Loop de Loop — for()

The `while` based loops are pretty common and quite flexible, but sometimes a simpler mechanism is needed. And something that's maybe a little easier for the programmer to read and understand. With a do/while loop, changes to the variables used in the conditionals can take place anywhere inside the statement block — so one has to scan down through the entire block to determine when and where the various changes that would trigger the loops termination might occur.

The **`for`** loop attempts to address a few issues at one time by grouping initialization, comparison, and iteration all on a single line.

The **`for`** loop construct is pretty simple in concept, but the syntax is a little unusual looking. Take a look at the example snippet below which prints 5 numbers, 0 through 4:

```
for ( i = 0; i < 5; i++ ) {
      Serial.println(i);
}
```

While that looks a bit cryptic, it's actually pretty straight forward. After the keyword **`for`**, there's a grouping of 3 items inside parenthesis. The first item is the starting condition for the loop and it is performed only once, before the loop begins — much like the `setup()` area in our IDE. The next item defines the terminating condition; when it evaluates to False, the loop will end. In this case we are comparing the value of the variable ***i*** to see if it is less than 5. The last items is a special statement called the ***incrementor***, it will be performed after each iteration of the loop occurs.

After the closing parenthesis, a code block is created with an opening brace, and any statements before the closing brace will be executed for each iteration of the loop.

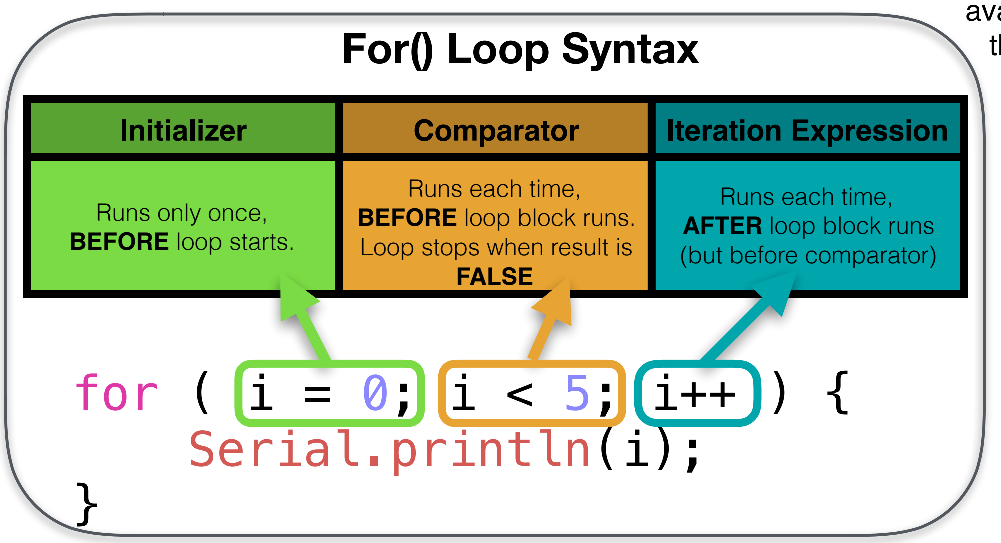
In this case, the variable ***i***, which was previously declared, is set to 0 before looping begins. Then ***i*** is compared to 5, it is still 0 and thus less than 5, so the statements inside the braces are executed. Here, that is just the `Serial.println()` statement which will output the contents of the variable ***i*** — currently 0. Lastly the incrementor is executed, increasing the value of ***i*** by 1. Now the process starts all over again.

Of course, ***i*** could be modified in the statement block, though that would sacrifice later readability by another programmer.

The increment does not have to be a single ++, it can be a complex statement involving other variables. Similarly the comparison doesn't not have to be a simple single variable numeric comparison. Consider the following example:

```
for ( int timer = 0 ;
      digitalRead(BUTTON1) == HIGH;
      timer++ )
{
    Serial.print("no button push after ");
    Serial.print(timer);
    Serial.println(" loop cycles");
}
```

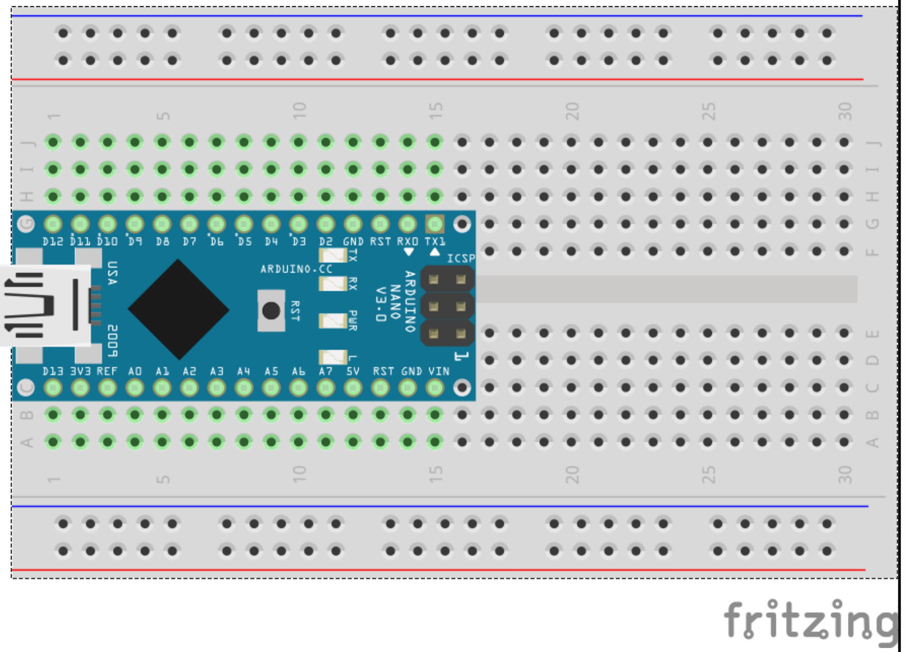
First note that while the formatting is different, it is the same command, we have just split the Initializer, Comparator, and Iteration Expression across several lines. The compiler doesn't care about lines, spaces, etc. It is simply looking for the appropriate symbols. But *people* do. And sometimes very long lines can be hard to read. Notice also, that, not only is the variable '`timer`' being initialized to 0, but it is also being *declared* as part of the initializer section as well. This has two implications, first you do not have to move back up in your program and declare it, secondly it is ***only*** created and available for use inside this for loop. After that it will be destroyed.


For() Loop Syntax
Initializer
Runs only once, BEFORE loop starts.
Comparator
Runs each time, BEFORE loop block runs. Loop stops when result is FALSE
Iteration Expression
Runs each time, AFTER loop block runs (but before comparator)
for (i = 0; i < 5; i++) {
Serial.println(i);
}

If we assume we have a push button setup to return LOW when pushed as we did previously, this will continue to loop until the button is pressed, incrementing the 'timer' variable every time though.

Once the for loop's comparison condition is no longer True, the loop stops and execution continues with whatever statements are next in the program.

Experiment: for() he's a jolly good fellow

Components	**Wiring Diagram**
✓Computer ✓Arduino IDE	

Connection Instructions
No connections required, we will be using the internal LED for this example.

Sketch(es)	forSOS.ino

Analysis Questions
What are some benefits of reducing redundancy in your program? Do you think there are other tools which will help further simplify redundancy?

Programming Tasks
Remember the morse code experiment, and how repetitive the code was? Use some 'for' loops to make it a little more concise.