results. Does the result match what was supposed to happen?

Talk with the partner who developed this algorithm to share your process and feedback:

- What input did you use?
- How was the intended output different than what actually happened?

Use this feedback to fix the identified issues then go back and start with a new input. If there are no more inputs to test, go to the next questions.

Think about the efficiency of the code. Do you observe any repeating patterns in the procedure?

- Have a new partner use this flowchart to work through your code
- Repeat this debugging algorithm until two partners are able to complete your code without errors.

Can this code have the same result with less steps?

Talk with the partner who developed this code to share your process and feedback:

- Where do you observe patterns in the code?
- In what ways, could you modify the code to be more efficient?

Use this feedback to fix the identified issues.

- Have a new partner use this flowchart to work through your code.
- Repeat this debugging algorithm until two partners are able to complete your code without any errors.