# Making choices with "if"

Lesson 9

## "Control Structures"...

Up to this point, we haven't let Wink make any of his own choices. All the programs we have written so far simply run a set of functions in order. This is fun, but a robot becomes much more useful if it can change its behavior based on different things that may be happening.

There are a few different types of code that can cause a computer program to change how it is running based on different things. These types of code are called "control structures" because they control how your program runs, or more specifically, they control what lines of code run and what lines of code don't run.

By using control structures, you can do things like, making a light turn on if a button is pressed, and making the light turn off if the button is not being pressed.

One of the most common and powerful control structures in any programming language is the "if" control structure, and we are about to learn all about it in this lesson. Because I'm going to need to write "control structure" a lot during this lesson, I'm going to write "CS" instead going forward to save space.

The "if" CS basically considers if something is true or not, and if that thing is true, some code is executed, and if that thing is not true, then the code is not executed.

Remember in the previous lesson where we counted how many times Wink ran the loop() function? We could use "if" to consider whether or not the loopCount is greater than 5, and if the loopCount is indeed greater than 5, Wink can execute a line of code that turns on his eyes. We'll do exactly this for our first real example, but first let's talk about how you actually write an "if" CS.

The "if" CS looks at something called a "condition" and determines if that condition is true or not. If the condition is true, then whatever code is inside the "if" curly braces is executed, and if the condition is not true, then whatever code is inside the "if" curly braces is not executed.

This is easy to understand if we look at a few examples.

```
if (2 > 1){
  // if true, code inside here is executed
}
```

} This is an example 'if' control structure.

It looks complicated but it's not really. Let's discuss the parts of the "if" CS...

It begins with the word "if" followed by a "condition". The "condition" is the part inside the pair of parenthesis. In this case, the condition is 'two is greater than one'. Many different kinds of conditions can be used, this is just an example to help you understand how things work.

Following the "condition" is an open curly brace { The open curly marks the beginning of the code that will be run if the condition is true. The end of the "if" CS is a closing curly brace } The closing curly marks the end of the code that will be run if the condition is true.

2 SKILL LEVEL

"If" always evaluates whether or not the "condition" is true. In this case, it asks the question "Is it true that two is greater than one?". Because two is indeed greater than one, this condition is said to be "true", and because it is true, then whatever code is inside the curly braces will be run.

Note that there is no semicolon ; used for the "if" CS itself, though any code inside the curly braces does need semicolons after each line as usual.

Now let's try our first real example. This is based on the example from a previous lesson where Wink counted how many times he ran the loop() function.

```
# include WinkHardware.h

void setup(){
  hardwareBegin();
  playStartChirp();
}

int loopCount = 0;   //declare loopCount

void loop(){

  if (loopCount > 5){  //if loopCount is greater than 5...
    eyesCyan(100);     //then run code inside the curly braces
  }

  loopCount = loopCount + 1;  //add 1 to loopCount
  delay(1000);                //delay 1 second
} //end of loop()
```

} Normal setup

} Declare loopCount variable

} The statement eyesCyan(100); is executed if it is true that loopCount is greater than 5.

*Wink_Ch09If_Ex01*

Now let's talk about what happens.

When loopCount is declared, the value is set to 0. The first time the loop() function is run, loopCount is 0. When the "if" CS is run, the "condition" inside the parenthesis is considered. The program asks "is it true that loopCount is greater than five?". In this case, it is not true, because loop count is only zero. Because the condition is not true, the code inside the curly braces is not run. Everything inside the curly braces is skipped and the next line of code below the closing } curly brace is run.

loopCount is then increased by one, so it now holds the value of 1. This continues until eventually, loopCount becomes six. At that point, the sixth time the loop() runs, the "if" CS runs and asks "is it true that loopCount is greater than five?", and because loopCount is six at this point, the condition becomes "true" because it is indeed true that six is greater than five. Because it is true, the code inside the curly braces is run, in this case it is eyesCyan(100); which turns on the eyes.

2 SKILL LEVEL

## More about the "condition"...

Remember that the "condition" is the part that is evaluated and tested whether it is true or not. It is the part inside the parenthesis. In the above example, we were basically trying to get the eyes to turn on after five seconds, but you may have noticed it actually took six seconds for the eyes to turn on. This is because we told the "if" CS to be true only if loopCount was **greater** than five. So this isn't going to be true until loopCount becomes six.

Sometimes we may want the "if" CS to be true if a number is greater than **or equal to** a number. We can make a simple change to the "greater than" sign and change it to "greater than or equal to". Like this...

```
int loopCount = 0;   //declare loopCount

void loop(){

  if (loopCount >= 5){  //if loopCount is greater than 5...
    eyesCyan(100);      //then run code inside the curly braces
  }

  loopCount = loopCount + 1;  //add 1 to loopCount
  delay(1000);                //delay 1 second
} //end of loop()
```

*Wink_Ch09If_Ex02*

} The "condition" is now "greater than or equal to". This will cause the "if" condition to be true when loopCount is five, or anything greater than five.

Notice the greater than > symbol is now followed by an equals sign, so the condition is now true if loopCount is greater than **or equal to** five.

The symbol used in the condition (like the "greater than" symbol) is called a "comparison operator". This basically means that the symbol is used to "compare" the items included in the condition. There are six different comparison operators that can be used within the condition of your "if" CS.

Here's a quick chart...

| Comparison Operator | What it means |
|---|---|
| == | equal to |
| != | not equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |

2 SKILL LEVEL

Most of the comparison operators in the chart should make sense, but you may have noticed that the "equal to" comparison operator is actually two equals signs together. What's up with that? You would think "equal to" would just be a single equals sign, right?

I'll explain why it is different in this case.

You may remember from the Variables lesson that when assigning a value to a variable, you also used an equals sign. More specifically, you used a single equals sign. Like this...

```
birds = 10;  //single equals sign is an "assignment"

if (birds == 10){ //double equals is a "comparison"
  eyesRed(100);
}
```

} The single equals sign "assigns" the value of 10 to the variable birds

} The double equals "compares" one item in a condition to another item in a condition.

In the first line of code above, we are using the single equals sign to "assign" a value to a variable (10 in this case). The single equals sign is an "assignment" operator.

Then, in the "if" CS that follows, we use the double equals sign to "compare" the two items in the condition. In this case, we are going to compare birds to the value 10.

If it is true that birds is equal to 10, then the condition will be "true" and the code inside the curly braces will be run. If the value is not exactly equal to 10, then this condition will be false and the code inside the curly braces will not be run.

## A different way to format your "if"...

You can format your "if" a few different ways. Some programmers like to put the open and closed curly braces on their own lines. Some people feel that this is easier to read. Up to this point, we have put the open curly { on the same line with the condition, but some people like to put it on it's own line following the condition. Either way is fine. Here's an example...

```
if (birds == 10)    //the condition is on it's own line
{                   //the open curly is on it's own line
  eyesRed(100);     //the statements inside the curlys here
}                   //finish with closed curly

        //the above code works the same as the below code
        //either method is okay and commonly used.

if (birds == 10){   //open curly on same line as condition
  eyesRed(100);     //statements inside the curlys
}                   //finish with closed curly
```

} Example where open curly { is placed on it's own line. Some people feel this method is more easy to read.

## Making a second choice with "else"...

So now you know how to make code run if a certain condition is true. Sometimes this is all that is required to make your program work, but sometimes, you may want to do one thing if a condition is true, but do something else if the condition is not true.

For example, you may want a light to turn on if a button is pressed, and if the button is not being pressed, to turn off the light. We'll demonstrate how to use the button in your "if" CS later. For now let's see how we use "else" with our earlier example using loopCount.

Think of "else" as telling the program to do something "else" if the "if" condition is not true.

Let's see how a real example looks...

```
# include WinkHardware.h

void setup(){
  hardwareBegin();
  playStartChirp();
}

int loopCount = 0;   //declare loopCount

void loop(){

  if (loopCount == 3)  //if loopCount is equal to 3..
  {                     //open curly for 'if'
    eyesCyan(100);     //then run code inside the curly braces
  }                    //closing curly for 'if'
  else                 //if not true, else do this other thing
  {                    //open curly for else
    eyesOff();         //turn the eyes off
  }                    //closing curly for else

  loopCount = loopCount + 1;  //add 1 to loopCount
  delay(1000);                //delay 1 second
} //end of loop()
```

} Normal setup

} Declare loopCount variable

} The statement eyesCyan(100); is executed if it is true that loopCount exactly equal to 3.

} Any time loopCount does not equal 3, run "eyesOff();" instead.

*Wink_Ch09If_Ex03*

When you run this code, Wink's eyes should remain off for three seconds. Then his eyes should turn on for one second, then turn back off. This happens because the first three times the "if" CS is run, loopCount does not equal 3, so the code inside "else" is run, eyesOff() in this case which turns Wink's eyes off. After the loop has been run three times, loopCount will equal 3. At that time, when the "if" CS is run, the condition will be true, so eyesCyan(100); will be run which turns on the eyes. We then add 1 to loopCount and it the loop repeats. This next time, loopCount is equal to 4, so the condition is no longer true, so the code inside "else" runs again, turning the eyes off again.

2 SKILL LEVEL

Let's take a minute to study how the "else" is actually written. The "else" always follows directly under an "if". The "else" does not have a condition (it has no parenthesis with a comparison inside) because it is run automatically if the condition of the "if" is not true. The "else" does have its own set of curly braces. These curly braces mark the beginning and end of the code executed by the "else".

After the "if" and "else" are finished, the next line of code below the "if" and "else" is run normally as you would expect.

I know that was a lot of information coming at you really fast. If you're a bit confused, go back and slowly read and think about each step. You may need to go over it a few times to understand it. This was a challenging concept to learn back when I began programming so it's okay if you're a bit confused.

**Helpful tip...** You're eventually going to have quite a few sets of those of curly braces in your code. I'll point out a neat feature in the Arduino IDE. When you place your cursor next to a curly brace, the Arduino IDE will automatically draw a little circle around the matching curly brace for that section of code. It does the same thing for parenthesis. This helps you visually pair them up as your code eventually becomes more complex. Also pay attention to the note about formatting later in this lesson - by spacing your curly braces correctly you can easily keep track of them.

## Really simple light seeking...

Now we can make Wink do something smart with what we've learned. We know how to use "if" and "else". We know how to make Wink's motors go and how to read his light sensors. Now we know everything we need to create a really simple light seeking program.

Before we write the program, let's consider how we can make this work.

The idea is to have Wink spin around in his own footprint while looking toward the brightest light. In order to do this, we can read Wink's left and right ambient light sensors, then determine which sensor is seeing the most light. We can then cause Wink to make a choice and use his motors spin to the left, or use his motors to spin toward the right.

Because we are going to read both left and right light sensors, we will probably need a couple variables to hold the values of the light sensors.

Inside our "if" CS, we will probably want to compare these two sensor values. If one sensor value is larger (like the right sensor), then we will want to run code that makes Wink turn toward the right. We can use "else" to make him do the opposite thing.

Consider how you would put this together, then go to the next page and see if my example is similar to the one you would write. If you want to challenge yourself (and I hope you will), then open the Wink Base Sketch and write this program completely on your own, then compare with my example on the next page.

## Simple Light Seeking - an example...

I suggest you to try to figure out how to solve the simple light seeking problem on the previous page on your own before moving ahead. If you are doing these lessons as a group, at least have a discussion with your friends and think of how you can make this work.

Here is the example I came up with...

```
# include WinkHardware.h

void setup(){
  hardwareBegin();
  playStartChirp();
}

int leftLight, rightLight;    //declare the variables

void loop(){

  leftLight = analogRead(AmbientSenseLeft);   //read left
  rightLight = analogRead(AmbientSenseRight); //read right

  if (rightLight > leftLight) //if right is greater than left
  {                           //open curly for ‘if’
    motors(100,-100);         //spin to the right
  }                           //closing curly for ‘if’
  else                        //if not true, else do this...
  {                           //open curly for else
    motors(-100,100);         //spin to the left
  }                           //closing curly for else

} //end of loop()
```

*Wink_Ch09If_Ex04*

} Normal setup

} Declare the variables we will need

} Read both sensors.

} If rightLight is greater than leftLight, then run code to make motors spin and rotate Wink to the right.

} If the condition was not true (that rightLight is greater than leftLight), then they must have been either equal, or leftLight must have been larger. So we will run code to make motors spin and rotate Wink to the left.

} End of loop. Program goes back to beginning of loop() and does the whole thing again.

Isn’t it amazing that you can make the robot do something that seems so smart with such a small amount of code?

Try running this code and see how Wink behaves. It may be helpful to get a small flashlight and experiment with shining light on and near Wink’s nose. He will be most sensitive to the light if you shine it direct at the side of him along the surface (rather than straight above) because his sensors are most sensitive looking out along the surface he is sitting on.

Once you experiment (and probably notice a few strange things he does), we’ll talk about it in more detail. If you notice any strange behavior, take a minute and really think about what is happening in the code and why the “strange” things may be happening. Often, a program may do things you didn’t expect so it’s good to begin thinking your way through these things.

## Discussion about Simple Light Seeking...

So now you've written code that is making Wink do something "smart". Let's talk about a few things you may be noticing about the behavior.

**Endless Spinning:**

Wink sometimes will just spin in a circle. Why would this be? If the room lighting is fairly even, then he may never really see one side as being brighter than the other. In our code, we haven't told Wink to "be still if the two light readings are almost equal", instead, we've told him to rotate one direction or the other no matter what. Even if the readings are identical, the code in "else" will be executed each time loop() is run, which will make him always rotate one direction or the other.

Another reason for this, is that the light sensors aren't perfect and they aren't exactly identical. During manufacturing, one will tend to be a bit more sensitive than the other. This could be corrected by using high cost sensors and filters, but even then, they may never read exactly the same, even under perfectly even lighting.

If you want a challenge, you can use the Serial.print() functions we learned before to tell you what the sensor readings are. (You may want to "comment out" the motor code while you do this so he doesn't try to spin. You can do this by adding a pair of // forward slashes right before the two motors() lines. This will make these lines turn into comments and not compile. This is a common way to temporarily remove code while testing without actually deleting it). View the values of leftLight and rightLight and try to get nice even lighting on Wink, so both sensors are getting about the same amount of light. Then look at the readings on your screen. You may notice that one is consistently a bit higher than the other. You can then add an extra line to your code after the reading to account for this. For example, if the leftLight is always about 5 counts higher then rightLight, you can add a new line before the if CS to always add an extra 5 to rightLight to compensate.

**Wink twitches side to side while looking at a light:**

This is called an "oscillation". When something is "oscillating" it is moving back and forth between two things. Wink will see the light stronger on one sensor then start to turn that way and he'll build up a bit of rotational speed doing this. As he rotates, he'll move past the light then the opposite sensor will start to see the light and try to reverse the movement. Because he's already got a bit of speed he tends to over-shoot a bit then goes back the other way. This repeats over and over, first one direction, then the other. This can be compensated for a few ways. One way is to change the speed of the motors based on the difference between the two sensors. This is a more advanced topic that we'll cover in the Skill Level 3 lessons.

**Wink tends to "lock heading" in a certain direction, then switch to a different heading:**

Wink may naturally lock onto a light source in a direction without beginning to oscillate. Then if a shadow passes his sensors just right, he may find another source in a different direction and lock on to that source. It really depends on the exact light levels in different directions in your room. Try to run him with nice even room lighting and wave your hand over him to cast shadows and you may see this happen.

## Discussion about formatting...

You may have noticed that some lines of the code are indented. This can be done by pressing the TAB key on your keyboard. The C language doesn't care if you indent or not, but it makes the code much easier for humans to read if you follow a few standard rules about indenting your code.

Normally, whenever you use a control structure (like "if"), you will line up the "i" in the word "if" with the closing curly of the if CS. You will also align this with an "else" statement if used, as well as the closing curly of the "else".

You will also notice that the code inside the curly braces is also indented yet another step. This helps you visually see what blocks of code go together. You can have several "if" statements inside other "if" statements (or other control structures). This is called "nesting". Normally each "nested" control structure is indented yet another step so your eyes can quickly see what code is included inside what control structure.

Also note that the loop() function has a pair of curly braces, one at the very start and one at the very end. Each "if" also has a pair of curly braces, and each "else" has a pair of curly braces. It is important to always remember to pair them up. If you forget one of your braces, your code won't compile because the computer doesn't know where one block of code starts or ends. By using the formatting rules I have suggested, you'll also be able to more easily identify where you may have forgot a curly.

## Using "else if" for even more control...

I mentioned at the start of this lesson that "if" is very powerful. We need to mention yet another way you can use "if". In our first example, we did a simple "if", then we gave the program a second option to run if the condition of that "if" was not true. Well there's yet another option you can use. It is called "else if".

"else if" tests a second condition, which is considered if the condition of the first if was not true. Have a look at this code then we'll discuss...

```
birds = 5;

if (birds == 10)
{
  eyesRed(100);
}
else if (birds < 10)
{
  eyesGreen(100);
}
else
{
  eyesBlue(100);
}
```

} If birds equals 10, eyesRed(100); is run, and the code in "else if" and "else" is skipped.

} If the first "if" condition is not true, then the next "else if" in line is considered, and if that condition is true, the code is run.

} If the condition in the first "if" was not true, and no condition in any "else if" condition was true, then whatever is inside "else" will be run as a last resort.

2 SKILL LEVEL

The "else if" is like a second "if" option under the first "if". If the first "if" condition is not true, then the program moves on to the first "else if". If that condition is true, then the code inside is run and the entire "if" process stops and the code continues down below the closing "else" curly.

It's almost like saying "do you want a chocolate cookie?", and you say "no". As a second option I then ask you "would you like a strawberry cookie?". If you say "yes" then the cookie selection is complete and you go off and eat your cookie. If you say "no" to this strawberry option, I may then hand you a vanilla cookie if that's the flavor we're giving to all the people that don't want chocolate or strawberry.

You can use as many "else if" conditions that you want. You can have a first "if" condition, then follow it by many "else if" conditions, then at the end a final "else" that will be run if none of the "if" or "else if" conditions were true.

Like this example...

```
buttonPress = 2;

if (buttonPress == 1)
{
  eyesRed(100);
}
else if (buttonPress == 2)
{
  eyesGreen(100);
}
else if (buttonPress == 3)
{
  eyesBlue(100);
}
else
{
  //do nothing
}
```

} In this case, eyesGreen(100); would be run.

} In this case, we don't want to do anything for "else". You can leave off "else" completely, or you can just leave a comment here stating that you're "doing nothing".

Note that "else" isn't actually required. It technically makes your code a tiny bit smaller to leave it off (and some compilers are smart enough to not include it), but I often will still include it as a note to myself and other human readers that we don't want to do anything if the first "if" and "else if" conditions were not true.

2
SKILL LEVEL

## Challenge: Add a "dead band" to light seeking...

I know this has been a really long lesson. If you're head is starting to spin then take a break and play for a while. Come back when you're ready for a final challenge.

Let's combine "if", "else if" and "else" to add a "dead band" to our light seeking. A "dead band" is a term used in robotics and electronics to refer to an area where no reaction happens. Remember in our previous example, if Wink saw perfectly even light from both sensors, he would still spin because we didn't give him any option to just sit still.

In this challenge, we're going to allow Wink to spin one way toward a light, or spin the other way toward a light, or, if the two light sensors are reading almost the same values, he won't spin at all and will instead just sit still.

If you're up for it, try to figure it out yourself first then look at this example.

```
int leftLight, rightLight;    //declare the variables

void loop(){

  leftLight = analogRead(AmbientSenseLeft);    //read left
  rightLight = analogRead(AmbientSenseRight); //read right

  if (rightLight-10 > leftLight)       //if right is greater
  {
    motors(100,-100);                  //spin to the right
  }
  else if (leftLight-10 > rightLight)  //if left is greater
  {
    motors(-100,100);                  //spin to the left
  }
  else                                 //otherwise...
  {
    motors(0,0);                       //be still
  }

} //end of loop()
```

*Wink_Ch09If_Ex05*

} Declare the variables we will need

} Read both sensors.

} If rightLight is more than 10 greater than leftLight, then run code to make motors spin and rotate Wink to the right.

} If leftLight is more than 10 greater than rightLight, then run code to make motors spin and rotate Wink to the left.

} If neither of the above are true, then the light level must be fairly even. This is the "dead band". In this case, make the motors stop.

2
SKILL LEVEL

## Challenge Discussion...

Study the example and try to understand what is happening.

I've introduced another new idea to this example. Notice that I'm doing math right inside the condition of the "if" and "else if". How cool is that? The way this works is that the program first does the math, then uses the result of that math for the comparison.

In this case, in the first condition, we ask if it is true that "rightLight minus 10 is greater than leftLight". Think about that for a minute. In order for this to be true, the reading we got off the right sensor must have been at least 10 greater than the left. This is because we're taking the rightLight value and subtracting 10 from it, then making the comparison, and if this is still greater than leftLight, then this condition is true.

You can do all kinds of math right inside the condition itself.

Another (more complicated) way you could have done this is to create another variable, then do the math outside the condition before the "if", then use the new variable inside the condition, but this is much more complicated and also uses more of Wink's memory. For this reason it is considered best practice to do this sort of math right inside the condition like in the example.

## Experiment with the values...

So now that we've got it working, try experimenting with the values. Play with the motor speed numbers. What if you set them all to 250? Have fun with that because Wink is going to be hard to control as he'll want to constantly over-shoot when spinning. Try it anyway and you'll see what happens. Also play with the dead band numbers. What happens if you change the dead band from 10 to 1? What if you make it 50, or 200?

Experiment with these numbers and a flashlight to see how they change Wink's behavior.

## Changing all the values at once...

If you did very much experimenting with numbers in the previous example, you may have noticed something rather annoying. In order to change the spinning speed of the motors, you had to change the value in four different places, and if you forgot to change it in one place, instead of spinning, Wink drove in an arc. And if you wanted to change the dead band, you had to change that in two places.

Sometimes, especially with robotics, you’ll need to “tune” how a section of code runs. For example if you’re making a robotic arm move, should you run it at speed “10” or speed “500”? You may not know the best speed until you try out a few different values. Maybe you discover that anything over “500” makes the arm move so fast that it falls over, or breaks parts. You may want to limit the maximum speed of your robot arm, and that limit may be written in 37 different places in your code. Imaging changing all those values one at a time!

In applications like this where you have a value that you plan to experiment with, it is good practice to define the value as a variable, then assign the value you want to that variable, then use the variable in place of the actual value through your code. If you do this at the top of your code, you can make the change in one place and it instantly works through your entire program. It is also good to make the comments for this value really obvious so that you, or some other user of your code, knows exactly what values to adjust, and what they control. You can also include a suggested starting point or a suggested range for the value in the comments.

```
int dBand = 10;   //change this to adjust dead band
int mSpeed = 100; //change this to set motor spin speed

int leftLight, rightLight;    //declare the variables

void loop(){
  leftLight = analogRead(AmbientSenseLeft);    //read left
  rightLight = analogRead(AmbientSenseRight); //read right

  if (rightLight-dBand > leftLight)       //if right is greater
  {
    motors(mSpeed,-mSpeed);               //spin to the right
  }
  else if (leftLight-dBand > rightLight)  //if left is greater
  {
    motors(-mSpeed,mSpeed);               //spin to the left
  }
  else                                    //otherwise...
  {
    motors(0,0);                          //be still
  }
} //end of loop()
```

*Wink_Ch09If_Ex06*

Declare dBand and mSpeed while giving them both a value. Comments are added so we know what these values adjust.

You can now change these single values at the top of your code to tune the behavior of the code below.

Insead of writing the number “10”, we use the dBand variable instead. The same with motor speed, instead of writing “100” we use the mSpeed variable instead.

2 SKILL LEVEL

## Summary...

Wow. That was a long lesson, but you've learned one of the most important concepts of any programming language - how a computer can make a choice based on any condition.

This is super important to understand and become familiar with, so you may want to run through this lesson a few more times. If it didn't all make sense the first time through, then take a break for a day or two and go through it again when you're refreshed.

Before we get to the next lesson, have some fun and play with what you've learned so far. What else can you do with "if" and the other stuff you've already learned? Can you make a room alarm that makes noise when someone turns on the lights? Can you make Wink drive at different speeds based on how bright the room is? (Keep in mind for this one that the light sensors go up to 1024, where motor speed only goes up to 255. You may need to do some math here).

What happens if you operate Wink's eyes while trying to read the light sensors or doing the light seeking examples in this lesson? You may notice a few problems. How could you solve these problems? We'll cover this in detail in an upcomming lesson but it could be a good challenge to try to figure it out on your own.