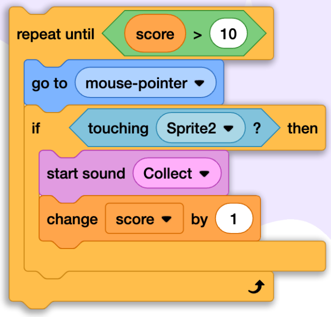


Making Interactive Projects with Conditional Statements

Have you ever wanted to create a Scratch program that is interactive or offers multiple outcomes? Some Scratch programs are **static**: the outcome is fixed and the same thing happens each time. Some are **dynamic**: they are capable of action or change each time they are run. In order to create dynamic programs, the programmer can use conditional statement blocks to give instructions on how the project should respond in different circumstances.

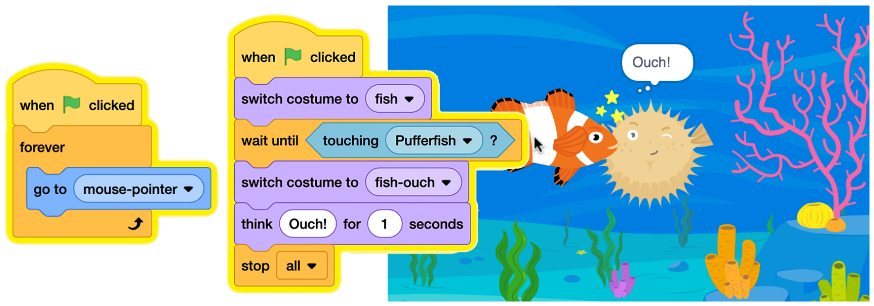
In this guide, you'll find:

What Are Conditional Statements

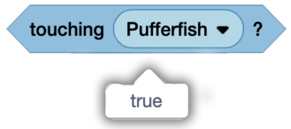
Imagine a project where a fish can be controlled by the user's mouse (see example here). The fish's script says: forever, go to the mouse-pointer. What if you wanted a unique action to take place if the fish touches another sprite, like a pufferfish?

We can create a script that uses a "wait until" conditional statement block and tells the program to wait until the sprites are touching. Then, once touching, we can program an action like changing costumes and saying, "Ouch!" Now, the program is dynamic and interactive!

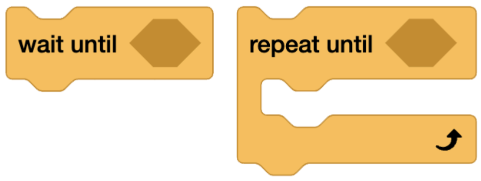
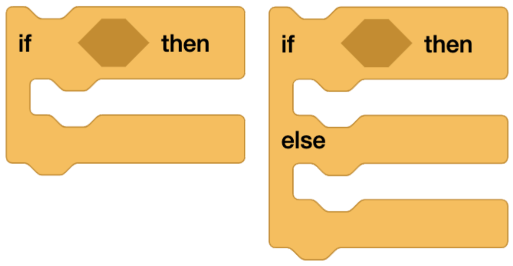
- If the user never moves the fish over the pufferfish, nothing happens.
- If they do move the fish on top of the pufferfish, they get to see a fun animation!



A conditional statement can be true or false. In this case, if the sprites are touching it is true. If they are not touching, it is false.



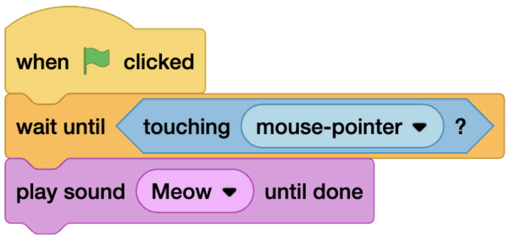
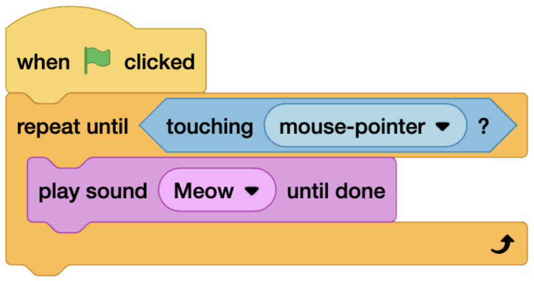
Under the Control category in Scratch, you will find a number of conditional statement blocks that allow you to adjust your program based on specific conditions. In Scratch, you'll encounter two types of conditional statements:

 `wait until < >` `repeat until < >`	 `if < > then` `if < > then` / `else`
"until (something) is true or false"	**"if (something) is true or false then"**

Let's explore these conditional statement blocks, practice with some examples, and learn how to create dynamic and interactive programs!

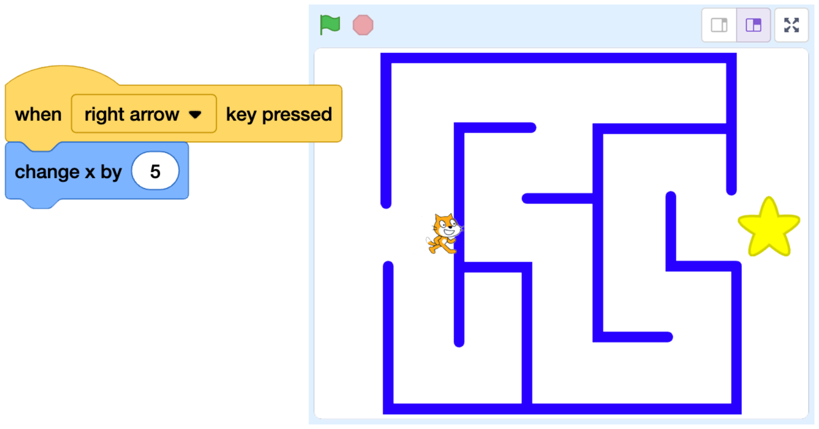
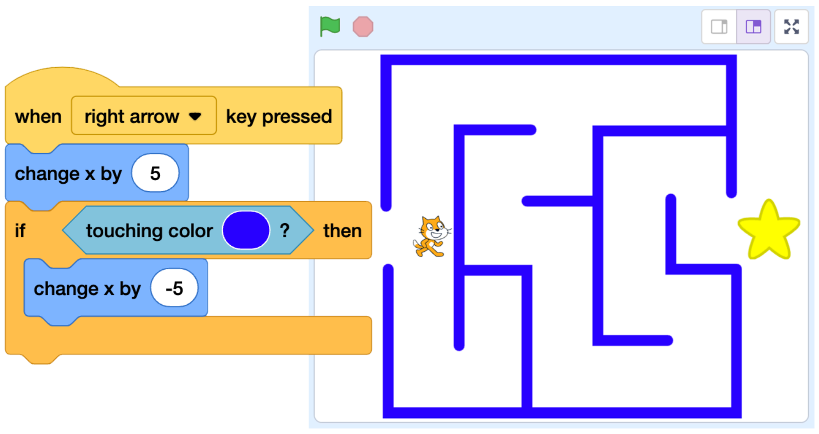
Until True or False

In the fish game above, we used an "until (something) is true or false" block to trigger an action. The two scripts below use "wait until" and "repeat until" to perform actions. Let's take a closer look to see the difference:

 `when [flag] clicked` / `wait until <touching (mouse-pointer ▾) ?>` / `play sound (Meow ▾) until done`	 `when [flag] clicked` / `repeat until <touching (mouse-pointer ▾) ?>` / `  play sound (Meow ▾) until done`
"Wait Until" Start with a simple script like: when the green flag is clicked, **wait until a certain condition is true** and play a sound. In this case, we could look under the sensing category and pick a condition like "touching mouse-pointer." To test this script and see how it works, click the green flag, wait, and then hover your mouse over your sprite. Your sound will play *only* when the mouse touches the sprite.	**"Repeat Until"** Adjust your script to say: when the green flag is clicked, **repeat** playing a sound **until a certain condition is true**. (A short sound is best for this test.) Let's use the same condition "touching mouse-pointer." To test this script and see the difference, click the green flag, wait, and then hover your mouse over your sprite. The sound plays repeatedly and only stops once your mouse touches the sprite.

If True or False

Another version of conditional statements in Scratch is checking to see "if (something) is true or false." For example, say you want to create a maze game (see example here) where the player controls the sprite with keyboard arrow keys:

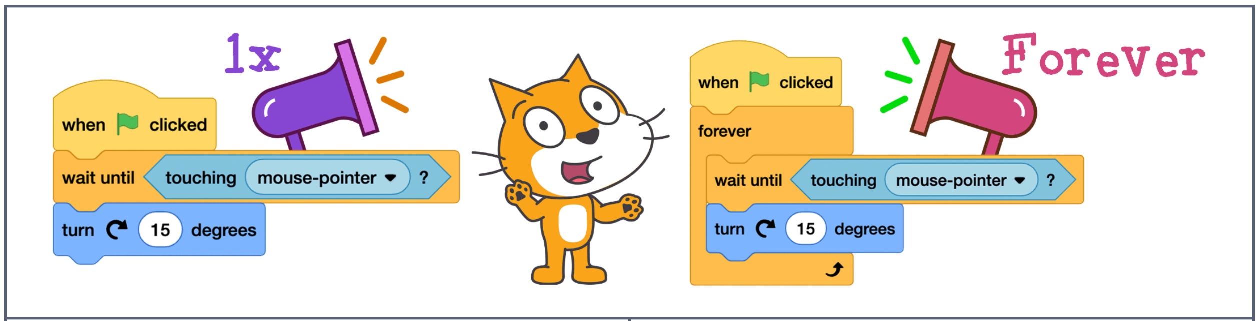
Initial Code When a key is pressed the sprite is coded to move a certain number of steps. But you need to prevent the sprite from walking through the walls. What can make that possible? Conditional statements!	 when right arrow key pressed change x by 5
Add an "If Then" Statement We can use the color of the maze walls as our condition to affect the behavior of the sprite. Add an **"if then"** block to the bottom of the sequence. Use the "touching color" sensing block as the condition and select the color of the walls. What should happen if it touches the walls? If the sprite moves the same number of pixels in the opposite direction, it will look like the sprite hasn't moved at all and has been stopped by the wall.	 when right arrow key pressed change x by 5 if touching color ● ? then change x by -5

Test out this maze code and then add additional scripts to move the sprite left, up, and down.

How could you use an "if then" or "if then else" block to program a win message, a sound, or an animated celebration when the sprite reaches the end of the maze? How could you use conditional statements to program obstacles to avoid or items to collect for points?

Conditional Loops

What if you want the program to continually check to see if the condition is true or false (continually "listen") and perform actions over and over? You'll need to place the sequence inside a loop to create a **conditional loop**. Compare the difference between these two scripts:

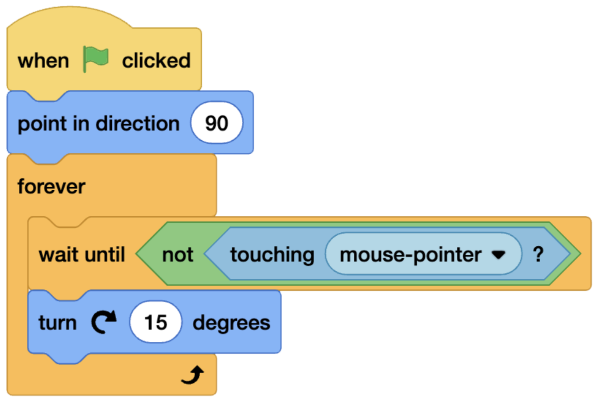


```
when [flag] clicked
wait until <touching (mouse-pointer ▾) ?>
turn ↻ (15) degrees
```

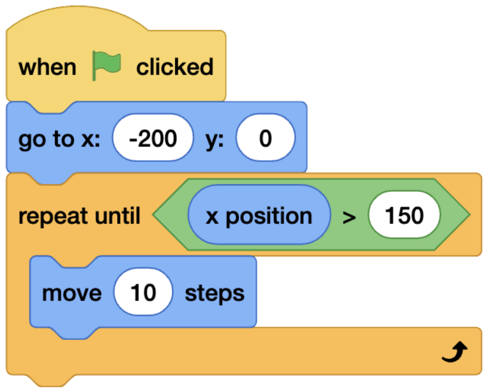
```
when [flag] clicked
forever
    wait until <touching (mouse-pointer ▾) ?>
    turn ↻ (15) degrees
↺
```

In this example, once the condition is met and the code is run, the program stops (you can tell the script has stopped running because it is no longer highlighted). It only "listens" once for the condition.	In this example, the program continuously "listens" to see if the condition is true. The script inside the forever loop restarts after the condition has been met and the code is run. It checks if the condition is true/false again.

Condition Examples



```
when [flag] clicked
point in direction (90)
forever
    wait until <not <touching (mouse-pointer ▾) ?>>
    turn ↻ (15) degrees
↺
```

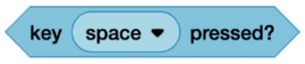


```
when [flag] clicked
go to x: (-200) y: (0)
repeat until <(x position) > (150)>
    move (10) steps
↺
```

"Not" Operator What if we want something to happen if something else is not happening (like if the mouse-pointer is not touching the sprite)? Under the Operators category, look for a "not" operator. Test the code stack above by hovering the mouse-pointer on and off of the sprite. What difference do you notice?	**Comparison Operators** Comparison operators compare the values of two pieces of data and determine if the equation is true or false. This conditional statement above says that the sprite should move 10 steps at a time until the x position is greater than 150. Test this code stack and see the result.

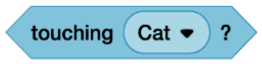
There are a wide variety of conditions in Scratch that you can choose from to complete your conditional statement. **Blocks that report "true" or "false" values are known as Boolean blocks**, and you can identify them by their elongated hexagonal shape. Some examples of conditions you could use include:

- user actions, such as pressing certain keyboard keys or positioning or clicking the mouse



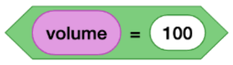
```
key (space ▾) pressed?
```

- sprite interactions (touching another sprite), comparisons (distance between sprites), and touching colors of sprites or backdrops

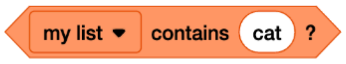


```
touching (Cat ▾) ?
```

- data input by users, data stored in variables and lists, or data stored in reporter blocks



```
(volume) = (100)
```



```
[my list ▾] contains (cat) ?
```

Debugging Conditional Statements

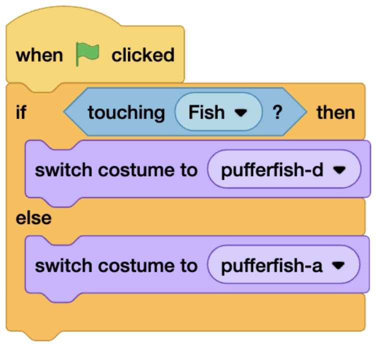
As you are creating a program, you may want to do some user-testing by having a few friends or family members try it out. Is everything working as expected, or do you need to do some debugging to address errors or unexpected behaviors?

Debugging: Why Isn't It Checking?

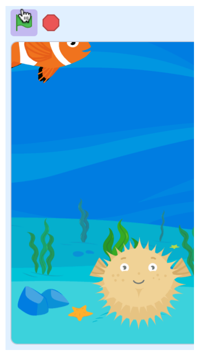
If you only need the program to check your condition once, you don't need to place your conditional statement inside of a loop. But if you want the program **to continually check** to see if the condition is true or false (continually "listen") and perform actions over and over, you'll need to place the code sequence inside of a loop to **create a conditional loop**.

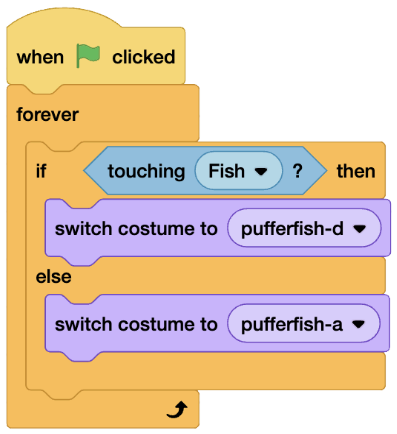
For instance, in the first example to the right, when I begin the fish program and click the green flag, the fish and pufferfish aren't touching, so the program stops the script because the condition has been checked and was false.

In the second example, the program continually checks to see if the condition is true or false and will run the code under "if then" when true or under "else" when false.



```
when ⚑ clicked
if <touching (Fish ▾) ?> then
    switch costume to (pufferfish-d ▾)
else
    switch costume to (pufferfish-a ▾)
```





```
when ⚑ clicked
forever
    if <touching (Fish ▾) ?> then
        switch costume to (pufferfish-d ▾)
    else
        switch costume to (pufferfish-a ▾)
↺
```

Debugging: Sequential Order

Say I want to add some code to the pufferfish sprite in the fish game, so the player sees different costumes in different circumstances.

In the first example on the right, I have nested two “if then else” statements:

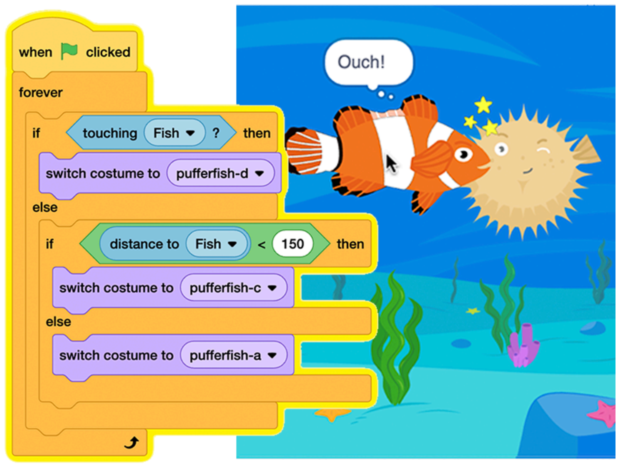
- First, the program checks to see **if** the sprites are touching, and if that is true, **then** it shows costume d.
- **Else**, if touching is false, it checks **if** the distance between the sprites is close, and if it is, **then** shows costume c.
- **Else**, it shows the initial costume a.

What if the order of the nested “if then else” statements was different? In the second example:

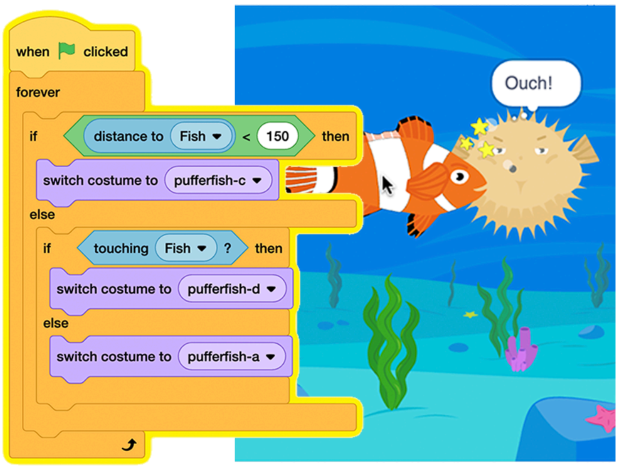
- First, the program checks to see **if** the distance between the sprites is close, and if it is, **then** shows costume c.
- **Else**, if the distance is greater than 150 pixels, it checks **if** the sprites are touching, and if that is true, **then** it shows costume d.
- **Else**, it shows the initial costume a.

In the second example, we never see the winking pufferfish (costume d) when the sprites touch. Why?

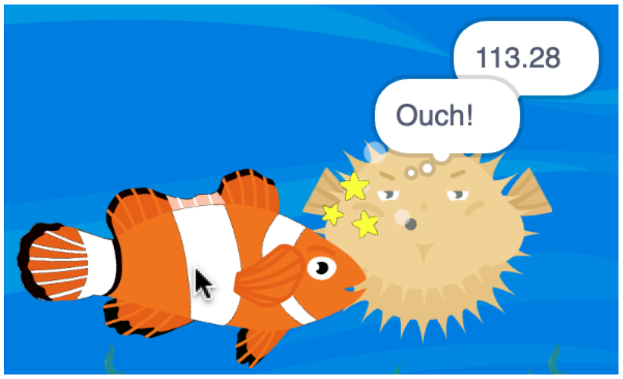
In the sequential order of the second example, the program is checking first to see if the distance between the sprites is less than 150. Since that is also true when the sprites are touching, the program doesn’t move on to check touching.



```
when ⚑ clicked
forever
  if touching Fish ? then
    switch costume to pufferfish-d
  else
    if distance to Fish < 150 then
      switch costume to pufferfish-c
    else
      switch costume to pufferfish-a
```



```
when ⚑ clicked
forever
  if distance to Fish < 150 then
    switch costume to pufferfish-c
  else
    if touching Fish ? then
      switch costume to pufferfish-d
    else
      switch costume to pufferfish-a
```



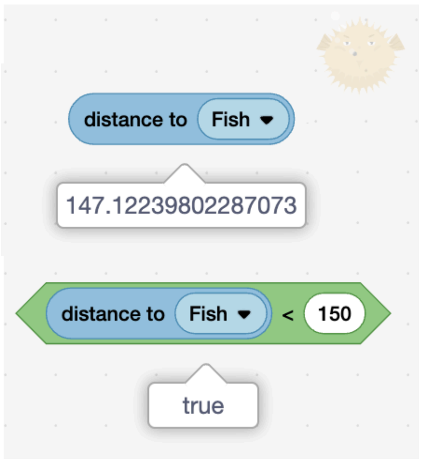
Debugging: Mathematical Expressions

As I'm working on my pufferfish code, how do I **determine the** distance I want to **input in my comparison operator** that is checking distance? It can be hard to judge pixel distance by sight.

This "distance to Fish" reporter will report the distance between the two sprites' center points. **Click on the reporter block on the script area** to see the value.

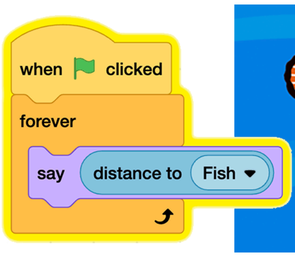
Now, put "distance to" into an operator block like greater than or less than, move the sprites together and apart and **click on the condition on the script area to see if it reads true or false**.

If you want to tinker and quickly see what the distance reading is when the fish is in different positions, you could also have the pufferfish **continually say** the "distance to Fish" as you move the fish around.

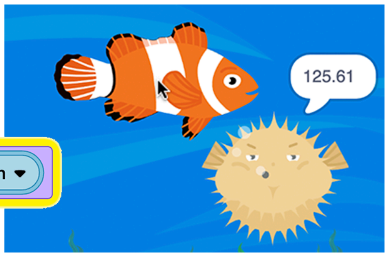


```
distance to (Fish ▾)
147.12239802287073

<(distance to (Fish ▾)) < (150)>
true
```

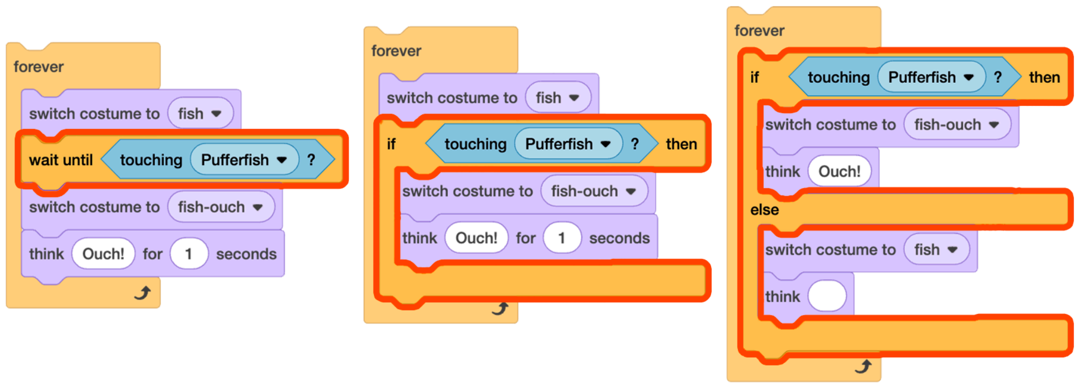


```
when ⚑ clicked
forever
    say (distance to (Fish ▾))
```



Multiple Pathways/Solutions

There is often more than one solution/more than one way to code a program to get a similar result. For instance, if we return to the code for the fish in the fish game, say I want to adjust the code to add a conditional loop so the fish and the pufferfish can interact more than once.



```
forever
    switch costume to (fish ▾)
    wait until <touching (Pufferfish ▾) ?>
    switch costume to (fish-ouch ▾)
    think (Ouch!) for (1) seconds
```

```
forever
    switch costume to (fish ▾)
    if <touching (Pufferfish ▾) ?> then
        switch costume to (fish-ouch ▾)
        think (Ouch!) for (1) seconds
```

```
forever
    if <touching (Pufferfish ▾) ?> then
        switch costume to (fish-ouch ▾)
        think (Ouch!)
    else
        switch costume to (fish ▾)
        think ()
```

Notice all three solutions above use similar blocks, but one uses "wait until," one uses "if then," and the third uses an "if then else" conditional statement. Look at, recreate, and tinker with these three scripts. What is the same and what is different?

Unplugged Practice with Conditional Statements

Explore if/then and while/until conditional blocks using our Simon Says Conditional Statements unplugged game activity. One person functions as the programmer while the participants, as computers, need to determine if statements are true or false and then act accordingly.

See our companion coding cards: **Conditional Statements Coding Cards**

See our companion resource videos here for more:

Conditional Statements: Make Interactive Projects (Part 1)| Tutorial

Conditional Statements: Nesting, Debugging, and Beyond (Part 2)| Tutorial

Tip: If you'd like to translate this guide, **click here to make a copy** of this Google doc.