# (Conditional) Interaction

## CF1

Gabe Johnson | Sept 12 2024

# Today: Slides, then live coding

# Linear Script? Or Choose-Your-Own-Adventure?

You can write programs that always do the same thing, regardless of input.

Boring!

But with the cunning use of *conditional statements* you can have your code behave differently depending on user input, sensor data, or the contents of files.

# Simple if-statement

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

Here’s how to read it:

If the mouse’s X position is greater than half the window width, then set the fill color to red.

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

The anatomy of an if-statement is like this:

```
if (someCondition) {
  …
}
```

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

It always starts with a lower-case ‘if’ keyword:

```
if (someCondition) {
  …
}
```

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

Then there are parentheses that contain a boolean expression that determines if the body of the if-statement will run.

```
if (someCondition) {
  …
}
```

In our example here, that boolean expression is: `mouseX > windowWidth / 2`

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

Following the parentheses we have curly braces. Like parens, these always come in pairs: opening brace and a closing brace.

```
if (someCondition) {
  …
}
```

```
if (mouseX > windowWidth / 2) {
  fill(“red”);
}
```

The stuff inside the curly braces is the code that will run *only if the condition is met*. It can be a single statement, or it could be something complicated.

```
if (someCondition) {
  fill(“red”);
}
```

In our example here, we just set the fill color.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
}
```

This is an augmented form of an if-statement that has an *else* clause. If the boolean expression is not met, we run the curly braced code following the *else* keyword.

In this case, we set the fill color to blue.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
}
```

One nice thing about structuring code like this is that you can be sure that *exactly one* of the clauses will run. So if you know you must set the fill color to either red or blue, this is a good way to do it.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else if (mouseY > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
}
```

An if-statement can also have “else if” clauses, like the one bolded here. This will run only if a previous conditional clause didn’t match.

The idea in this code is to set the fill color to red if the mouse is either in the right half, or the bottom half of the canvas. Otherwise, it sets the fill color to blue.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else if (mouseY > windowWidth / 2) {
    fill("red");
}
```

It is also possible to have an if-statement that contains an “else if”, but doesn’t have a final “else” statement. Use this if you don’t want to have default behavior that is run if none of the given conditions pass.

E.g. in this code, we might not even change the fill color.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
} else if (mouseY > windowHeight / 2) {
  fill(“red”);
}
```


Here’s something you *can’t* do. This code is broken.

An if-statement can have zero or one *else* clauses, and if it has one, it *must* be the last clause.

The above code is broken because it includes an *else* clause that isn’t the last one.

```
if (mouseX > windowWidth / 2) {
    fill("red");
} else if (mouseY > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
}
```

Same Behavior!

The code on the left does the same thing in the first two if-clauses. We can combine that logic using a “boolean or”.

The code below does exactly the same thing as the code at left.

```
if (mouseX > windowWidth / 2 || mouseY > windowWidth / 2) {
    fill("red");
} else {
    fill("blue");
}
```

# Boolean Expressions

## True and False, and Truthy and Falsy

A boolean expression is code that evaluates to true or false: “is it raining?” Or “is the mouse X value greater than 50?” These are yes/no questions. We’ve been looking at one that looks like this:

```
mouseX > windowWidth / 2
```

# Boolean Expressions

## True and False, and Truthy and Falsy

There are many other forms a boolean expression can take.

```
isMousePressed
```

There’s no equality operator, no greater-than-or-equal, etc. Variables can be interpreted as true or false on their own, if it makes sense.

# Boolean Expressions

## True and False, and Truthy and Falsy

Since p5.js is just Javascript, we inherit all the weirdnesses of that language.

One major weirdness (that is also extremely useful) is how it can interpret different values as being true or false, even if the data it is examining isn't even boolean. When we do this, we say that the expression is **truthy** or **falsy** - these are actual words and I am not making them up!

# Boolean Expressions

## True and False, and Truthy and Falsy

Examples of truthy and falsy

```
if ("sure whatever") { }
```
👈 truthy!

```
if (mouseX) { }
```
👈 truthy if mouseX isn't zero

```
if (someData.length) { }
```
👈 truthy if this list has nonzero length*

* we haven't covered lists yet, don't worry

# Boolean Expressions

## Combining with And and Or

In Javascript: we spell ‘and’ with two ampersands: &&

And we spell ‘or’ with two pipe characters: | |
(probably far right side of your keyboard above the back slash).

# Boolean Expressions

## Combining with And and Or

Use these two operators to combine individual boolean expressions to make more sophisticated expressions. Be careful to not be too clever, because it can be hard to read.

```
if (mouseX > windowWidth / 2 || mouseY > windowHeight / 2) {
  // this code runs if the mouse is in the right half,
  // or if it is on the bottom half.
}
if (mouseX > windowWidth / 2 && mouseY > windowHeight / 2) {
  // this code runs if the mouse is in the lower right corner
}
```

# Boolean Expressions

## Combining with And and Or

You can combine any number of *and* and *or* expressions. The code below might be read as “if it is sunny and I have free time, or if today is Saturday, I’m going to go have fun.

```
if (sunny && freeTime || todayIsSaturday) {
    goHaveFun()
}
```

# Boolean Expressions

## Combining with And and Or

Beware of this land mine! When you have a boolean expression that involves the *or* operator, the language will be “smart” and stop evaluating the expression when it knows the answer.

In this example, say that thingA() returns false. It will then run thingB() - which in this story, returns true. Javascript now knows that this whole expression is true, so it won’t even invoke thingC().

```
if (thingA() || thingB() || thingC()) { ... }
```

# Boolean Expressions

## Combining with And and Or

There is a similar land mine for *and* expressions: it has to evaluate everything to know if the expression is true, but it only has to find a single false before it knows that the expression is false.

```
if (thingA() && thingB() && thingC()) { ... }
```