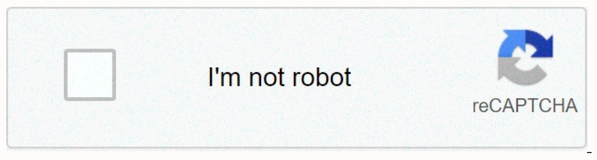

Continue

If statement arduino

by Lewis Loflin This is part of a series about snippets for Arduino. Many visitors to my You Tube Channel and this site are beginners. They have limited knowledge of programming or hardware. This requires learning both. Think of a microcontroller as a box full of basic logical circuits, ports, etc. To check the box, we need to tell it which hardware to use. We have to tell the box how to manipulate the ports and hardware. that's what machine code does. I use compiler Arduino-1.6.3. Results may vary with other compilers or a non-Nano Arduino board. Fig. 1 shows the test setup for this series, in this case an Arduino Nano. I guess you can program your arduino board. Nano and most Arduino boards currently have an LED on digital pin 13 (DP13). The use of the I2C LCD screen is optional, but makes it easier to understand the process. This uses three LEDs on DP9, DP10, DP11. Two normally open push button switches are connected to DP2 and DP3 to the ground. Internal pull ups are used - the switch closed reads as fake or 0. Instead of constantly writing digitalRead (PIN) I created subroutines S1 () and S2 (). If either S1() or S2() is called a closed switch returns 1 or open switch returns 0 - they also update the LCD screen. I also wrote a subroutine called POT () that returns a value of 0-1023 from the potentiometer connect to analog pin 0. The full layout of the chart above appears at the bottom of the page. This can be cut and glued directly to the Arduino compiler. In the electric sense is a HIGH or 1 5-volt; a LAV or 0 is 0-volt or painted. And true can be replaced by 1 or any non-zero number; false can be replaced by 0. We are concerned with the following: Comparison operators == (equal) != (not equal to) < (less than) > (greater than) <= (less than or equal to) >= (greater than or equal to) Boolean operators & (and) || (or) ! (not) From here I'm just going to record the code within the loop(). invalid loop() { delay (100); if (S1()) digitalWrite (LED1, 1); other digitalWrite (LED1, 0); } Test 1 Note: true = 1; false = 0. Note the code above. An if statement has the general form of: if (condition) light_led1; otherwise turn_off_led1; The Other section is optional. Condition is Boolean term using true or false A true state light LED1, a false state turns LED1 OFF. The Arduino compiler defines true as the word true, the number 1, or a non-zero number. The compiler defines false with the word false or the number 0. The above code calls function S1(). If the switch on DP2 is open, it returns 0 or false - thus other is performed and LED1 is turned OFF. LED1 is only ON while the S1 is pressed by returning a 1 when called. invalid loop() { delay (100); if (S1()) digitalWrite (LED1, 1); if (S2()) digitalWrite 0); } What is it we want to press a switch (S1) to turn LED1 LED1 saw another (S2) to turn LED1 off? We need to exclude the second part of the if function and use another if function. Understand if (S1()) is understood to be if (S1() == 1) On each iteration of loop S1() is called and returns a 1 (true) or 0 (false). Another way to write this to do the same is if (! S1()) turns OFF LED1 when pressed, ON when released. It! means a logical NOT. A true one has been changed to false, a fake has been changed to true. What was fake before holding LED1 OFF is now true leaving it ON. A logical NOT is the same as the little XOR. if (S1() ^ 1) is the same as if (! S1()). Note the snippet above. Press the S1 LED1 turns on and stays on. Press S2 and LED1 turns off and stays off. (This can also be an engine control for example.) In each case, press S1 or S2 returns a 1 or 0; command if do the rest. Another note is () must be used correctly and in sets of 2nd void loop() { if (S1()) { byte temp = digitalRead(LED1); temp = !temp; temp ^ 1 digitalWrite(LED1, temp); } // end if } // end loop Test 3 In test 2 I used 2 switches to control 1 LED. What if I only have 2 switches and have to check 2 LEDs? Now it's time for electronics. The LED IO pins were programmed as OUTPUT back in setup(). A digitalWrite() sends either a HIGH or 1 to the pin that switches it to 5 volts that turns on the LED. A 0 or LAV switches the pin to grounding turn off the LED lamp. This 0 or 1 bit is stored in a hardware lock circuit connected to the physical pin LED is connected. Even if a pin is programmed as an output with pinMode(), digitalRead() can read the lock value that returns a 1 or 0. Let's go through the snippet above. Line 1 if (S1()) { is our if function plus an opening {. 6 lines down is } // end if close brace. This is required are several commands used is S1() is true. There must always be a final curly buckle for each opening curly buckle. It's a good idea to label all the closing braces to keep the confusion down. Line 2 byte temp = digitalRead(LED1); reads the lock pin state for LED1. This 1 or 0 is stored in variable temp. Line 3 temp = !temp; Temp ^ 1. ! temp will NOT or invert the value of temp. 1 becomes 0, 0 becomes 1. This is the same as temp = temp ^ 1. See Arduino XOR Flashing LED. Line 4 digitalWrite (LED1, temp); will write the opposite or reverse lock value back to the LED1 lock. Let's take a closer look at lines 1-4. Line 2 can be rewritten as follows and excludes the need for line 3: byte temp = !digitalRead(LED1); You can NOT value digitalRead(). Or an option: digitalWrite (LED1, !temp); I NOTed variable temp in digitalWrite() command which is also legal. This also eliminates the need for line 3. void loop() { if (S1()) { byte temp = !digitalRead(LED1); temp); wait for the switch open while (S1()) { } } // end if } // end loop Test 4 I have rewritten the original code eliminate line 3 but added a new twist. When you press S1, I want to switch between the LED1 lock. I will not leave if the statement until I release S1. while (S1()) { } does just that. One while command checks the state of the S1() and returns true if I press the switch. It will inactive in an endless loop do nothing ({}) until I release the switch. Below is the code to turn ON and OFF LED1 and LED2 with 1 switch each. invalid loop() { delay (100); if (S1()) { byte temp = !digitalRead(LED1); digitalWrite (LED1, temp); wait for the switch open while (S1()) { } } // end if (S2()) { byte temp = !digitalRead(LED1); digitalWrite (LED1, temp); wait for the switch open while (S2()) { } } // end if } // end loop void loop() { if (S1()) digitalWrite(LED1, HIGH); other digitalWrite (LED1, LOW); if (S2()) digitalWrite (LED2, HIGH); other digitalWrite (LED2, LAV); if (S1() & S2()) digitalWrite (LED3, 1); other digitalWrite (LED3, 0); } // end loop Test 5 With the code above if S1 is pressed on LED1 is ON until S1 is released. If the S2 is pressed, the LED2 lights up until S2 is released. The third if statement is different. Here we use a logical AND statement using a double & or &. A single & is a bitwise function and will produce a compilation error. Here we check S1 OG S2. Only when both are true (return 1) is LED3 switched on. A problem occurs when both S1 and S2 are pressed, all three LEDs are turned on. I just want THE LED3 turned ON. I want the other 2 LEDs to stay off during the process. On the next page we will program a complete engine control program for Arduino. It will operate an H-Bridge with speed control in both directions. Or will operate two single motors with independent speed control for each engine. It will also have a master ON-OFF power control. It will go into several uses for if-else statement. invalid loop() { delay (100); if (S1()^1) digitalWrite (LED1, HIGH); other digitalWrite (LED1, LOW); if (! S2()) digitalWrite (LED2, HIGH); other digitalWrite (LED2, LAV); } // end loop Test 6 This is a variant of Test 5 to illustrate a logical NOT and bitwise XOR. They do exactly the same, and when compiled use the exact same amount of memory. Functions S1() and S2() both return a true (1) when pressed. A NOT or XOR becomes a true against false, false to true. In both sentences S1(), S2() when you do not press return a fake. In both cases with NOT and XOR, this changes to true turn on LED1 and LED2. When one of the switches is pressed, they return a true one that is NOTed or XORed to falsely turn off the LEDs. Videos: My YouTube Videos on Electronics Introduction to Arduino Part 1: Programming Arduino Output Part 2: Programming Arduino Input Part 3: Arduino Analog to Digital Conversion Part 4: Using Arduino Pulse-Width-Modulation Repost Arduino AC Power Control If() statement is the most basic of all programming control structures. It allows you to make something happen or not, depending on whether a given condition is true or not. It looks like this: There is a common variation called if-else that looks like this: if (someCondition) { } other { }It is also the second - if, where you can check another condition if the first one is false: if (someCondition) { } other if (another Condition) { }You want to use if sentences all the time. The following example turns on an LED on pin 13 (the built-in LED on many Arduino boards) if the value read on an analog input exceeds a certain threshold. Hardware RequiredArduino BoardPotentiometer or variable resistanceCircuitclick image to enlarge the image developed using Fritzing. For multiple circuit examples, see the Fritzing project pageSchematicclick image to enlarge the CodeI code below, a variable called analogValue is used to store the data collected from a potentiometer connected to the tray on analogPin 0. This data is then compared to a threshold value. If the analog value turns out to be above the set threshold, the built-in LED connected to digital pin 13 is turned on. If analogValue turns out to be < (less than) the threshold, the LED will remain off. See AlsoLast revision 2015/07/29 of SM Last revision 05.