Diploma in Computer Studies September 2021

Welcome to Creative Computing

DCR2284

Learning Objectives

- ❑ At the end of the course, students will be able to:
  - ❑CO1: Describe the creative concepts in mathematics and computing.
  - ❑CO2: Explain the importance origins of geometry to develop motion, images and sound.
  - ❑CO3: Build the Processing application to construct shapes and objects.
  - ❑ CO4: Write the coordinate transformations for motions using Processing..

Conditionals (2-b)

Conditionals

or

What if?

You *can* handle the truth!

- The ideas of `true` and `false` are central to our general understanding of the world
- Of course, in reality, working out what's `true` and what's `false` is quite difficult
- But in programming, things are a *lot more certain*
- Probably something to do with all those 1s and 0s floating around in there...

Decisions, decisions....

- One way we use the ideas of `true` and `false` is that we use them to *make decisions*
- We say things like "*if* the metro stays broken down for two more minutes *then* I'm going to get off and walk home"
- That is, we're going to *do something* based on whether a statement turns out `true` or `false`

From context to action

- This idea of going from *knowing something* to *doing something* is at the heart of what makes programming and software interesting
- This is a huge part of what makes programs *react* to context, instead of just doing the same thing every time
- They might react to the weather, or the date, or the keyboard, or something else...
- ... but in all cases they need to use these "*if* this *then* I'll do that" kinds of structures

Some ifs...

- If the player presses the spacebar, do an amazing skateboard trick
- If the date is the 25th of December, play a Christmas Carol
- If the weather is cloudy, making the interface grey and hard to read
- If the user is shouting, making the screen vibrate
- AND SO ON!

What is `true`? What is `false`?

- In Processing we talk about things that can be `true` or `false` as *conditional expressions* and they're often kind of like maths:

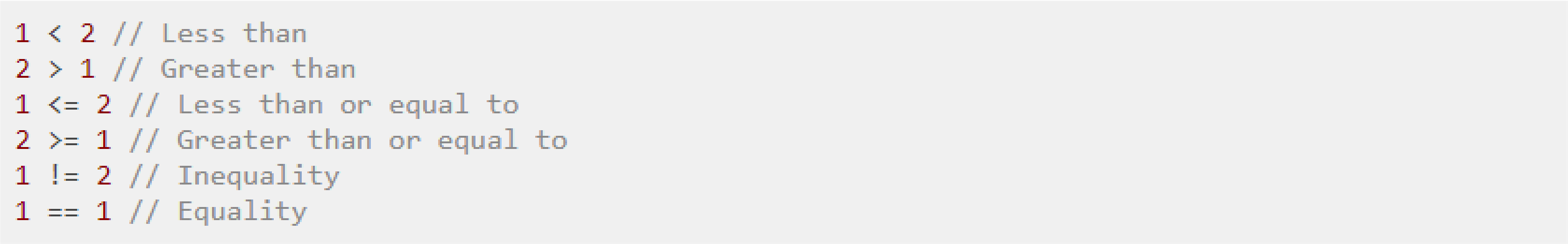
`23 < 24` is `true`
`1 + 1 == 3` is `false`

- We use these kinds of expressions to *check what's happening* in our code, and then react to it
- Usually it's better if we use variables!

`avatarX > width` would be `true` if the avatar has gone off the right side of the screen...

Conditional operators

- We make *conditional expressions* with *conditional operators* and the main ones are:



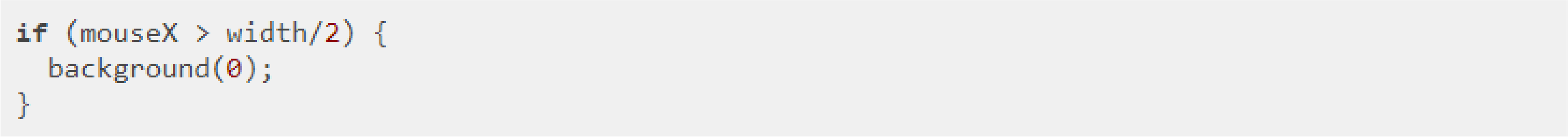
```
1 < 2 // Less than
2 > 1 // Greater than
1 <= 2 // Less than or equal to
2 >= 1 // Greater than or equal to
1 != 2 // Inequality
1 == 1 // Equality
```

- See? Maths. All the above are `true`
- Note that this means we are very often checking *numbers* with these operators (like `int` and `float`)

Getting iffy

- So how do we *use* these conditional expressions to check what's going on in our program?
- We use `if` statements
- An `if` statement *checks* if a condition is `true`, and will do something based on the result

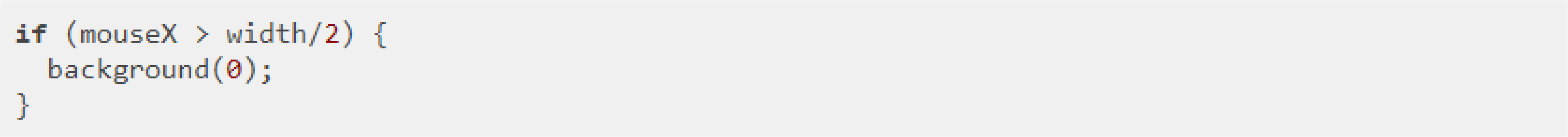
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- This is an `if` statement that checks whether the mouse is to the right of the middle of the window
- And if the mouse *is* over there, it makes the background of the window black
- Let's break it down...

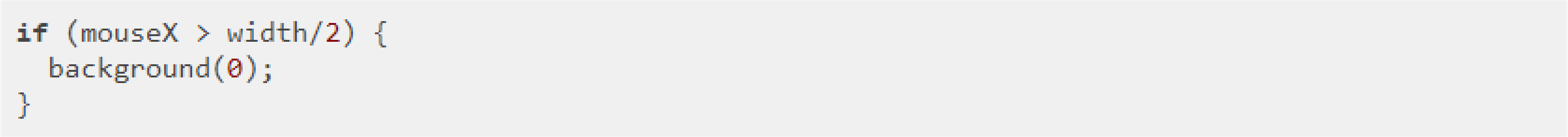
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- First we have the word `if`
- This is what tells Processing we're going to use an `if` statement
- It kind of means we're about to *ask a question*

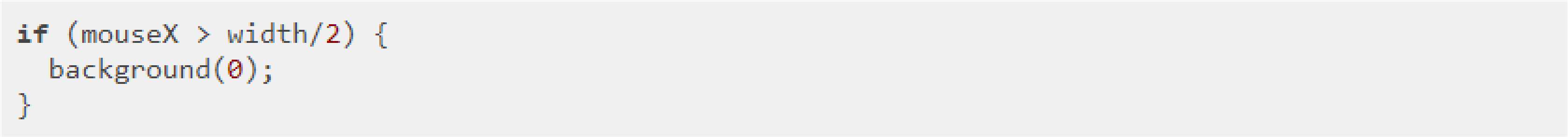
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- Next we have `(`, an opening parenthesis
- We've seen this before in *functions* where we were saying "I'm going to tell you the parameters"
- This time it's similar, but it means "I going to tell you the *condition* I want you to check"
- So parentheses tend to mean "I'm going to give you information to help you do your job"

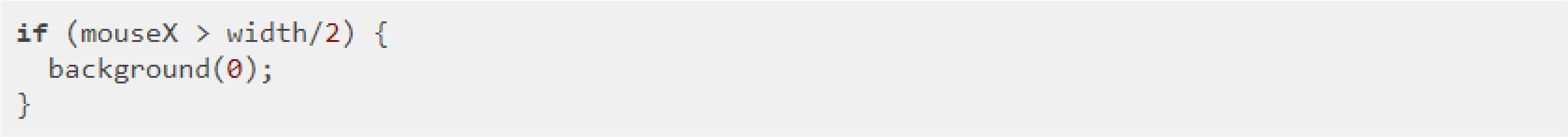
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- Next we have `mouseX > width/2`, our *condition*
- This is *the thing we want to check*
- We want to know if it's `true` or `false`
- In this case we're asking "is the mouse's x coordinate greater than half the width of the window?"

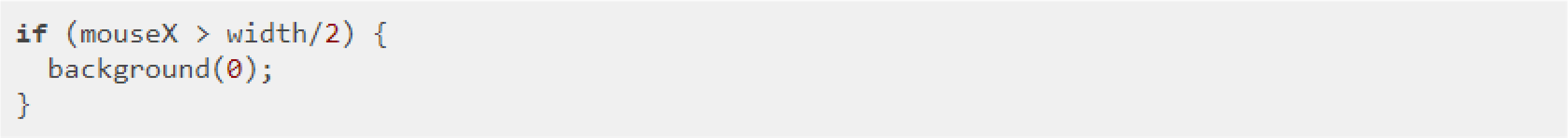
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- Then we have `)`, a closing parenthesis
- That is, "I'm done telling you what to check!"

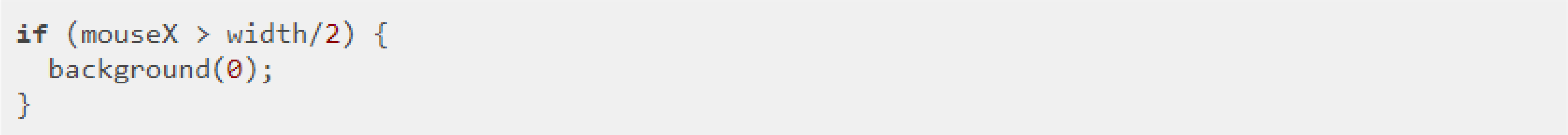
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- Now we have `{`, an opening curly bracket!
- Like in a *function* this means "Now I'm going to tell you *what to do*!"
- But in this case it specifically means "Now I'm going to tell you what to do *if that condition is `true`*"

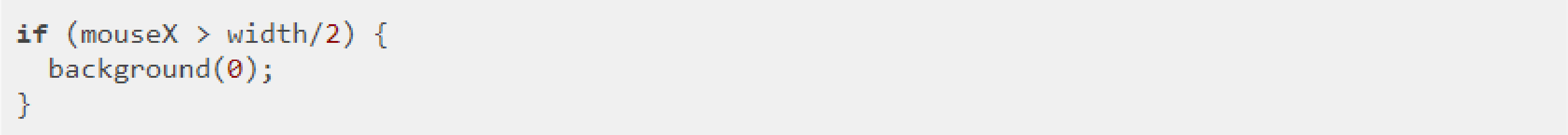
A basic `if` statement



```
if (mouseX > width/2) {
  background(0);
}
```

- Now we have the actual code we want to run *if the condition is true*
- In this case we just want to make the background black with `background(0);`
- But we could do *anything* in here!
- *ANYTHING!!!*
- And we can have *as many lines of code as we want in here*

A basic `if` statement



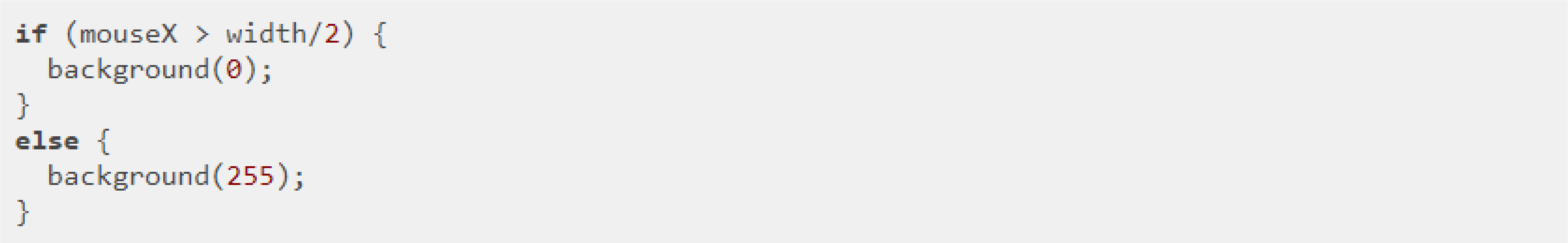
```
if (mouseX > width/2) {
  background(0);
}
```

- Finally we have `}`, a closing curly bracket
- That is, "I'm done telling you what do if that condition is `true`"
- As you can see, we use *curly brackets* to surround *blocks* of code that belong together
- In this case the curly brackets are around *all the code to run if the condition is true*

What `else`?

- Of course we might not *only* want to react to the condition being `true`
- We may also want to do something only if it's `false`
- And for this we can extend the `if` statement with an `else` to do just that

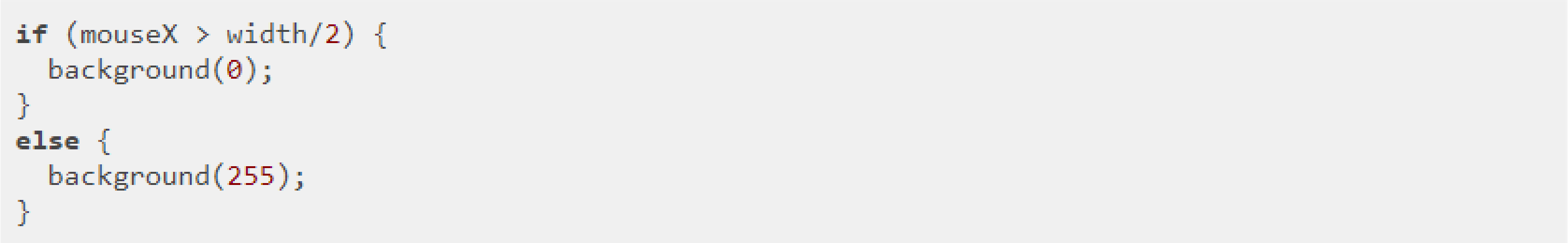
An `if` `else` statement



```
if (mouseX > width/2) {
  background(0);
}
else {
  background(255);
}
```

- Here we have the same `if` statement, but now with a bit extra after the closing curly bracket of our original `if`

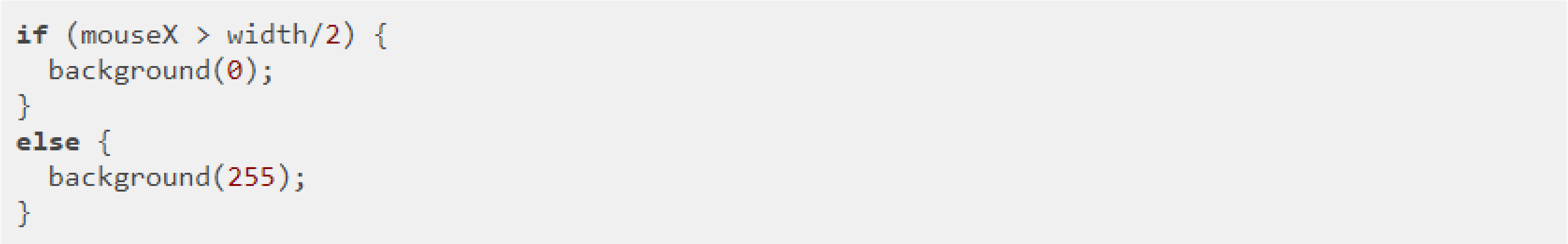
An `if else` statement



```
if (mouseX > width/2) {
  background(0);
}
else {
  background(255);
}
```

- First we have the word `else`
- This signals that we're going to deal with the case where the condition turns out to be `false`
- In this case, that means when `mouseX <= width/2`
- That is, when the mouse is to the *left* of the middle of the window

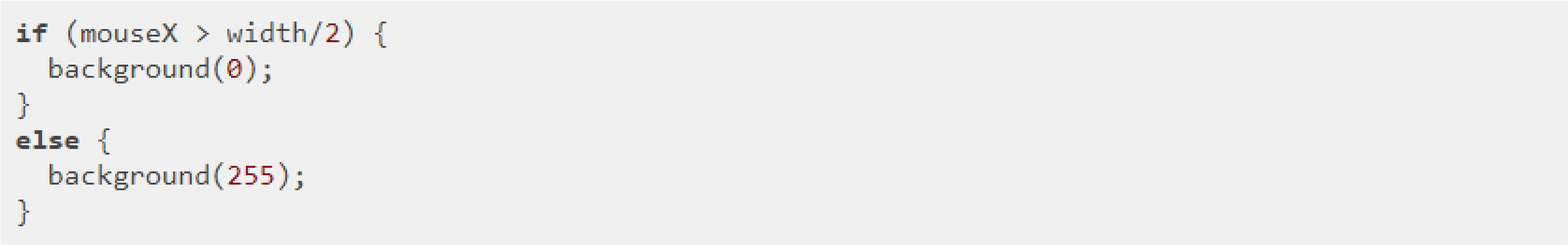
An `if` `else` statement



```
if (mouseX > width/2) {
  background(0);
}
else {
  background(255);
}
```

- Then we have our friend {, the opening curly bracket
- Which means "Now I'm going to tell you what to do if the condition is false"
- Note that we *don't* need parentheses, because we don't need *new information* here
- We're still relying on the information in the original condition

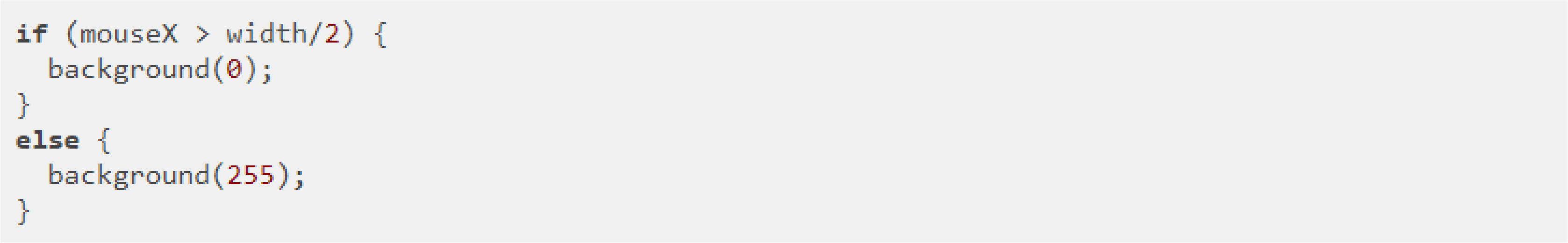
An `if` `else` statement



```
if (mouseX > width/2) {
  background(0);
}
else {
  background(255);
}
```

- Then we have the code we want to run when the condition is `false`
- In this case `background(255);` to make the background white

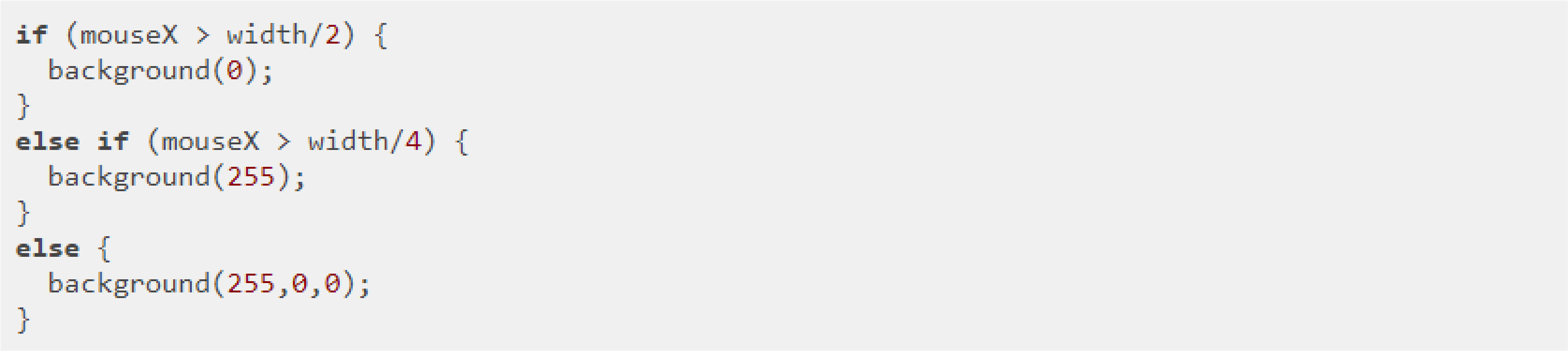
An `if` `else` statement



```
if (mouseX > width/2) {
  background(0);
}
else {
  background(255);
}
```

- Finally we have }, the closing curly bracket
- This says "I'm done telling you what to do if the condition is false"

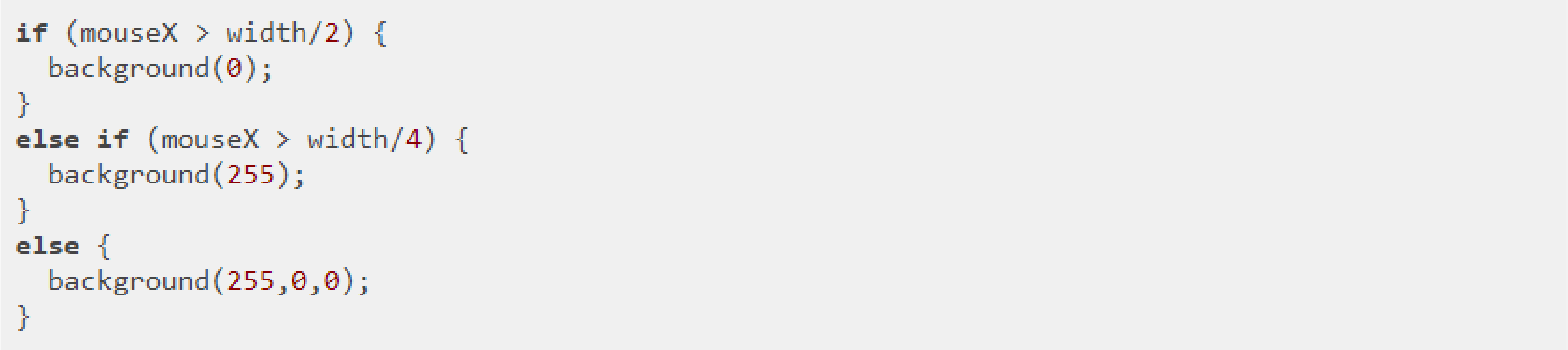
An `if else if` statement



```
if (mouseX > width/2) {
  background(0);
}
else if (mouseX > width/4) {
  background(255);
}
else {
  background(255,0,0);
}
```

- So we can have *another if* after our else that will check *another condition*
- Note that it will *only* check that second condition *if* the first condition is `false`, right?
- And note we can still have an `else` at the end that handles if *both* the conditions are `false`

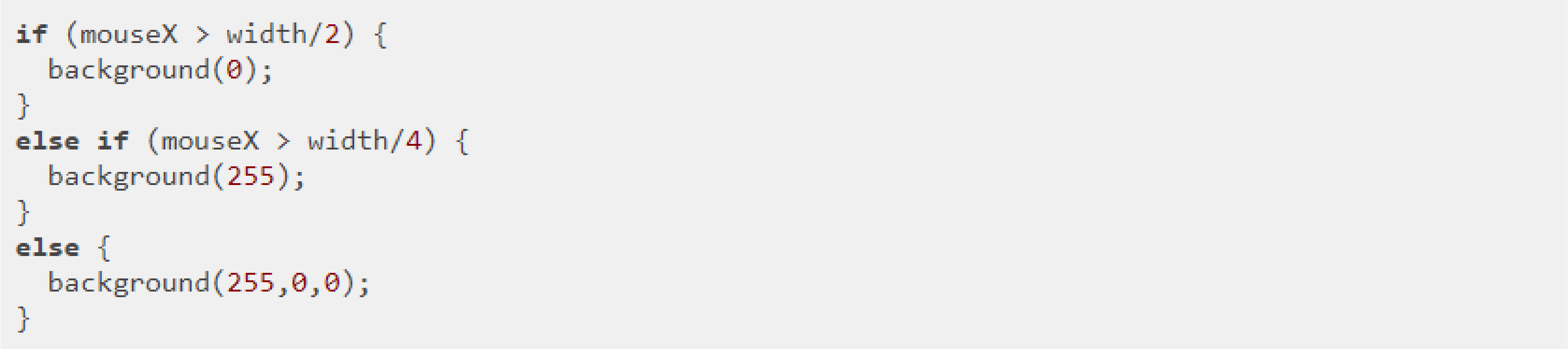
An `if else if` statement



```
if (mouseX > width/2) {
  background(0);
}
else if (mouseX > width/4) {
  background(255);
}
else {
  background(255,0,0);
}
```

- What do you figure this will do?

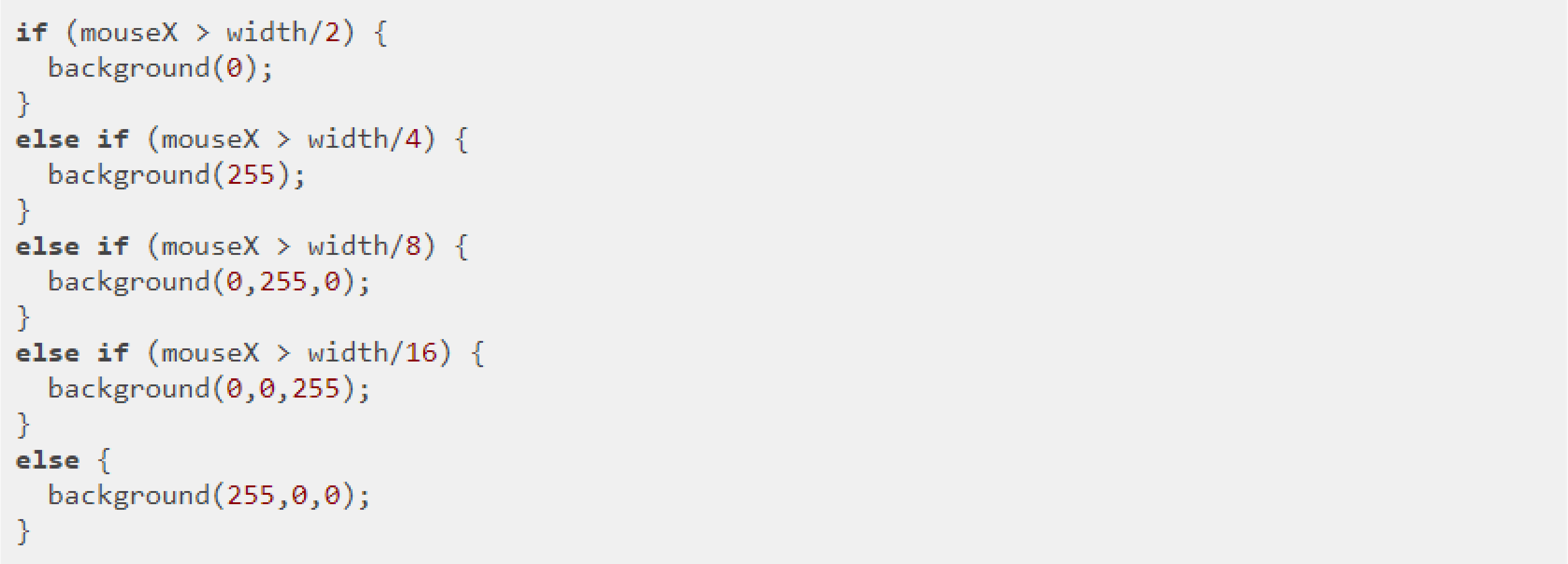
An `if else if` statement



```
if (mouseX > width/2) {
  background(0);
}
else if (mouseX > width/4) {
  background(255);
}
else {
  background(255,0,0);
}
```

- What do you figure this will do?
- Yeah, it will make the background *black* if the mouse is in the right half of the screen, *white* if it's in the right half of the left half of the screen, and red if it's in the left half of the left half of the screen...
- Interesting how the code is kind of *easier* to read than that.

if else if else if else if...



```
if (mouseX > width/2) {
  background(0);
}
else if (mouseX > width/4) {
  background(255);
}
else if (mouseX > width/8) {
  background(0,255,0);
}
else if (mouseX > width/16) {
  background(0,0,255);
}
else {
  background(255,0,0);
}
```

- This can go on for a while!

Cold, hard logic

- Sometimes we need to check more complicated ideas than we can express in a math-style condition
- To help out, programming uses *logic operators*

&& means AND
| | means OR
! means NOT

- Kind of nice, since this is *literally* how computers work at the circuit level!
- But how does this work in code?

Logically speaking...

```
(condition1 && condition2)
```

This is `true` if *both* `condition1` and `condition2` are `true`, otherwise it is `false`.

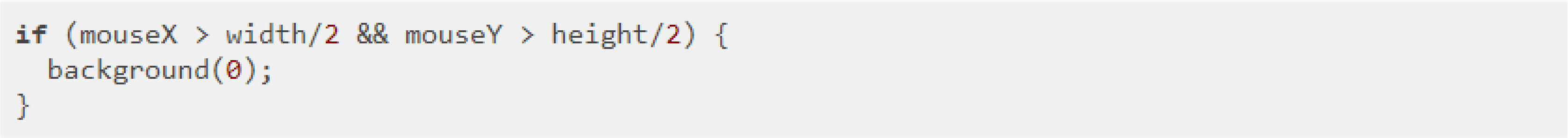
```
(condition1 || condition2)
```

This is `true` if *either* `condition1` *or* `condition2` are `true`, otherwise it is `false`.

```
(!condition)
```

This is `true` if `condition` is `false`, and `false` if it's `true`.

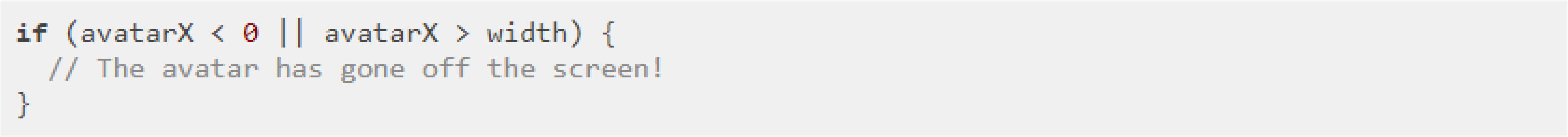
In practice...



```
if (mouseX > width/2 && mouseY > height/2) {
  background(0);
}
```

- We can recreated the *nested* *`ifs`* from before using && this time
- The background will be black if the mouse is in the right half AND in the bottom half of the window

In practice...



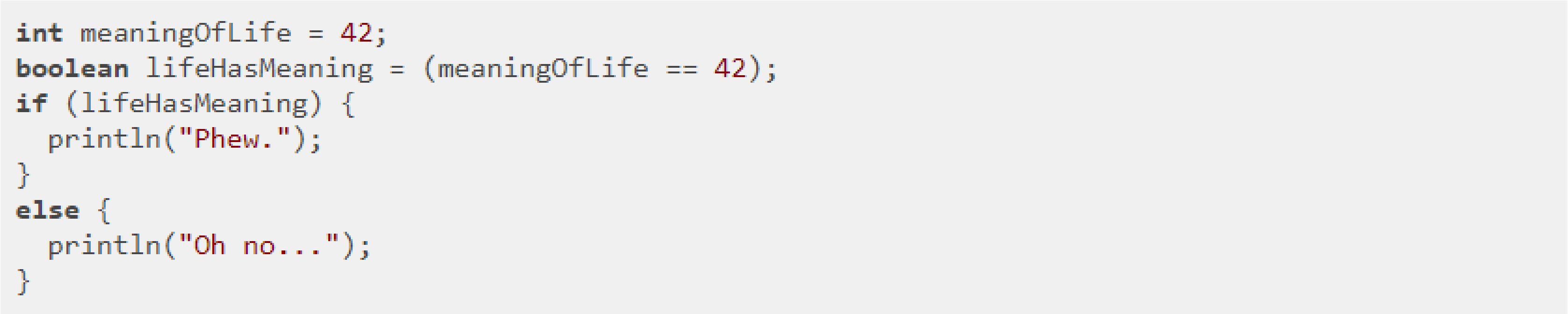
```
if (avatarX < 0 || avatarX > width) {
  // The avatar has gone off the screen!
}
```

- We can check multiple possibilities in one line now
- This checks whether the avatar is *either* off the left edge of the window *or* off the right edge of the window
- Maybe it should... die for this!

```
int avatarX = 0; // Avatar's x location
int avatarY = 0; // Avatar's y location
int avatarSize = 20; // Avatar's size
int avatarVX = 5; // Avatars x (horizontal)
velocity
void setup() {
size(500,500); // Set the size of the
window!
}
void draw() {
background(255); // Fill the background for
animation effect
avatarX = avatarX + avatarVX; // Move the
avatar's location by velocity
rect(avatarX,avatarY,avatarSize,avatarSize)
; // Draw the avatar
// Check if the avatar has gone off the
screen...
if (avatarX < 0 || avatarX > width) {
// If it has, reverse its velocity
avatarVX = -avatarVX;
}
}
```

`true` and `false` are booleans

- We've already seen variables can have a *type* like `int` and `float` and `String`
- There is another type called `boolean` we can use to store either `true` or `false`
- Which means we can track a condition in a variable!



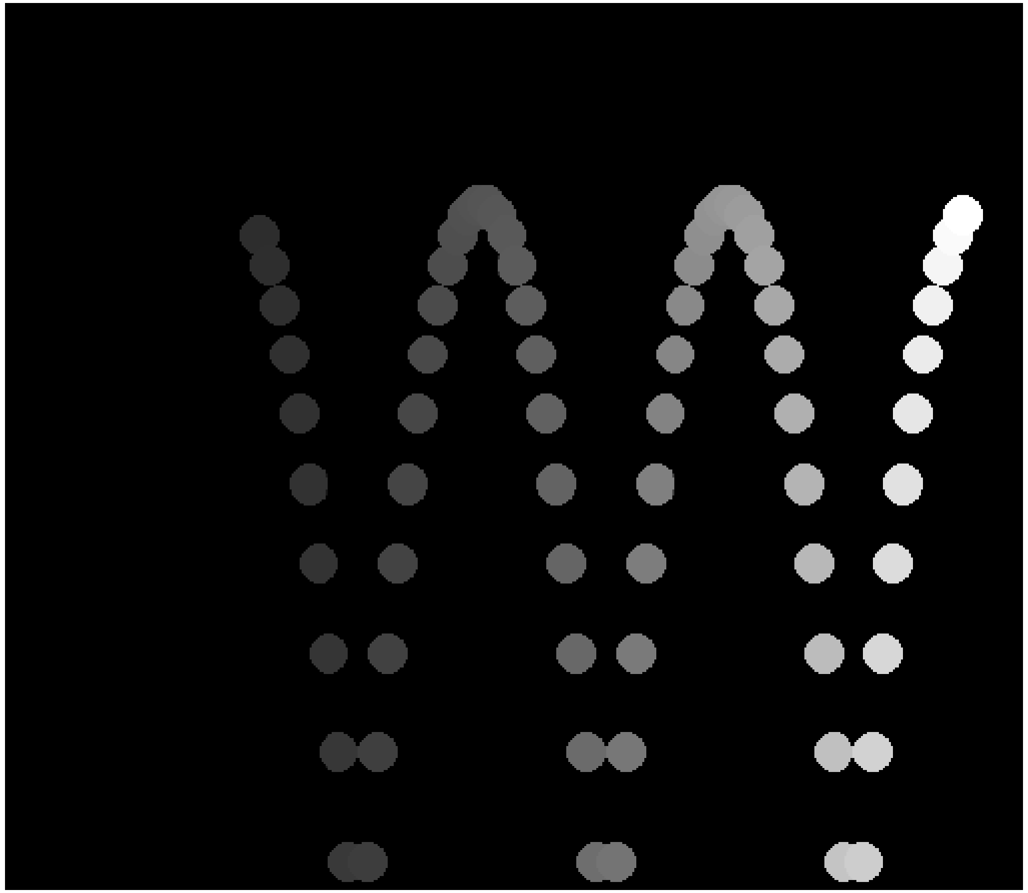
```
int meaningOfLife = 42;
boolean lifeHasMeaning = (meaningOfLife == 42);
if (lifeHasMeaning) {
  println("Phew.");
}
else {
  println("Oh no...");
}
```

Push the button...

```
boolean lightIsOn = false;
void setup() {
size(500,500);
}
void draw() {
if (lightIsOn) {
background(255);
}
else {
background(0);
}
}
void mouseReleased() {
lightIsOn = !lightIsOn;
}
```

Stretch Break!

Try this shape must bounce but must not go beyond the screen coordinates.



Thank you