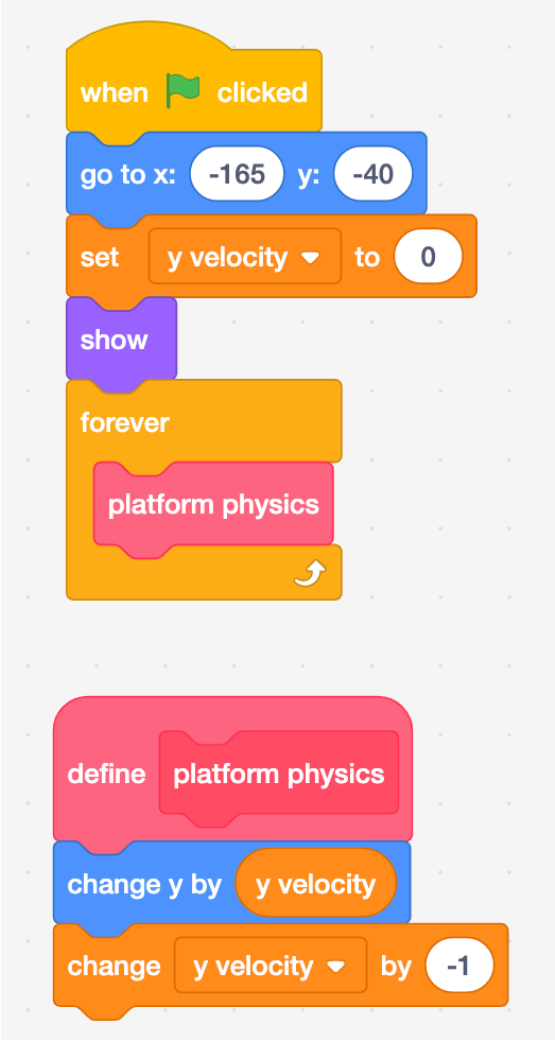
Platform Physics game by PullJosh on Scratch

Step 1

Start by creating a **y velocity** variable in the player sprite. This keeps track of how fast the Player sprite is going the the **y** direction.



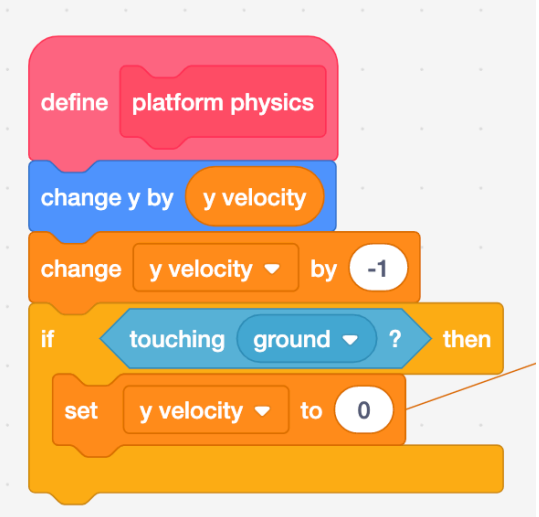
Then create a **platform physics** block which is inside the game loop doing all the stuff with velocities. Make sure you check “run without screen refresh” or it will run too slowly.

Add the blocks on the right to the script of your Player sprite. Because we increase y velocity each time we run the **platform physics** block it looks like our sprite is speeding up as it falls, which makes it look more realistic

When you run this code you’ll see that the sprite falls but there’s a problem: it falls through the ground.

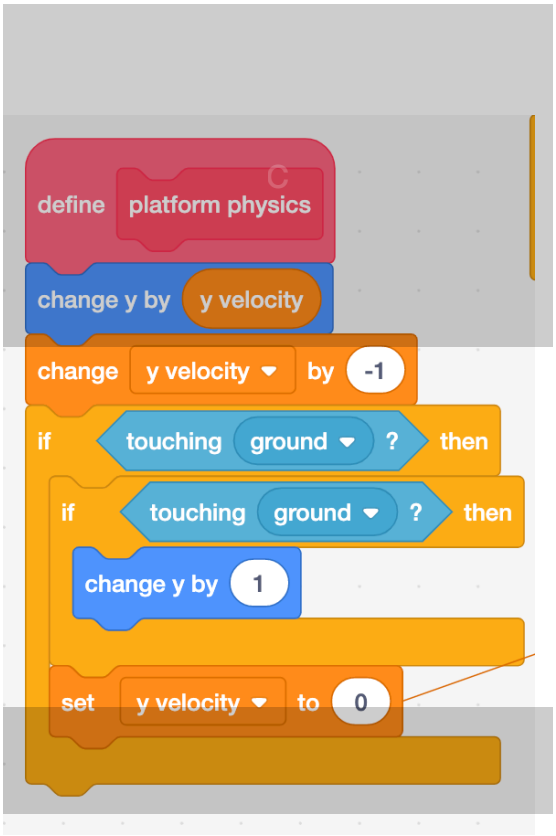
Step 2

So that the player doesn’t fall through the floor, change the **define platform physics** script to this:



This is a bit better - it doesn’t fall so far through the floor, but it’s still not great

We really need to move the sprite back up a bit if it's fallen through the ground, so add an extra if block like this:



This brings the player back up onto the ground if it's sunk through it, but it does it quite slowly which looks weird.

This is because each time we move it up by 1 it's going through all the other steps in the **platform physics** script as well. We really just want to fix it all at once inside **platform physics**.

Step 3

To do this we need to add a **repeat** block round our second **if touching ground block**, but how many times do we need to repeat?

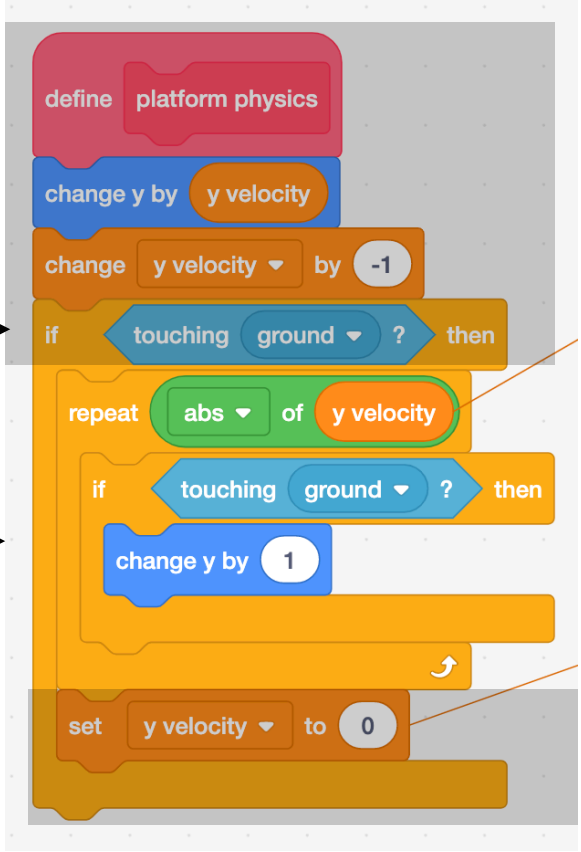
We know that the last amount we changed **y** by before it reached the ground (or ended up in the ground) was **y velocity**. So the most we need to move it to get it out of the ground is **y velocity** in the other direction (though it might not be as much as that).

Remember that **y velocity** is a negative value because our sprite is moving down the way. We can't repeat something a negative number of times, so we repeat it **abs of y velocity** times, where **abs of** means the "absolute value" of **y velocity**.

The absolute value of a number is just the number itself, without any negative signs, so the absolute value of 3 is 3 and the absolute value of -3 is also 3.

If our sprite is touching the ground it may be just touching it or partly inside the ground.

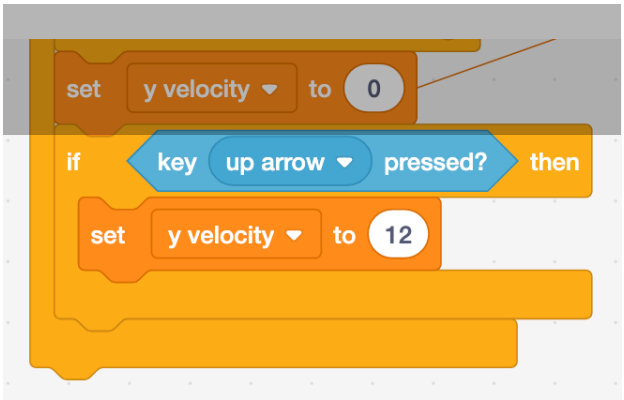
Keep checking to see if the sprite is touching the ground and if it is, move it up one pixel. Repeat this for the absolute value of y velocity times.



Step 4

To allow the sprite to jump when the player presses the up arrow key add these blocks just below set y velocity to 0 but still inside the if touching ground block.

This means you can only jump if you're starting from the ground (no double-jumping



Step 5

Next, we want to put in the ability to move the player left or right, using the arrow keys. To help us do this we need to make another variable that will store the player's speed moving across the screen, that is, in the **x** direction.

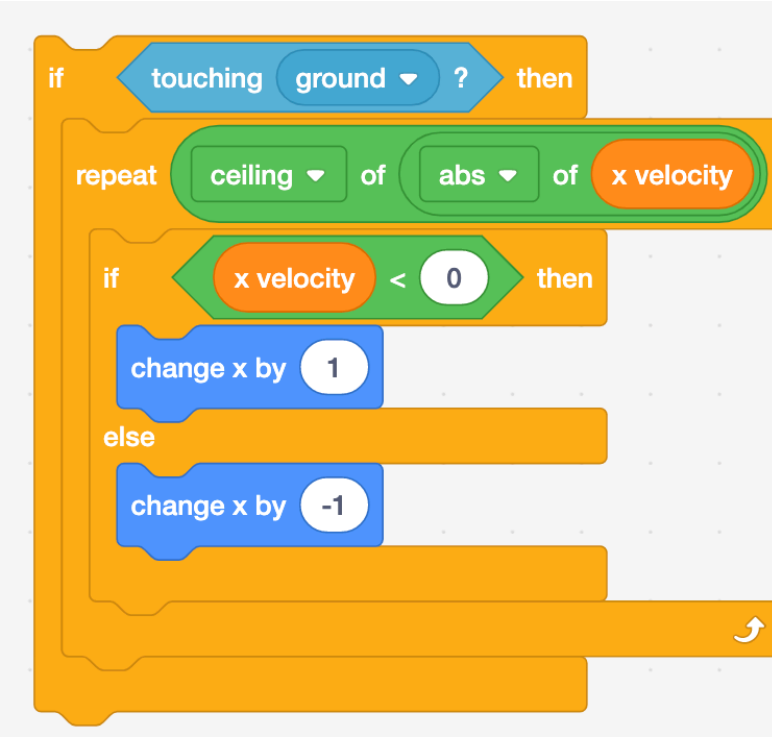
Call this new variable **x velocity.**

Now add these blocks, joining them onto the bottom of the ones already in the platform physics script:

Step 6

The sprite is now going left and right when arrow keys are pressed, but it's also drifting through walls.

To stop this and also make sure it doesn't get stuck partly inside a wall (as it did earlier in the ground) we need to add this to the code just under the **change x by x velocity** block:



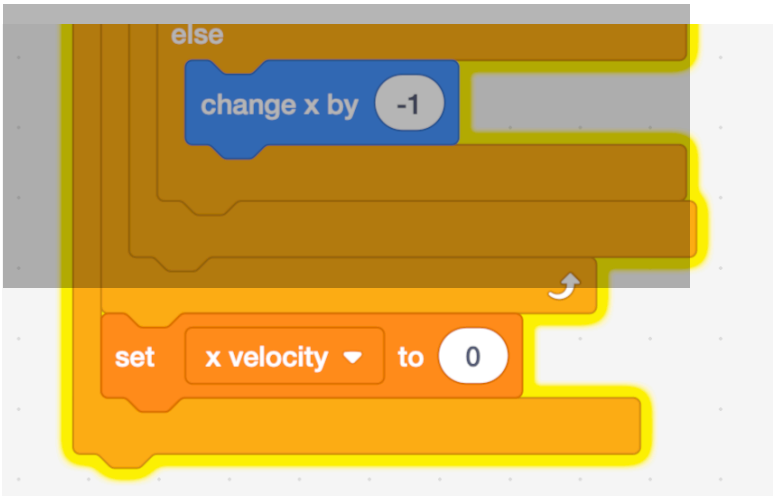
When we were extracting the block from the ground after falling, we knew we wanted to change **y** by 1, to make it move up a bit.

This time, if the sprite is moving left - i.e. **x velocity** was negative - we want to change **x** by 1. But if the sprite is moving right - ie. **x velocity** is positive - we want to change **x** by

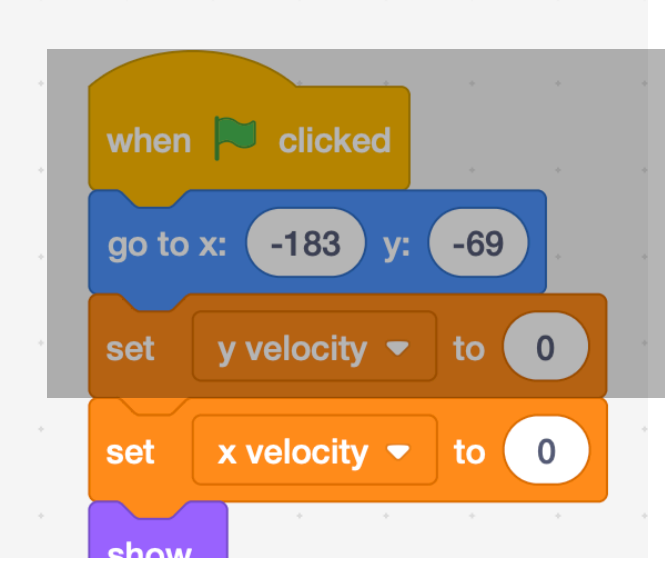
-1.

Step 7

Finally, if the sprite has hit a wall it should stop trying to move in that direction. So add a block to set **x velocity** to 0 right after the repeat block:



We should also set **x velocity** to 0 at the start of the game by adding this block here:



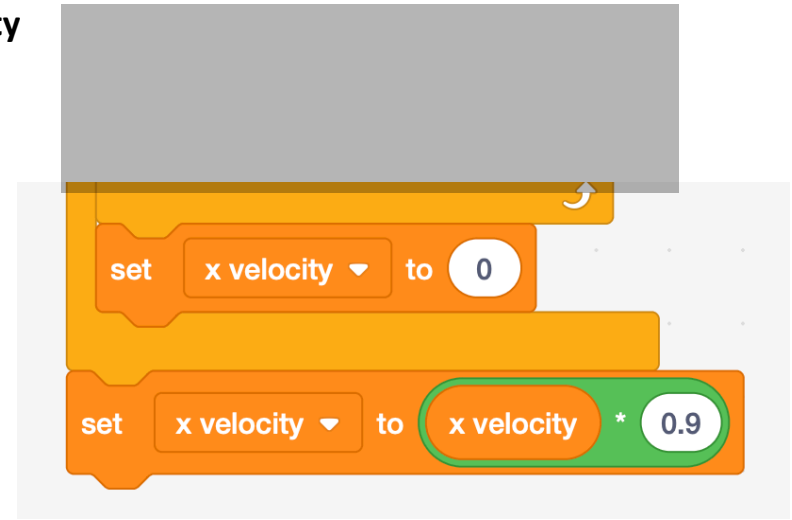
Step 8 - add friction

If we move our sprite right, up the stairs and onto the slope you'll see it just keeps going until it hits the right-hand wall. It might be that in your game you want the slope to be really slippy like that but, if you want it to be more like an ordinary bit of ground, you'll need to add **friction** to the physics.

We saw earlier that the y velocity of the sprite keeps increasing as it falls, as we add -1 to it each time the platform physics block is run. Adding friction will slow our

sprite down gradually by making **x velocity** a bit smaller each time the platform physics block runs.

To do this, at the very bottom of the **platform physics** definition, we set **x velocity** to (**x velocity** multiplied by a number between 0 and 1).

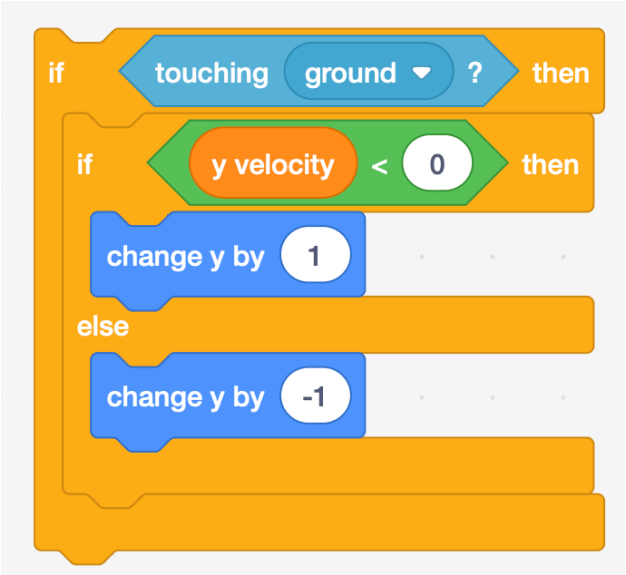


Step 9 - stop jumping through ceiling

We've almost fixed all our issues with walls/floors that the sprite can move through, but it's still able to jump up through a platform above it. To fix this we just need to adjust the code we have to get the sprite out of the ground slightly and make it more like the code to get the sprite out of walls.

If we go back up to the top of the platform physics definition, we want to replace the if block that says

With this version that checks whether the sprite is moving up (**y velocity > 0**) or down (**y velocity < 0**) and changes **y** accordingly.

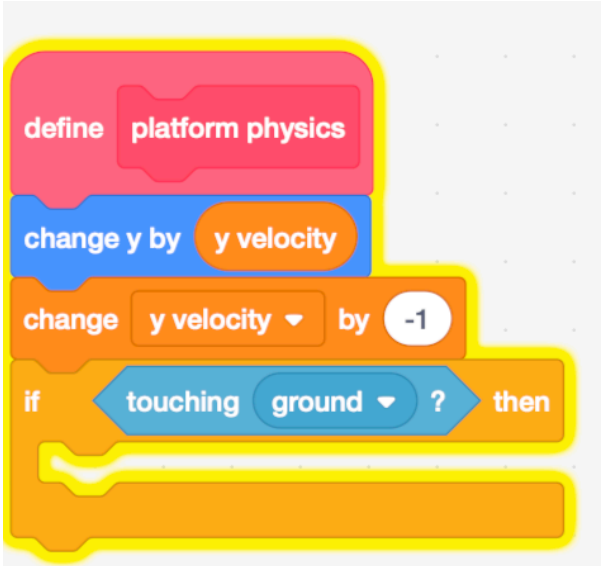


Try this out - there may still be a problem!

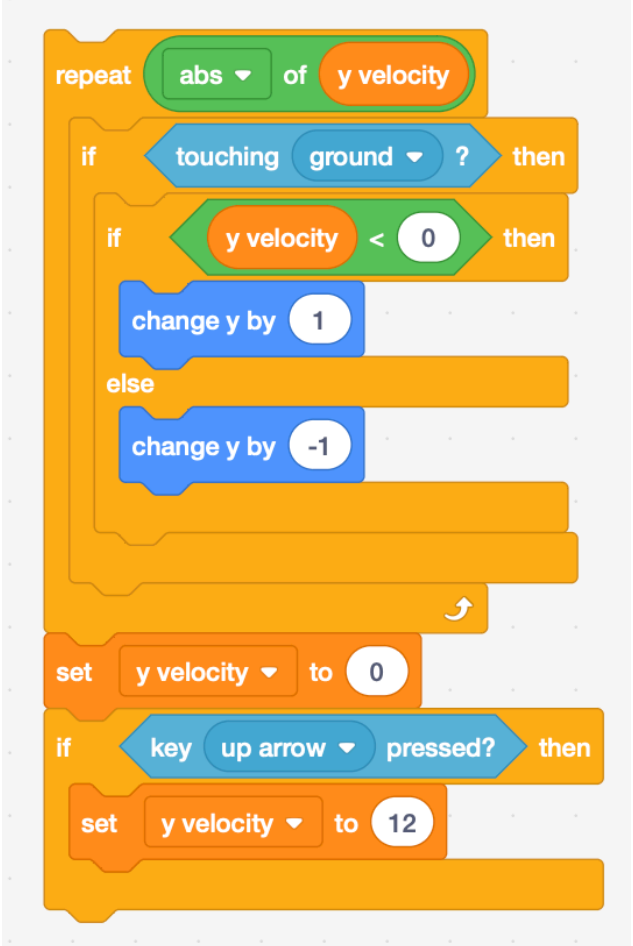
Step 10 - stop jumping through ceiling - the fix

So if you tried that, you'll see that it didn't seem to work! But don't worry - you don't need to throw your new section of code away. We just need to slightly rearrange our code and change one of our **if** blocks for an **if then else** block.

First of all, detach the section of code that starts with the if key right arrow pressed block. Now take the top section and split it into two sections. First these ones:



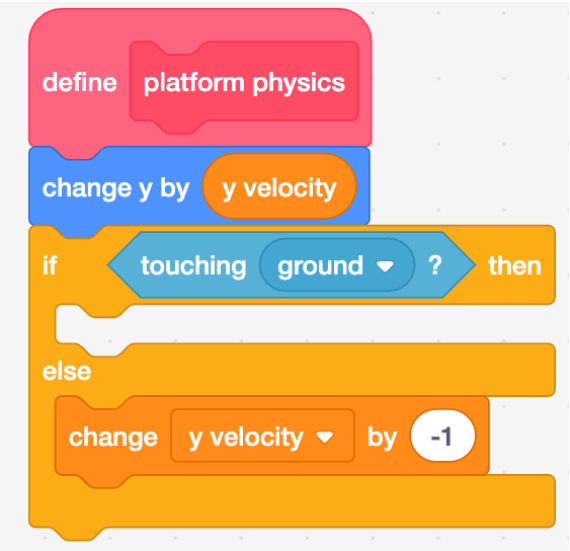
Then everything that was inside that **if touching ground** block:



At the moment, every time the **platform physics** block runs we start by changing **y velocity** by -1. If we add a **wait 0.5 seconds** block after the **change y by -1** block and set **y velocity** to show on the screen we can see what's happening. When our sprite hits the ceiling, the y velocity is -1. This means that, when we go into the "get us out of the floor or ceiling" loop Scratch changes **y** by 1, moving us up rather than down.

To fix this, we need to make sure we only change **y velocity** by -1 if we're not touching the ground at that point. This makes sense as a falling object keeps getting faster as it fall8s - its **y velocity** becomes a larger negative number - but it doesn't keep getting faster once it's hit the ground. That would be weird.

So, in the first section of code, change the **if touching ground** block to an **if touching ground then-else** block, and move the change y velocity by -1 to be inside the else branch of the block



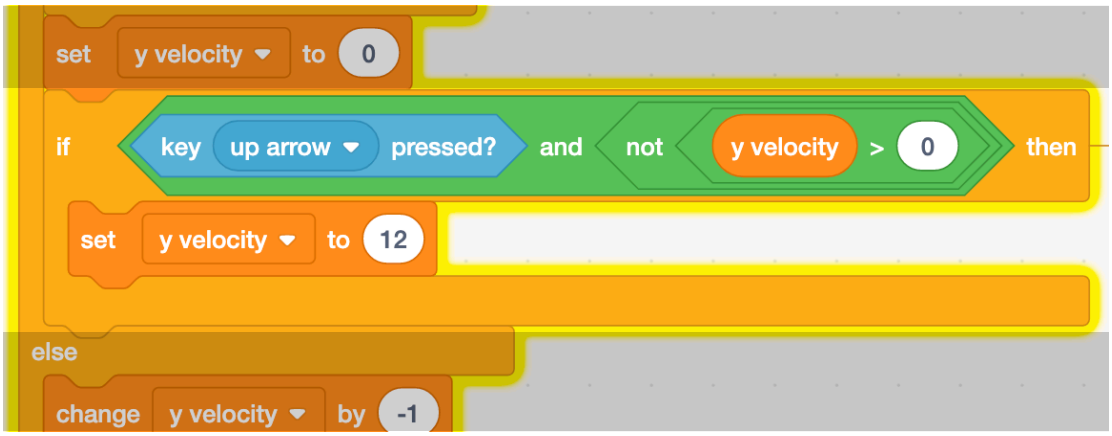
Now put the second block of code in **Step 10** (i.e. everything that was inside the original **if touching ground** block into the **if** branch of this block. Finally, reattach the section of code that deals with the sprite moving left and right to the bottom of this **if then else** block.

Step 10 - Stop getting stuck on the ceiling

The player can do this because, at the moment if they hit **ground** after we update the sprite's **y** value we assume they fell onto it and so they're allowed to jump. If the bit of **ground** they hit is actually a ceiling and they jumped onto it still let them jump. They can't now go through it like they did before but they can still keep jumping to let them stick there.

To fix this we need to add some code that works out we're touching the current bit of **ground** because we fell or because we jumped. We do this by adding an extra

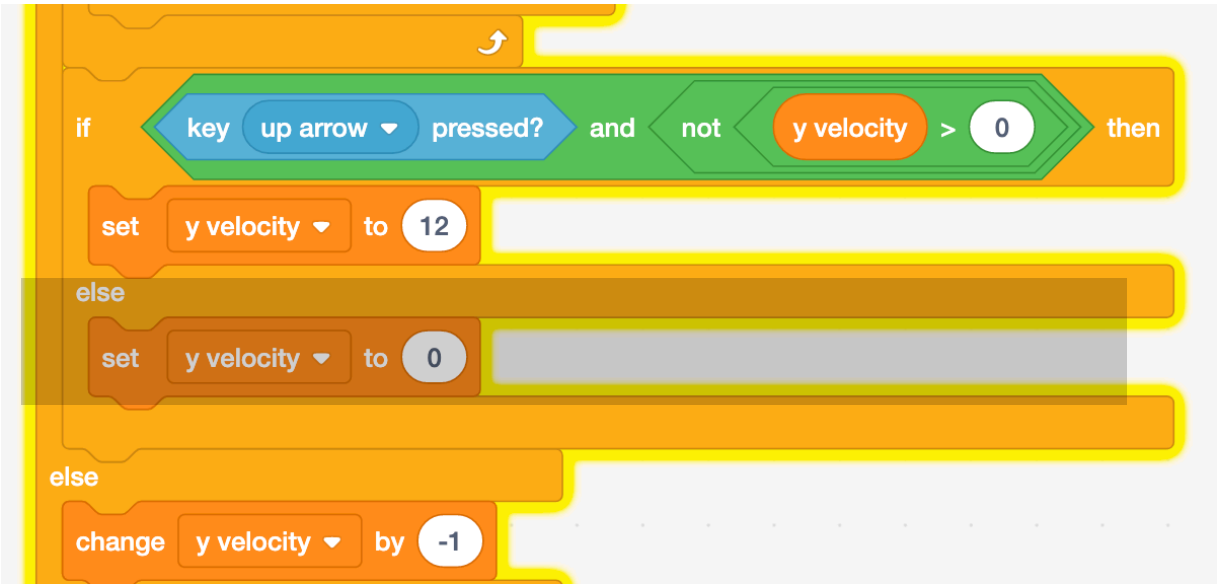
check into the condition on the if block that checks to see if the **up arrow key** was pressed. We want to make sure the player can only jump if they've just fallen onto the



ground - so if their **y velocity** is negative, or less than zero.

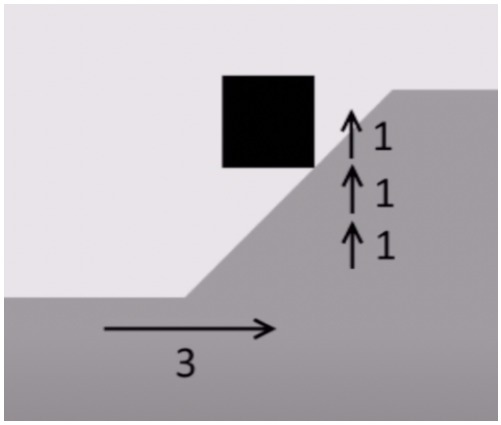
But if we try this out you'll see it hasn't fixed the problem - if we always set **y velocity** to 0 before doing this check it will always be true and we'll always be able to jump. If we try to fix it by setting y to 0 immediately after we set it to 12 it's more of less the same as not setting it to 12 and we can't jump at all.

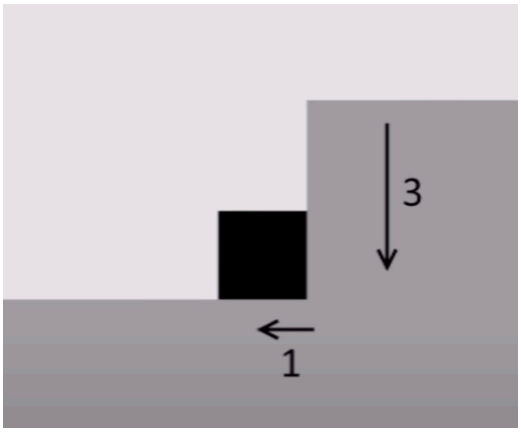
The answer is to change our **if** block for an **if then else** block where if the player is on the bottom of the ground sprite we set **y velocity** to 12. If their **y velocity** is greater than 0 - i.e. they're already jumping - we set it to 0.



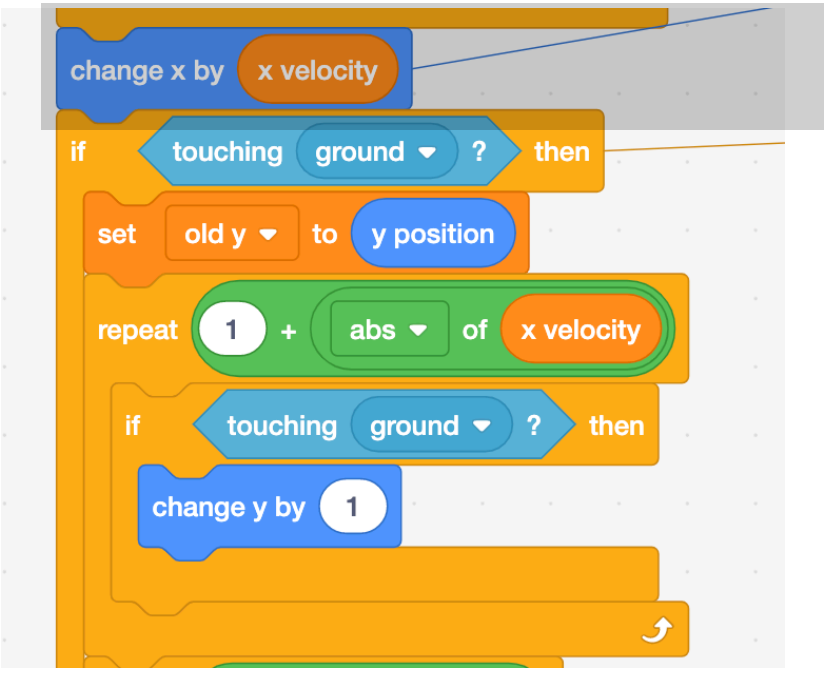
Step 11 - Stop getting stuck going up slopes

To stop the player sprite getting stuck moving up a slope we're going to see if the player sprite is touching ground after updating its x position. If it is, we're going to assume it's on a slope and for every amount along we just moved it, we're going to move it the same distance up. Since we just moved it by x velocity which is 3, we'll need to move it up 3 as well.





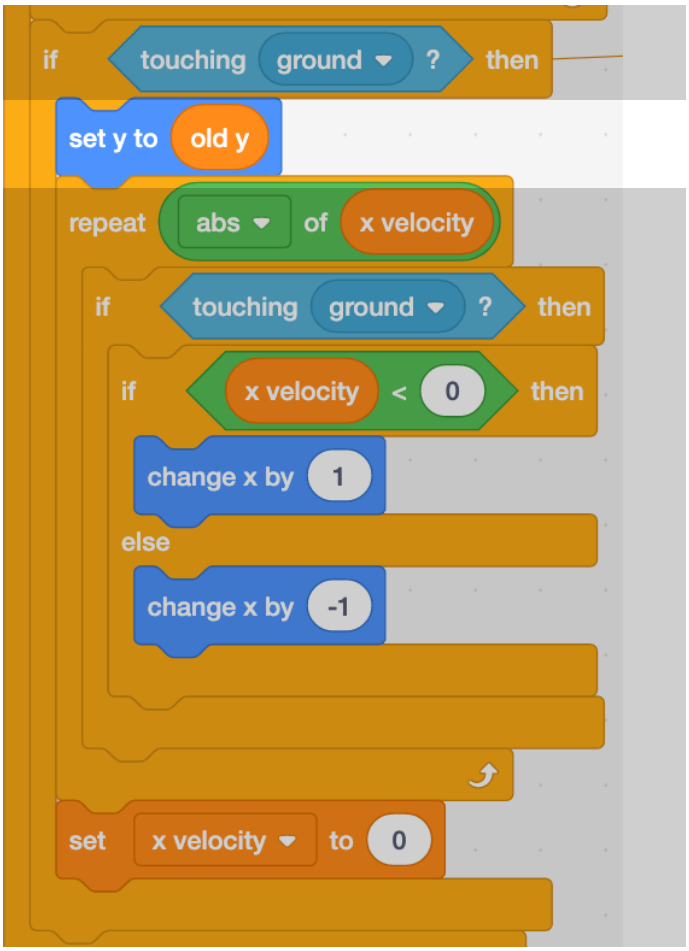
To do this, add these blocks to the top of the **if touching ground** block just under the **change x by x velocity** block.



The reason we've created a new variable called **old y** and set it to **y position** before we start doing this "get us up the slope" code is because we might **not** be on a slope. F

If that's the case, we'll need to set our y position back to where it was before we tried

our slope-escape.



The blocks you just added are followed by the "greyed out" blocks on the right here, which is the code we added in Step 6 to get the Player out of walls.

If the **if touching ground** block at the top here returns **True** after we've tried to get out of the slope we can assume the **Player** is not in a wall.

If that's the case, it must be touching a wall and we should set our player's **y position** back to where it was before we started getting it out of a slope that wasn't actually there.

So add this **set y to old y** block.

Step 12 - Ability to wall jump - ie. bounce off walls

At the moment, after we've checked for going up a slope, if we decide the player has hit a wall, we set the **x velocity** to 0 so that they stop moving towards that wall (see the code at the end of **Step 11.** However, to allow us to "wall jump", I.e. stick to a wall and bounce up and off it, we want to check after doing that to see if the up arrow key is being pressed.

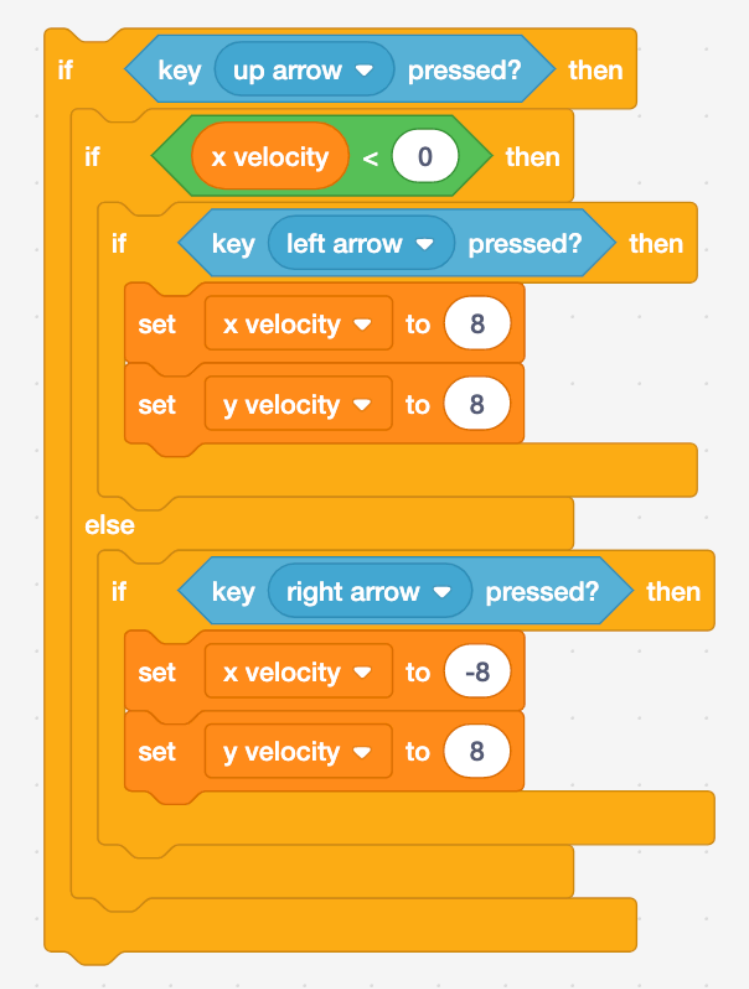
We also want to see which way they're going -i.e. is **x velocity < 0** (moving left) or is **x velocity > 0** (moving right)? We want people to wall-jump only if they're trying to jump while **pushing into** the wall they're next to.

Put together these blocks but don't connect them to the main platform physics code just yet.

Here we check to see if the up arrow key is pressed. If it is, we check to see what direction the player is moving in.

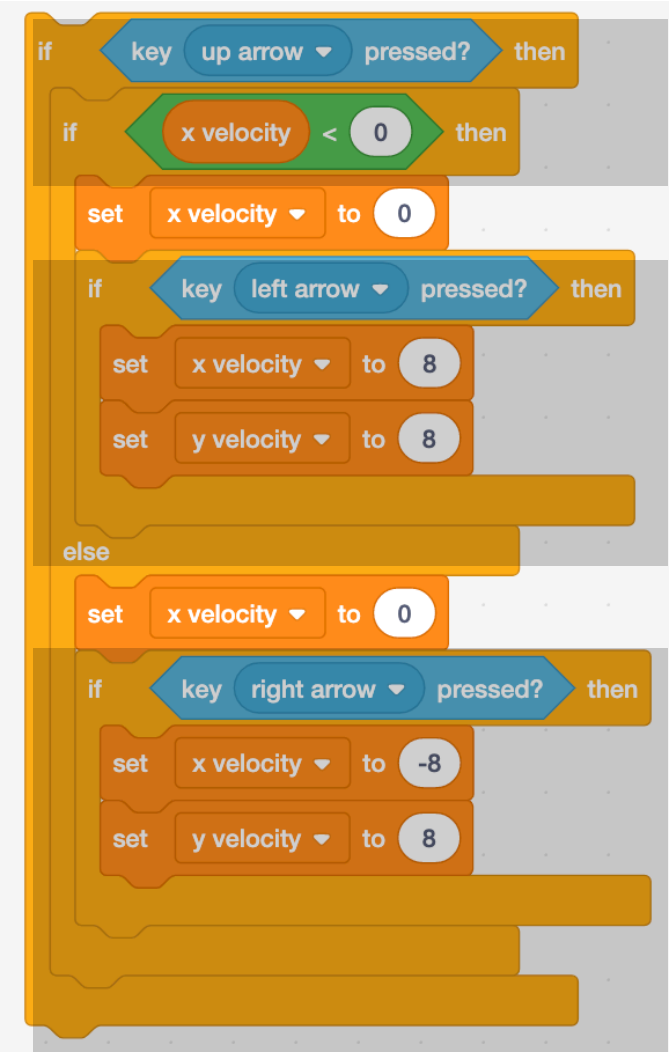
If they're **moving left** and are pressing the **left arrow key** to push into the wall, we set the x and y velocity so that they move up and to the right, jumping off the wall.

If they're **moving right** and pressing right arrow, we move them up and to the left.



You might have spotted that we have a slight problem: if we put this block of code after the block **set x velocity to 0** we won't jump at all because we're not moving left or right. If we put it before **set x velocity to 0** we'll jump but only straight up because that immediately undoes setting **x velocity** to 8 or -8.

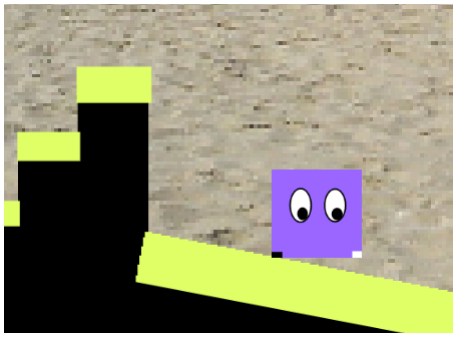
The answer is to modify our section of code so that we set **x velocity to 0** after we've checked for the player going left or going right, but before we see if the player wants to wall jump, like this:



One that's done, replace the set x velocity to 0 block near the bottom of the main platform physics code with this new larger block and you should be able to wall-jump.

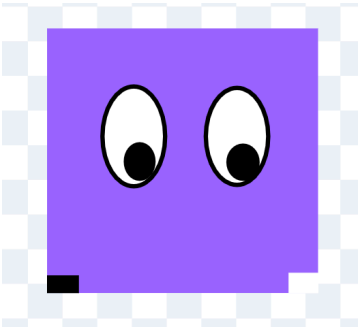
Step 13 - Make the player sprite detect slopes and adjust its angle

At the moment the **Player** sprite goes up and down the slope with only one corner touching it - the sprite is horizontal, but the ground under it isn't.



First, we need to add a layer of a different colour to all the surfaces of the **ground** sprite that the **player** sprite could land on due to gravity pulling it down. In this example we've gone for neon yellow, but you

should choose whatever colour you like - it can even be fairly similar to the **ground** sprite's main colour in order to be less noticeable.

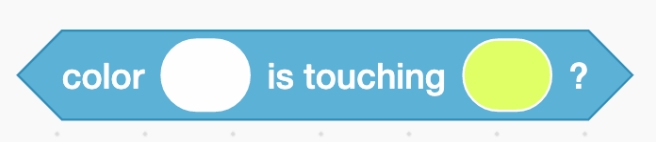


To get the sprite to spot when it doesn't have its whole base on the ground we need to add a couple of coloured "sensors" to the bottom of the sprite.

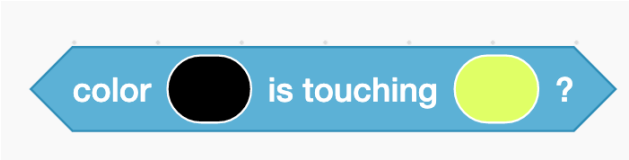
You can see here that we've added a **black** rectangle to the bottom left corner of the sprite, and and **white** rectangle to the bottom right corner. These have to be completely covering the main colour of the sprite - with no edges of purple round them.

We can use these as "sensors" by using Scratch's **color is touching** block.

If we want to know if the **bottom right** corner of the sprite is touching the ground we can use this block:



If we want to know if the **bottom left** corner of the sprite is touching the ground we can use this block:



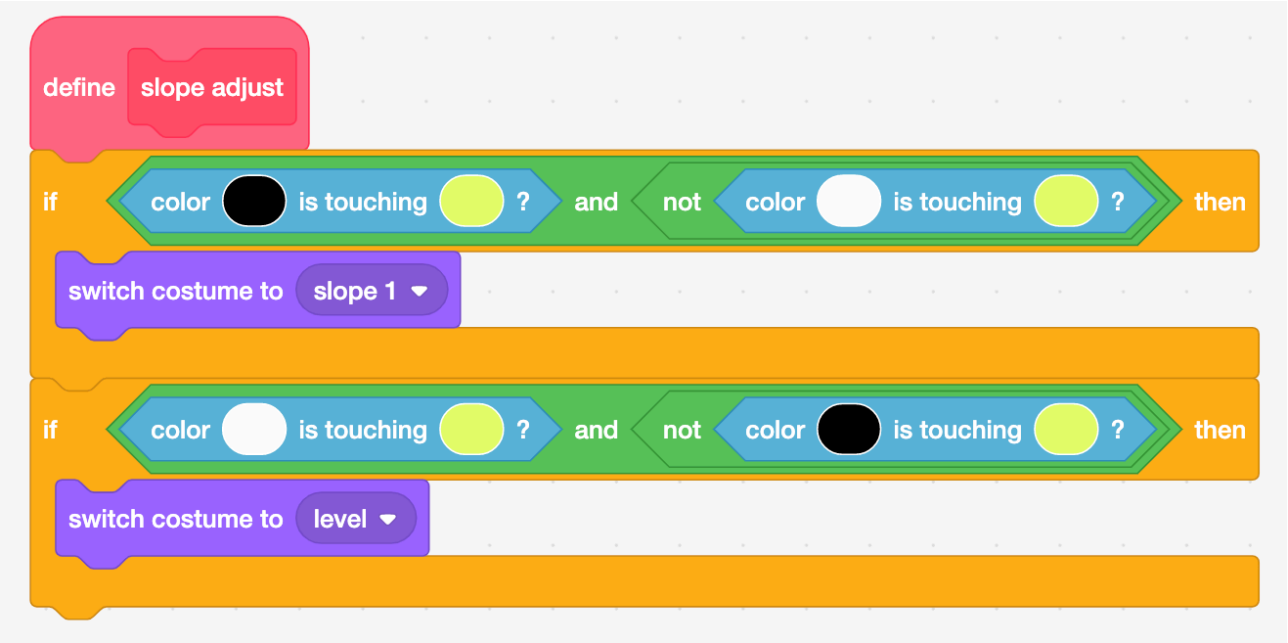
We need to do this because we want to know if one of the corners of the sprite isn't on the surface, but Scratch only keeps track of the position of the **centre** of each sprite.

Ideally, if we notice one corner isn't touching the ground we could keep lowering that corner until it meets the ground by turning 1 degree at a time. However, in our platform world with gravity this tends to go a bit haywire because the sprite is actually constantly being pulled into the ground, then pulled out.

To get round this we'll use the less elegant, but easier method of creating an extra costume where the sprite is angled to lie flat against the surface of our slope.

This isn't a general solution because if we make more levels with different angles or directions of slope we'd need to create different costumes for them and possibly add a bit more code.

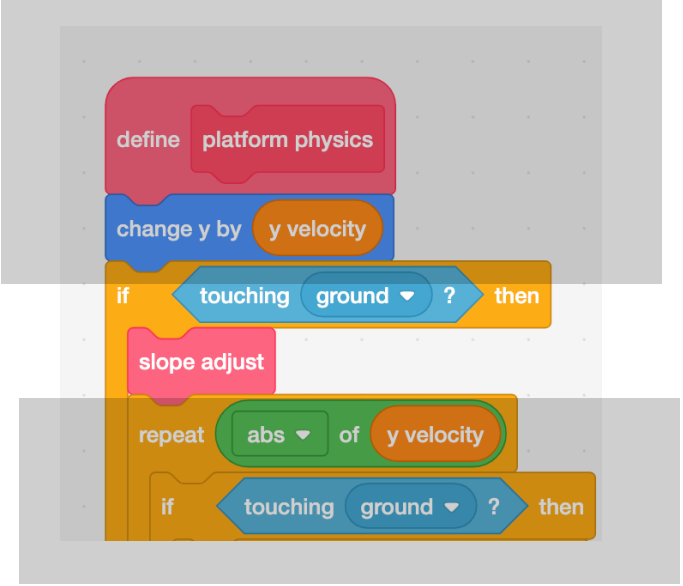
To keep our code a bit tidier, lets make a new block called slope adjust where we'll check to see what costume we need to use:



The first **if then** block checks to see if the bottom left corner (**black** sensor) is touching but the bottom right (**white** sensor) isn't. If that's true, we're in the situation in the first picture in this section and we need to switch to the sloping costume.

If we're in the sloping costume and go back to a flat area, the **player** sprite will have its **white** sensor touching ground but not the **black** one. This is the situation the second if then block checks for, switching costume to the one for level surfaces if that's true.

Now add the slope adjust block to the code of platform physics here, just inside the first **if touching ground** block.



Step 14 - Clean up - make the code easier to use and read