MODULE 8: INTERACTING WITH CODE

Agenda:

1. Boolean expressions (10 min)
2. If/else conditions (10 min)
3. Bouncing ball (20 min)
4. More ways to interact! (10 min)
5. Resources (10 min)

Boolean expressions

The most fun and interesting part of making sketches with code is interacting with your drawing in real-time. The possibilities of how you can interact with a sketch are endless, but we'll start with a few more basic ideas.

The main way to create choices for interaction in your code is with a test. Luckily, in the world of computers, tests are much more simple than in real life. You don't have to answer multiple choice questions or write an essay. Every test is a simple true or false question.

Another name for these true/false questions is a "boolean" test. A boolean test looks at the relationship between two numbers and then tells you if the statement is true or false:

10 is less than 15	---> true
7 is equal to 7	---> true
30 is less than 25	---> false

In code, we don't write out "10 is less than 15". We use a few special characters to create "boolean expressions" and test if a statement is true or false. So if we wanted to write those three tests above in code, we would type:

10 < 15	is true
7 == 7	is true
30 < 25	is false

Here are all the options you can use to make a true/false test in code:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

If, else conditions

Boolean expressions are the building blocks of interaction in code. But how do we actually use them to make our code do something interesting?

First, we ask the program a true/false question. Something like "Is 10 less than 15?" If the answer is yes ("true") then choose to do some set of instructions. If the answer is no ("false"), ignore those instructions and continue with the rest of the program.

When you add a true/false question in your code, you are creating a choice for the program: do one thing, or do something else. In our code, we want to say something like "If the mouse is on the right side of the canvas, make the circle red."

Let's see what this looks like in code:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  fill(0);
  if(mouseX > 200) {
    fill(255,0,0);
  }
  ellipse(200,200,50,50);
}
```

Move your mouse left and right over the canvas to see the circle change color. This code should look pretty familiar. The new section here is that line of code that says

```
if(mouseX > 200) {
                        //put the commands you want to run inside the { } block of code
}
```

Your program runs through your code from top to bottom, like always, then it comes to the statement which says “if the mouse’s x-position is greater than 200, do the commands inside this block of code. (i.e. make the fill color red)” And if the mouse’s x-position is less than 200, the program will ignore those commands and continue with the rest of the code (i.e. drawing an ellipse).

This concept can be expanded to include different instructions if the boolean expression (i.e. the true/false question) you’re testing is false. It looks like this:

```
if ( boolean expression) {
        //the code you want to run if the boolean expression is true
}
else {
        //the code to run if the boolean expression is false
}
```

As a simple example, try entering this in the code editor:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  if(mouseX < 200) {
    background(0);
  }
  else {
    background(120);
  }
}
```

Now move the mouse left and right across the canvas to see the background change color.

You can use if/else statements to test the properties of any variable you’ve created. If the statement is true, whatever code you put inside the “if { }” block will be run. If the statement is false, the code you put inside the “else { }” block will run instead. This can create interesting possibilities for animation and interaction, as we will see in the next sections.

Bouncing ball

Let’s take a look at some of these “if” statements in action. We’ll start with the ellipse moving across the screen from Module 7. This time we’ll use a variable for “speed” instead of adding a fixed number to “x” each time we loop through draw().

```
var x = 0;
var speed = 2;

function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  ellipse(x,200,50,50);
  x = x + speed;
}
```

After a few seconds, you’ll notice that the circle completely leaves the screen. What if we want the circle to “bounce” off the edge of the screen and go back in the other direction? How would we do this?

When the ellipse’s position is greater than the width of the canvas, we want the ellipse to “change directions” and move the other way. So instead of increasing the x variable, we want it to decrease.

Luckily it’s easy to do this without adding much code. We just need to make our speed variable a negative number once the circle reaches the edge. To make a positive number negative, multiply it by -1. (And to make a negative number positive, you also multiply by -1.) Try this:

```
var x = 0;
var speed = 2;

function setup() {
  createCanvas(400, 400);
}
```

```
function draw() {
 background(220);
 ellipse(x,200,50,50);

 if(x > width) {
        speed = speed * -1;
 }

 x = x + speed;
}
```

p5.js has a useful built-in variable called “width” we can use in this case. If the ellipse reaches the edge of the screen (i.e. it’s x position is greater than “width”), we change the direction of its speed by multiplying by -1. When it reaches the edge, it changes direction!

Now we need to make sure the ellipse changes direction if it reaches the left edge of the screen. We can do this by adding another “if statement” for when “x” is less than 0.

```
var x = 0;
var speed = 2;

function setup() {
 createCanvas(400, 400);
}

function draw() {
 background(220);
 ellipse(x,200,50,50);

 if(x > width) {
        speed = speed * -1;
 }
 if( x < 0) {
   speed = speed * -1;
 }

 x = x + speed;
}
```

Now the ball will bounce back and forth forever! One thing that is nice about using a variable for “speed” is that if you want the ball to move faster or slower just change the value of speed at the top of your code. Try a value like 10 to see what happens.

More ways to interact!

There are a few more tools we can use to interact with our drawings in real-time.

Let's say we want to change the color of a shape whenever you click a button on the mouse. There is a built-in variable called "mouseIsPressed" that we can insert into an "if statement" to do that. Try this code:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);

  if(mouseIsPressed) {
    fill(200,0,100);
  }
  else {
    fill(0);
  }

  rect(100,100,70,50);
}
```

Note that you have to type "mouseIsPressed" exactly how it appears with upper- and lower-case letters or it won't work.

Now instead of mouseIsPressed, try keyIsPressed. It should change color whenever you press a key on the keyboard.

Resources

The p5.js language has so many more tools to make your drawings interactive. In this last section, we'll see how you can learn more of these tools.

The best place to learn about all the functions and tools that p5.js has to offer is at the reference website:

https://p5js.org/reference/

Here you will find all of the tools of the language and examples on how to use these tools in your code. For example, scroll down until you see “quad()” in the Shape section. (You should recognize the other functions you know like ellipse() and triangle()).

When you click on quad() it takes you to a page that shows you how to use that function in your code. On all of these reference pages, you’ll see an example, followed by a description of what the function does, followed by a list of the parameters used by that functions. Notice that quad() is very similar to triangle(), it just uses a 4th set of x-y coordinates to make a shape.

Click on a few more reference pages to see what they look like. A lot of these functions won’t make sense to you yet, but you can learn more about them in the “Tutorials” and “Examples” section of the website.

Another great way to learn more is through video tutorials. Daniel Shiffman has an excellent YouTube channel called “The Coding Train” that covers a huge range of coding topics. He writes code in p5.js or its sister language, Processing, so there is a lot you can learn from him.

Start with his “Foundations of Programming in Javascript” series (did you know that you’ve actually been coding in JavaScript this whole time? p5.js is built on top of JavaScript, so you have a head start on programming for the web.)
https://www.youtube.com/playlist?list=PLRqwX-V7Uu6Zy51Q-x9tMWIv9cueOFTFA

Some of these videos will be review, but you’ll quickly learn new concepts like random(), map(), loops, and arrays.

Exercise: Make your character interactive

In this exercise you’ll add interactive elements to your character. Maybe it will change color when you click the mouse, or even change shape if you press a key on the keyboard.

You can also try to make something completely new. Just use some if/else statements to make your drawing interesting, animated, and interactive.