

05 - Conditional Statements and Comparison Operators

JavaScript Conditional Statements and Comparison Operators

WE'LL COVER THE FOLLOWING



- frame, frameRate, and frameCount
- Conditionals
- Summary
- Practice

frame, frameRate, and frameCount

In the previous chapter, we saw some of the variables that p5.js makes available for us. One important thing to note is that these variables can only be used from inside the p5.js functions **setup** and **draw**. If we were to try to use them outside these functions, we would get an error saying that they are not declared.

Let's see another useful variable that p5.js makes available for us:

frameCount.

Remember how we defined a **count** function in the previous chapter to be able to count the number of times that the **draw** function is getting called. We can actually use the variable called **frameCount** that p5.js provides us for this same purpose. **frameCount** is a variable that keeps count of the number of times the **draw** function is called throughout the lifetime of a program. By default, the **draw** function is called a maximum of 60 times per second. A setting called **frameRate** inside p5.js determines this value.


The introduction of this variable warrants a discussion about what **frames** are in p5.js. We can think of a **frame** as the result of the **draw** function call.


draw function gets called numerous times in a second, and the **frameRate** function determines this amount. If we are to call the **frameRate** function with no arguments, it will return us the current **frameRate** for p5.js - which we can save into a variable and **console.log** to see its value for every frame.


Output

JavaScript

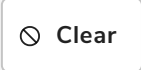
HTML







Console



0
1.9086701340792422
47.64173414883949
52.042675012501185
59.31198100833859
60.66120715764547

The default rate is around 60. This means that the **draw** function will be executed for a maximum amount of 60 times per second. This number depends on our system resources. For performance related reasons, such as due to limited system resources, the actual frame rate that can be achieved

might be lower than this target value. We can consider 60 as ideal frame rate that p5.js strives to achieve, but the actual frame rate and hence the performance might be less than this.

Think of frames as sheets in a flip book animation. More sheets viewed per second will mean smoother animation. That's why high frame rates are desirable. The animation might look jaggy if the frame rate is low. We can set the frame rate explicitly in p5.js by passing an integer value to the **frameRate** function. A **frameRate** of 1 will have our **draw** function called every one second.

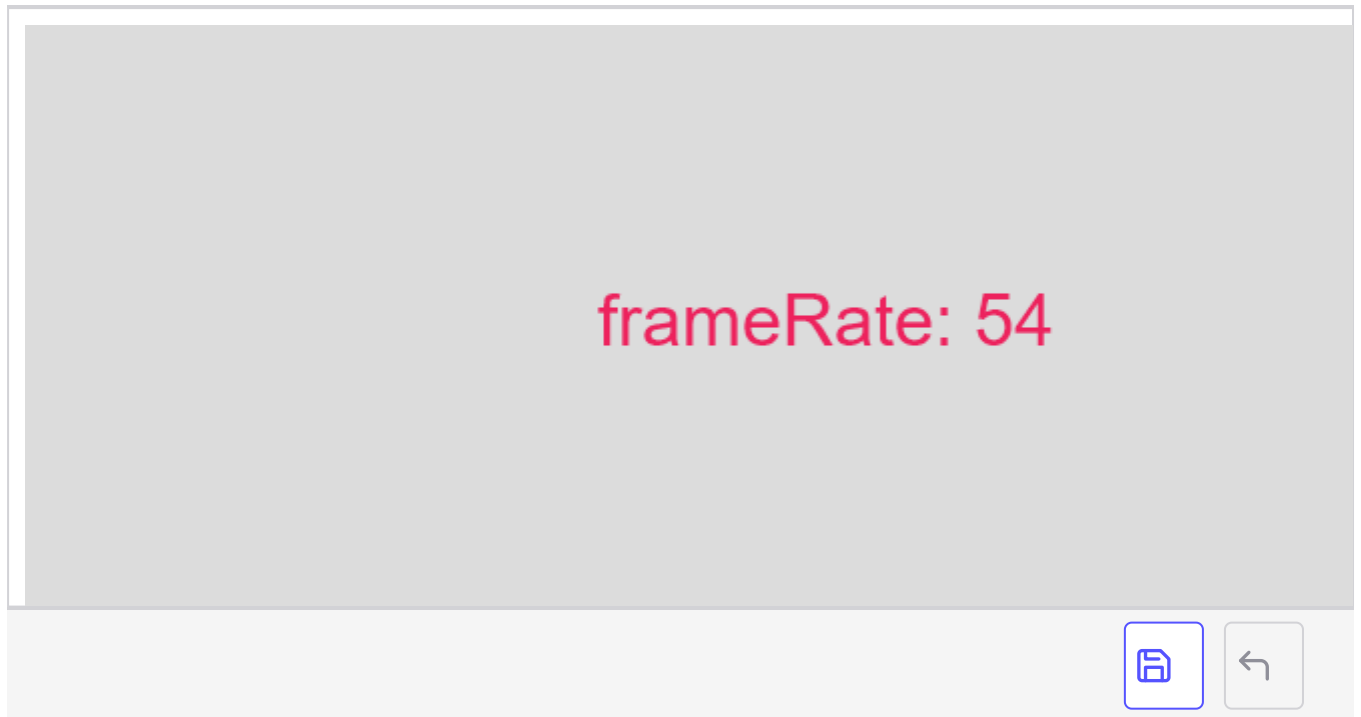
If we didn't want any animation, then we can call a function called **noLoop** inside the **setup** function. This function call will cause the draw function to be called only once.

To summarize, **frameCount** is the number of times the **draw** function is executed throughout the lifetime of a program. **frameRate** is the number of times the **draw** function is executed in a second. If the **frameRate** for a program were 60, the **frameCount** after 3 seconds would be around $60 \times 3 = 180$.

As mentioned earlier we can see what the current frame rate is by calling the **frameRate** function with no arguments. But instead of **console.log**'ing the result, we can actually do much better and display it on screen.

In p5.js, we can use the **text** function to display a value to the screen. **text** function displays the value that is given as the first argument at the x and y positions that are provided as the second and third arguments. With this, we can more easily visualize the frame rate in our program. Please note that the actual result is going to be hard to read at a high frame rate as it fluctuates a lot from one frame to another.

Output
JavaScript
HTML

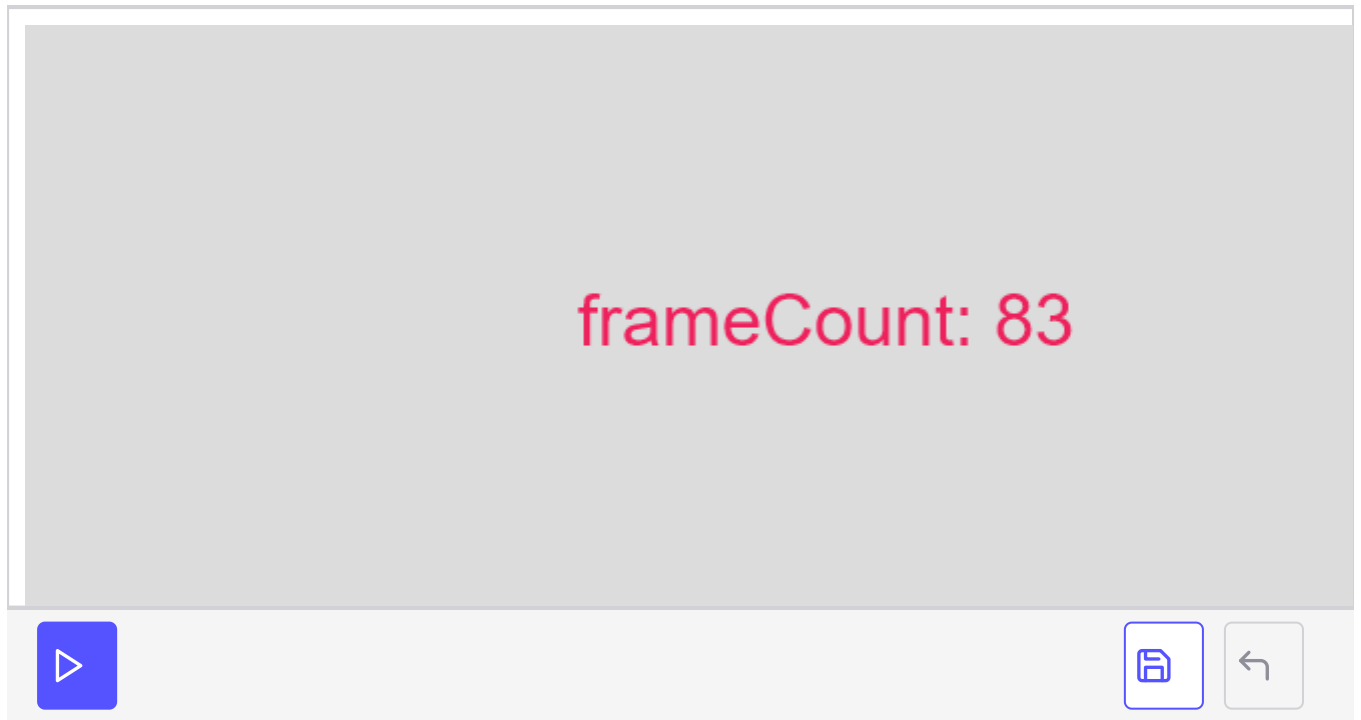


parseInt is a JavaScript function that allows us to convert a decimal number into an integer. It requires a second argument to signify which number base we are working with (which is going to be almost always 10)

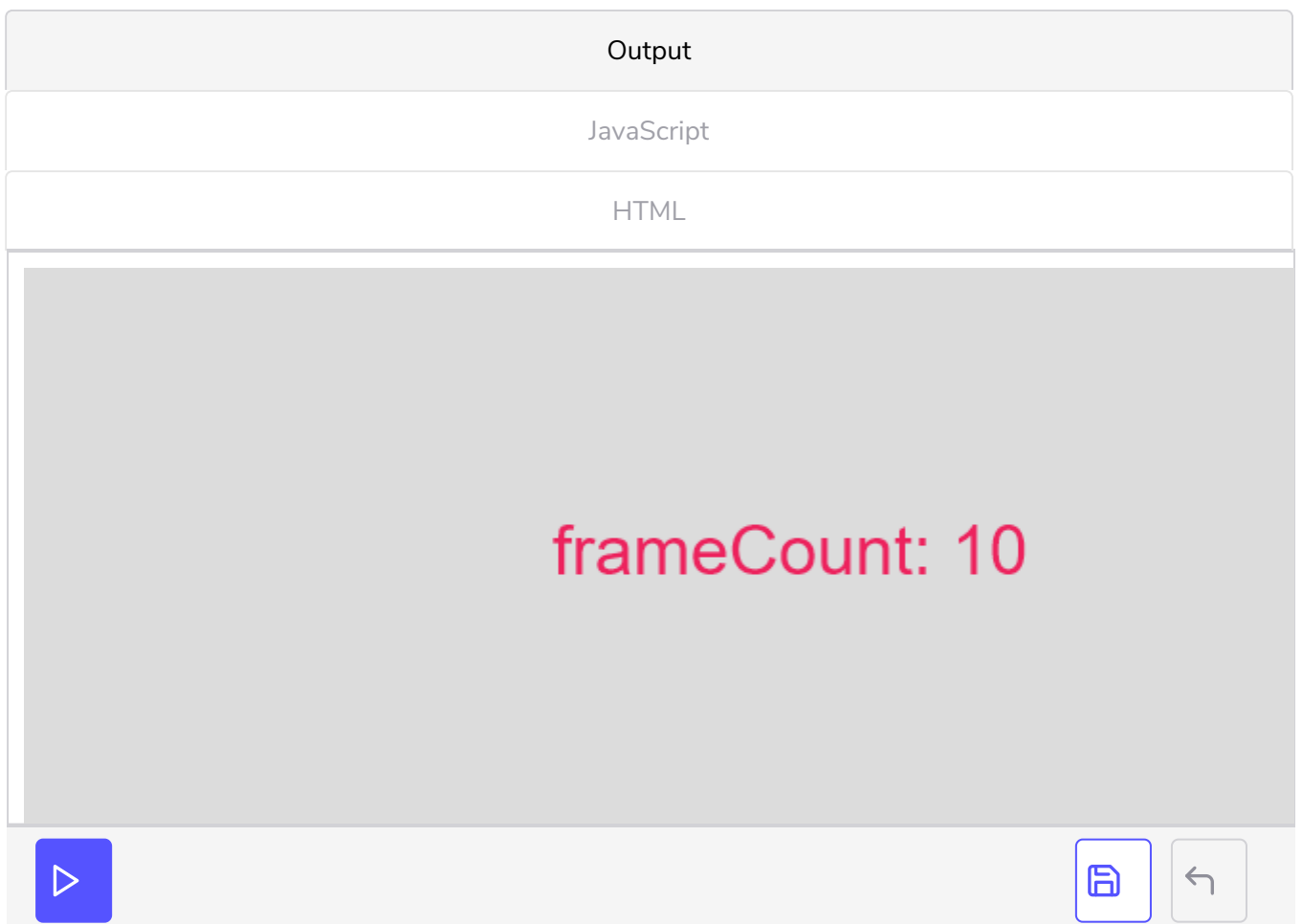
Notice also how we are using a p5.js function called **textAlign** with the arguments **CENTER, CENTER** to be able to align the text horizontally and vertically on the screen. Otherwise the text gets drawn from the top-left corner instead of being centered.

We can also try displaying the **frameCount** variable on screen. As mentioned earlier this is the variable that holds the number of times the **draw** function is called.

Output
JavaScript
HTML

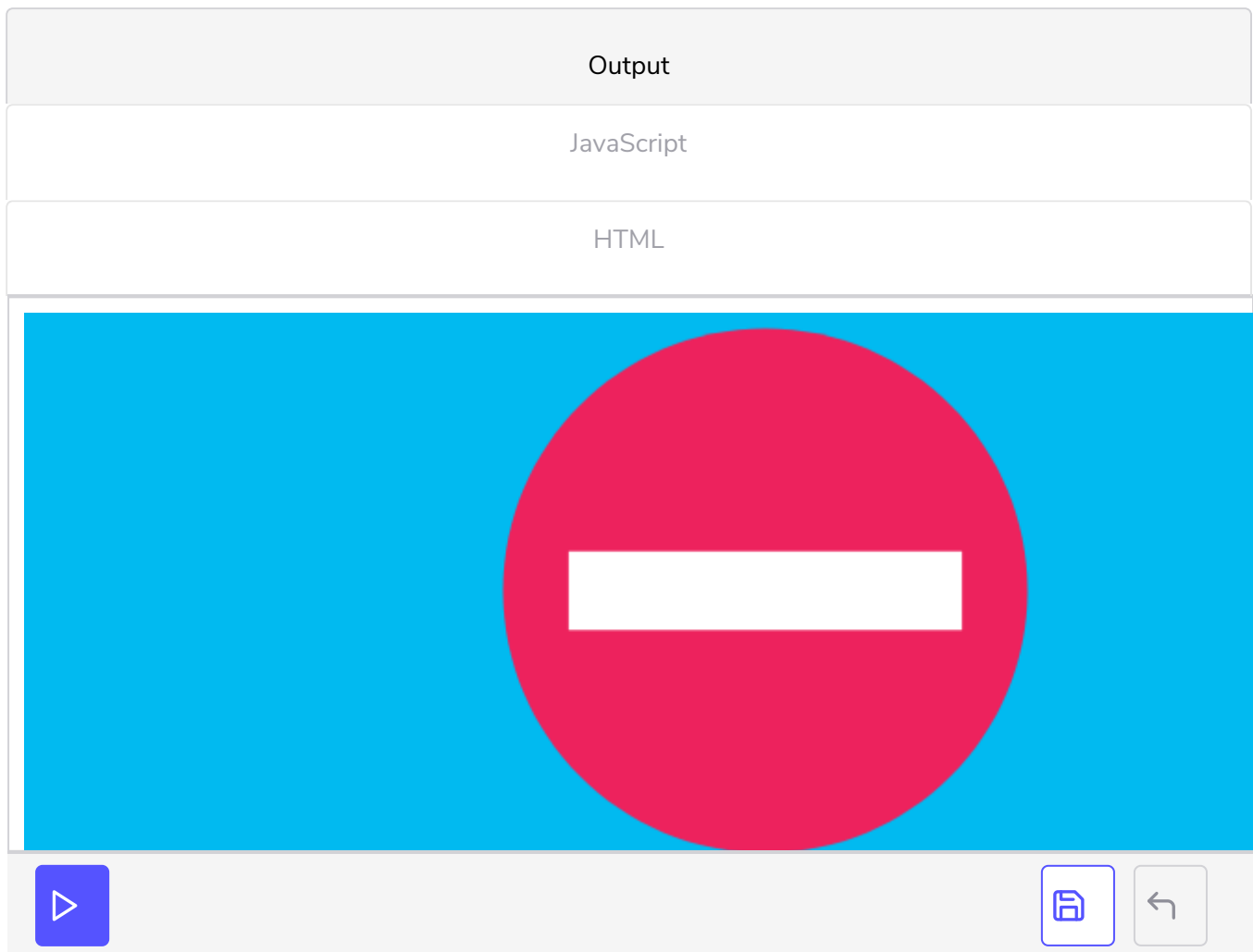


Using the **frameCount** variable, we can quickly have a value at our disposal that increments with each execution of the **draw** function. Notice that the **frameCount** variable will change slower if the **frameRate** is lower.



We could re-write our example from earlier chapter to make use of the built-in **frameCount** variable instead of using our **count** variable.

in `frameCount` variable instead of using our `count` variable.



Conditionals

So far all the programs we wrote executes in a top-to-bottom, linear fashion. But it is quite common in programming to have some parts of the program to execute only if a certain condition is satisfied. For example, using the variable **frameCount**, we are now able to animate a shape across the screen but what if I wanted this animation to start only after a certain frame, like after frame 100?

This can be done using a programming structure called an **if** statement. **if** statement allows us to execute a block of code only if a certain condition is satisfied. How an if statement is written is that we start off with the declaration **if** and inside parentheses that are next to it, we write an expression that should evaluate to **true** or **false**. Next, inside curly brackets right after the **if** statement, we write a block of code that we would like to have executed if the expression that we wrote evaluates to **true**.

```
if (<conditional statement>) {  
    // do something
```

```
}
```

true or **false** are actual values in JavaScript just like how numbers are values. They are just a different type of a value than a **Number** or a **String**. They are referred to as **Boolean** values or a **Boolean** data type. Since **true** and **false** are native JavaScript data types, we could type them without any quotation marks and not get an error.

```
console.log(true);
```



We can't get the same result if we are to type **True** or **False**. Programming languages are particular in how you write things. ****True**** is not equivalent to ****true.**** Moreover ****True**** is not a value that JavaScript recognizes so writing it without quotation marks will result in an error.

```
console.log(True);
```



We can use **comparison** operators to generate **true** or **false** values. Comparison operators allow us to compare two values to each other and as a result they generate a **true** or **false** value based on the result of that comparison. Here are examples of comparison operators. We have the **bigger-than** symbol which compares two numbers and if the number on the left-hand side is bigger than the one on the right-hand side it returns **true**, otherwise it returns **false**.

```
console.log(10 > 2); // would evaluate to true  
console.log(1 > 100); // false  
console.log(100 > 1); //true
```



Bigger or equals returns **true** if the value on the left-hand side is bigger or equal to the value on the right-hand side.

```
console.log(100 >= 100);
```



There is also **smaller** and **smaller or equals** comparison operators.

```
console.log(1 < 10); //true  
console.log(10 <= 10); //true
```



To compare two values to each other to check for equivalency we would use the ‘triple’ equal sign **===**. This is different than what we might be used to from our math classes where the equality operator is a single equal sign operator **=**. But in JavaScript we already use the single equal sign operator as an assignment operation.

```
console.log(1 === 1); //true
```



We can also make a comparison to check if two values are not equal to each other. For this purpose, we use an exclamation mark in front of the equal sign.

```
console.log(1 !== 1);
```



Make sure to try to use the comparison operations that we learned about to see what kind of results they generate in the console.

Let’s look at an example that makes use of **if** structures

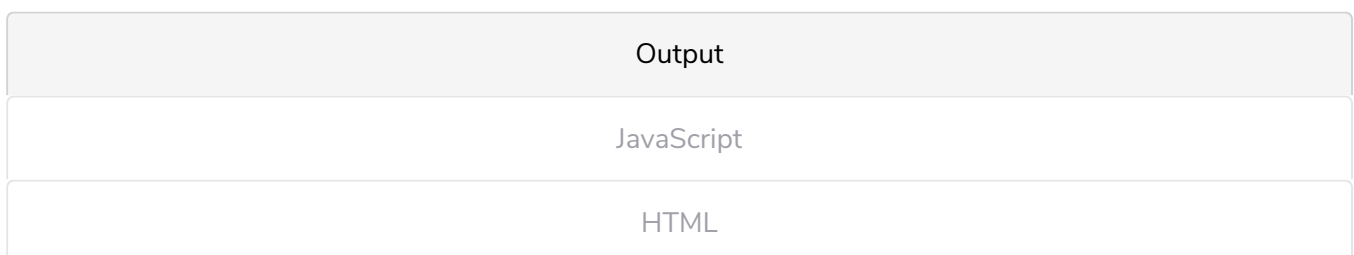
Output



The **if block** will be executed since the expression inside the parentheses will evaluate to **true**. After all, number one is equivalent to number one. We will see the word **TRUE** get displayed on screen because that's what the code inside the **if block** does.

If we were to change the value of the **num** variable to 2, then we won't see anything displayed on screen because this time, the comparison for the **if block** will evaluate to **false** and the conditional will not get executed.

There is this additional structure that we can use with an **if** block that is called an **else** block. An **else** block follows an **if** block and gets executed for every other comparison that is not covered by the **if** block. Let's extend the previous example using an **else** block.





FALSE



Now in this example, the **else** statement would get executed whenever **if** statement is not executed. That is for every value of the **num** variable that is not 1.

By the way, notice how we are repeating ourselves by writing the **text** function twice. We could **refactor** our code to be a bit more concise. Refactoring is, per Wikipedia, the process of restructuring existing computer code—changing the factoring—without changing its external behaviour.

Output
JavaScript
HTML


JavaScript

HTML

FALSE



The problem with this code before refactoring was, if we wanted to change the position of the text, we would need to remember to change it in both text function calls. It might seem easy to remember to do this but even small things like this can actually make code maintenance much harder.

There is one more conditional block that we can add to an if conditional and that is **else if** block. An **else if** block would allow us to handle additional conditions. For example, we can add a couple of **else if** blocks to the example above:

Output

JavaScript

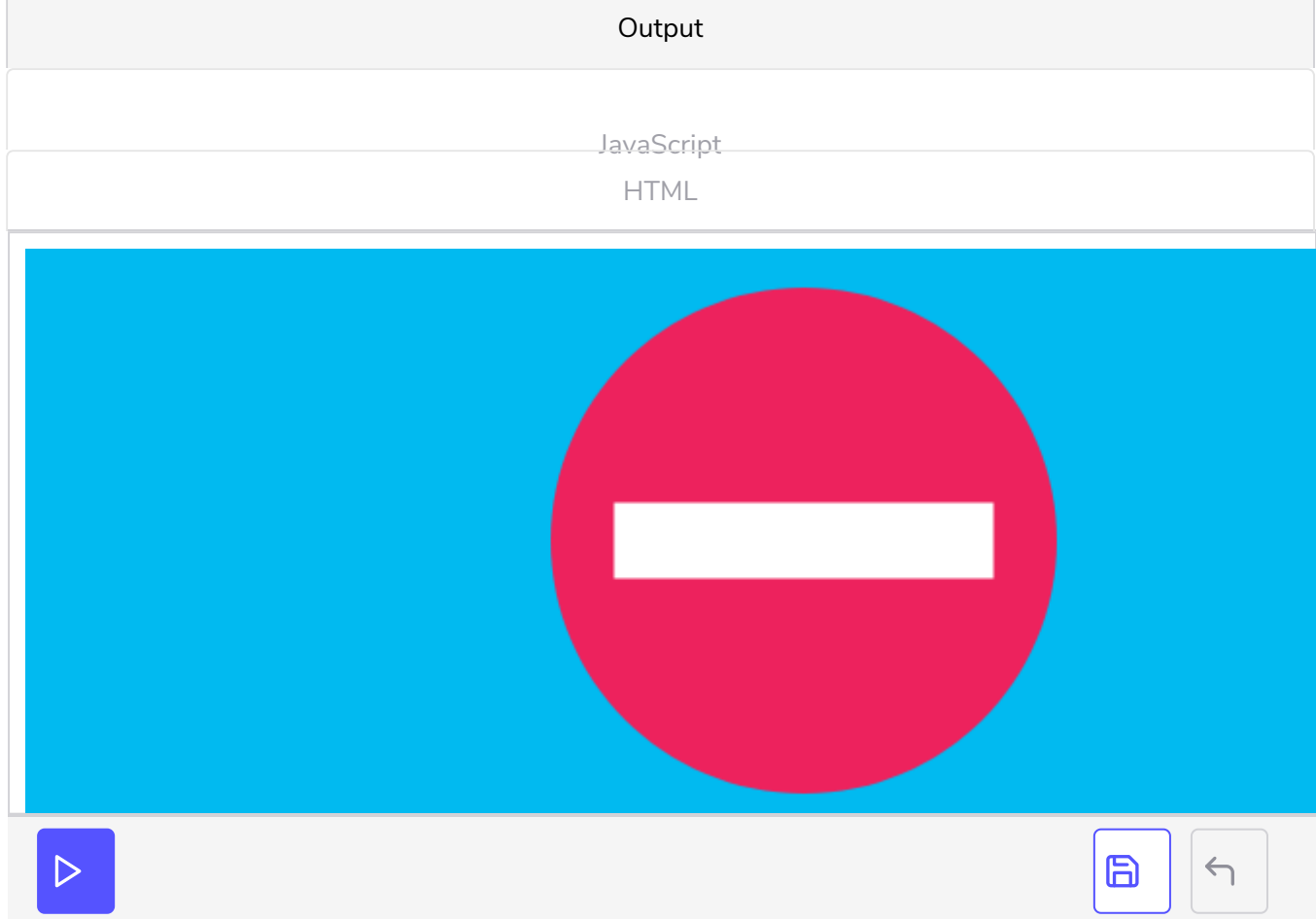
HTML

STILL TRUE



Try changing the value of the **num** variable to see how the code behaves. Using **else if** blocks, we can handle two more specific conditions for the value of **num**.

Using what we learned let's alter the code that we wrote in the previous chapter to make the behaviour of the animation conditional to the **frameCount** variable.



We changed the previous example so that if the **frameCount** value is less than 60, then the shape will be animated using the **frameCount**, if not it will remain static.

What can be a small refactoring that we can do for this code to make it a bit more maintainable? We should probably save the value 60 inside a variable so that we wouldn't have to change it in two places.

We can also combine two logical expressions together to create compound statements by using the **&&** or **||** operators. **&&** stands for **and**. Which allows us to write expressions that will only evaluate to **true** *only if all parts of the conditional statement* is **true**. Say we wanted to animate the shape only if the **frameCount** is bigger than 10 **AND** less than 60. We can combine these two conditions using a compound **and** statement.

```
if (10 < frameCount && frameCount < 60) {  
    size = size + frameCount;  
} else {  
    size = size + 60;  
}
```

|| stands for **OR**. **OR** compound statements returns **true** as long as one part of the conditional statement is **true**. Say we wanted to animate the shape if the

the conditional statement is **true**. Say we wanted to animate the shape if the **frameCount** is smaller than 60 **OR** if the **frameCount** value is bigger than 120. To express this, we could write the following.

```
if (frameCount < 60 || frameCount > 120) {  
  size = size + frameCount;  
} else {  
  size = size + 60;  
}
```

Summary

In this chapter, we learned about the concept of frames and how it helps us to create animated images in p5.js.

We also learned about the p5.js **frameCount** variable that keeps track of how many frames are displayed so far and the **frameRate** function that allows us to set the frame rate for p5.js.

We learned a couple of other p5.js functions such as the **text** function that allows us to draw text to the screen and the **textAlign** function that allows us to align the text that we draw on the screen.

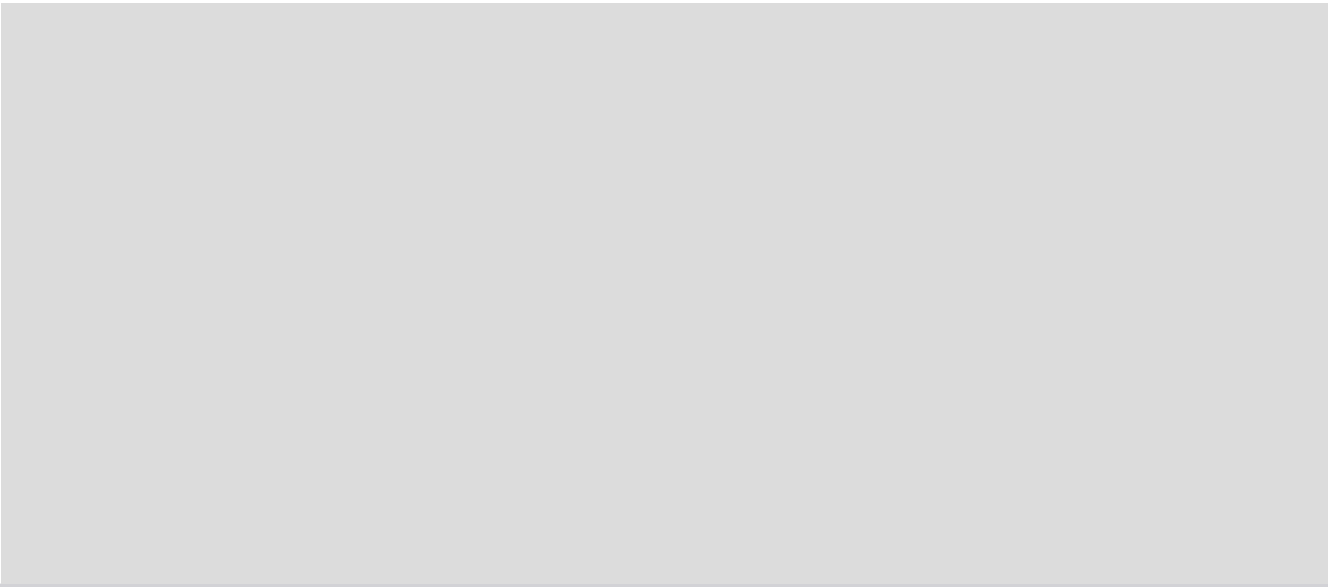
From the JavaScript world, we learned about comparison operators, *Boolean* data types; **true** and **false** and most importantly the **if**, **else if** and **else** conditionals. These structures are commonly used in programming and found in many other programming languages. They allow us to write code that behaves in a little bit more intelligent manner instead of executing blindly from top to bottom.

Practice

Create an animation where five rectangles that are initially off-screen are animated to enter the screen from the left-hand side. They should be moving at different speeds and they should come to a stop just before exiting the screen.

Output

JavaScript



Console

⊘ Clear