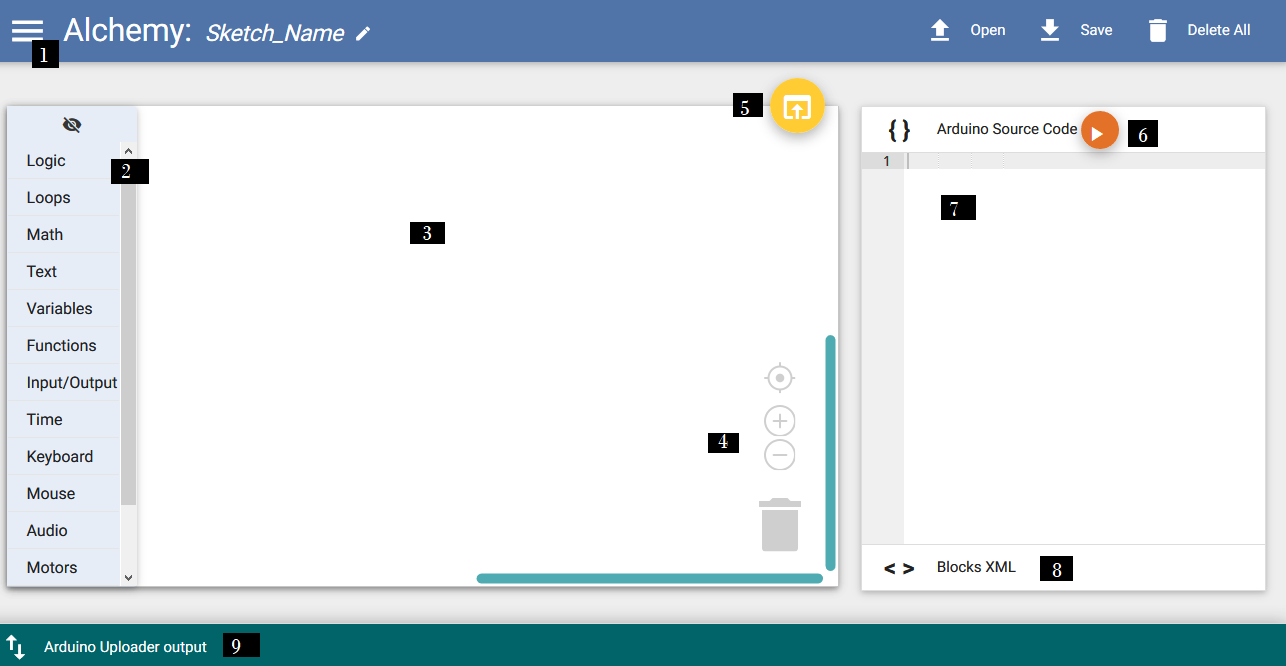
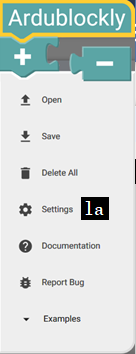
ALCHEMY   
USER GUIDE

# How to use the interface.

A quick overview of the user interface.



## 1: File menu.

Clicking this  menu icon opens the file menu (as shown at left). “Open” loads a program from the user’s file system. “Save” likewise stores a program to the file system. “Delete All” clears the workspace. “Settings” is used to configure your Arduino board to receive programs. (see 1a)

“Examples” allows you to load some premade example programs, these will be outlined later in this document.

## 1a: Settings.

This settings dialog is used to identify the connected board so that the program can be sent to the board.

If you’re trying to configure a new board or having trouble uploading a program to the board, you may need to adjust some of these other settings. Ensure that the “Arduino Board” setting corresponds to the board you are using for development, and that the correct COM port is selected.  
  
which COM ports are in use can be checked using the Device Manager utility in Windows, or on Mac or Linux using the terminal, run ls on your /dev/ directory,   
to find it

## 2: Block toolbox.

This menu is broken up into several collapsible categories that each contain blocks for use with the block-based editor. The block toolbox contains all of the blocks you’ll need to construct a visual program. Drag a block from the toolbox into the block canvas (3) and you will see the corresponding code generated in the code editor (7). The use of these blocks is outlined in the section “Block Library” later in this volume.   
This ‘eye’ icon  shows and hides the block toolbox.

## 3:Block Canvas.

You will use this space to construct your visual program.   
  
The canvas is an infinite space that will expand to contain any number of blocks. In order to navigate around the canvas, you can click and drag on the background to pan over the available space. The scroll bars at the right and bottom of the canvas will also allow you to scroll around the canvas.  
  
Drag blocks into the canvas and they are added to the program. You’ll notice that the blocks are shown with jigsaw-shaped edges that demonstrate the ways that they can fit together. Drag two blocks that fit together onto one another, and they will snap together to form a larger block-construction. Blocks that are joined in this way can be moved as a unit, that corresponds to a connected section of code, that will be executed together in the resulting application. This “connect” operation allows you to compose custom programs to do almost anything, by connecting logical structures together.  
  
****   
4:Block Controls.  
The “crosshair” icon recenters the canvas, to show a central overview of the program. You can zoom in and out with the ‘+’ and ‘–‘ icons, to see more or less of your program. The “waste bin” icon is used to delete unwanted blocks from the program – drag a block into the waste bin to delete it.  
  
5:Upload Sketch to Arduino.  
 Use this “upload” button when your program is ready to test on the board. Alchemy will verify the current program, compile it using integrated build tools, and upload it to the board.

## 6:Translate Code to Blocks.

When you make changes to the code, click this”update” button to store and translate those changes. Your changes are commited and can be seen in the block editor and in the native XML view of the program.

## 7:Code Editor.

You can write native arduino programs as source code in Alchemy’s code editor, and it converts your program back into blocks. Use this code editor to compose and edit your program as native arduino code. The code editor is a user-friendly, traditional coding environment that will be relatively easy for new users to manipulate, and familiar for users experienced with writing code.  
This ”update” button commits your changes and updates the block and xml views to match your updates.

## 8:XML Output.

Expand this view to view or edit the raw XML representation of the program. You can use this view to paste a raw XML source file like those listed in the “Example Programs” section of this document.

This ”puzzle-piece” button updates the block and code editors to match any changes in the XML view.

## 9:Error Console.

If there are any errors in the process of uploading the sketch to the board, they are displayed in this expandable error console.

# Arduino Documentation

The official arduino documentation is maintained at <https://www.arduino.cc/en/Main/Docs> and is a comprehensive source of information about the arduino platform and programming language.

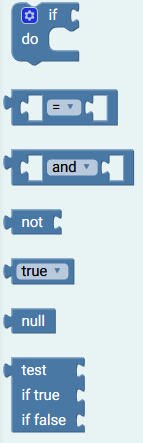
# Ardublockly Documentation

Alchemy is built on top of Carlos Pereira Atencio’s Ardublockly project, The Ardublockly user documentation is spotty but still largely relevant for use with Alchemy and is available at <https://github.com/carlosperate/ardublockly/wiki>

# Block Library

## Logic

Boolean variables are used to track data that is either true or false, and can be used as conditions to alter the flow of your program. Use boolean logic to control a branching structure.

If: If a given condition value is true, then execute the stack of blocks attached under this “if”. This allows you to create a branch within your program that is only executed when the condition is met, and returns true – and not executed otherwise.  
  
Use the “gear” icon to attach “else” and “else if” blocks to this If block. This allows you to create a structure with many branches, each controlled by a condition. An If block can have only a single “else” clause that is performed last, only if no other branch is executed.

Comparison: For the equal operator “=”, this block is true, if the first input is equal to the other.   
Use the dropdown on “=” to select other logical comparison operations. Many logical comparison operators (equal, not equal, less than, less than or equal, greater than, greater than or equal) are supported.

Logical Operation: An “and” block is true if both of its inputs are true.  
An “or” block is true if either one of its inputs is true, or if both of its inputs is true.

Not Operator: Inverts the logical value of the block given on the right. A true input becomes false, a false input becomes true.  
  
Boolean Value: Creates a true or false value.­

Null Value: Null is used in place of a value, for blocks and expressions that have no value or have not been assigned a value. Use this in comparisons to check if a value is unassigned.  
  
Test Block: Creates an if/else statement. If the condition is true, executes the “if true” block, otherwise executes the “if false” block.

### 

## Loops

Use Loops to create repeating structures.

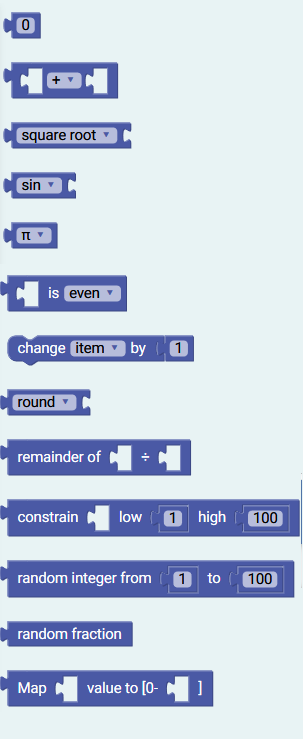
Repeat Block: Executes the stack of blocks attached, a specified number of times, in a row.

Repeat While/ Until: A “Repeat While” block executes the stack of blocks attached, while the given condition is true, until it becomes false.  
  
A “Repeat Until” block executes while the condition is false, until it becomes true.

Iterated Loop: use an iterator value, here ‘i’, to keep track of how many times the loop should execute.

Break/Continue: A “break” block exits out of the loop completely.   
A “continue” block moves to the next iteration of the current loop, increasing any iterator, and continues from the start of the loop.

## Math

Use math blocks to store numbers and do arithmetic.

Number Value: Creates a new number value with the specified contents. If no content is specified it defaults to 0.

Basic Maths: Allows uses to run simple arithmetic on numbers such as +, -, *×* and **÷.**

Advanced functions: Provides access to more advanced math function such as square root, absolute value and logarithms.

Trigonometry functions: Provides access to trigonometry functions such as sin, cosine and tangent.

Constants: Block contains some basic constants used throughout mathematics such as pi and e.

Even / Odd: Checks whether a block is even or odd.

Change Value: Increases or decreases a number value by a set amount.

Constants: Rounds a number either up, down or to the closest whole number.

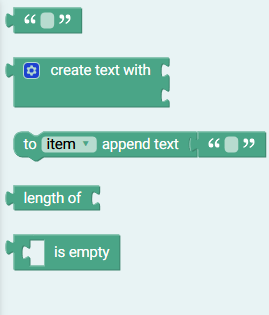
Remainder: Determines the remainder of a division operation, this is also called the modulo function

Constrain: rounds the number to fit between two specified numbers

Random Integer: Returns a random whole number between two specified numbers

Random Fraction: Returns a random number between 0 and 1

## Text

Use text blocks to store text data or to display readable messages to the user.

Text Value: Creates a new text value with the specified contents. If no content is specified, creates an empty text value.

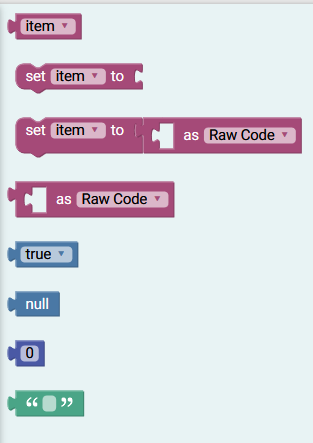
Concatenate: Join two or more text values together. Use the “gear” icon to add more inputs.

Append: Both “item” and the attached block are combined and stored as “item”.

Length: Counts the number of characters used to display a text value.  
  
Empty: Is true if the input is a text is empty - that has a length of zero characters.

## Variables

Use variables to store data values and retrieve these values elsewhere in your program.

Get Variable: Retrieves the data value associated with a given name.

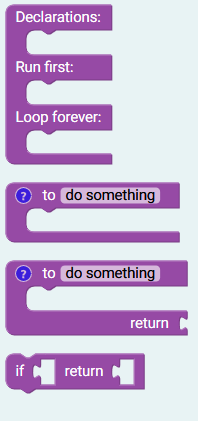
Set Variable: associates data with a given name. Data stored in this way is persistent and available elsewhere in your program.

Cast Type: Stores data as the chosen type (text, raw code, text, Boolean, or any numeric type). You can use this to specify types in places where data changes type or its type is ambiguous.

Type Literals: Boolean, null, numeric and text blocks are explained in their own categories, but the variables category is a handy place to find them, provided together for convenience’s sake.

## Functions

Use functions to create blocks of code that can be called and executed from elsewhere in your program.

Main Loop: This creates the main structure of a program. Anything listed under “Declerations” is run at the start of the program. Use this space to create new named variables. “Run first” is executed once, pput any code here that you would like to use to create an initial state each time the program is run. “Loop forever” then runs continually.

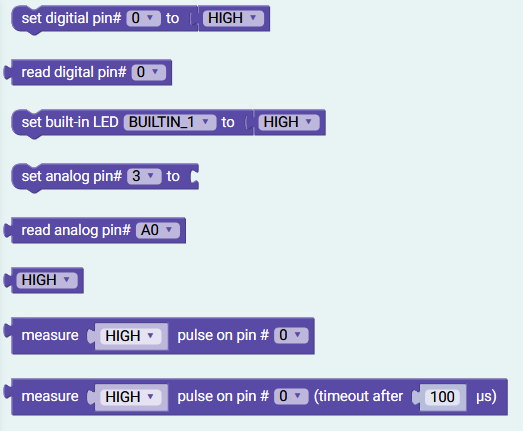
Function: Creates a named section of code that can be executed from anywhere. Any functions created in this way can be called from a named function block that will be dynamically added to this section.  
  
Return Function: Creates a function that runs, then returns some value. Functions can be used in this way to perform calculations that are repeated in your code, to simplify and make callable the repeated sections.

Return: Used in function that may return different values under different conditions, or which return early when a condition is met.

## 

## Input/Output

Use the blocks in this section to read and write to GPIO pins on the arduino, or to control the built in LEDs on the board.

Set pin: Sets a pin either high or low

Read pin: Checks wether a pin is high or low

Set LED: Turns the built in Arduino LED’s on or off

Set analogue pin: Sets an analogue pin to a desired number value

Read analogue pin: Reads the value of an analogue pin

High/Low value: High / LOW value block

Measure Pulse: Check for a pulse on a specified pin

Measure Pulse (timeout): Checks for a pulse on a specified pin, but stops waiting after x microseconds

## Time

Use these blocks to insert delays into your program or to keep track of how long the program has been running.



Wait milliseconds: stops the program for a set number of milliseconds.

Wait microseconds: stops the program for a set number of microseconds.

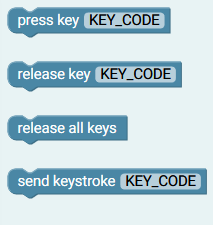
Elapsed time (milliseconds): Checks how many milliseconds have elapsed since the program started

Elapsed time (microseconds): Checks how many microseconds have elapsed since the program started

Wait forever: Stops the program running

## Keyboard

Use these blocks to create programs that simulate a keyboard over USB. The arduino keyboard library is not supported for all types of boards, and so these blocks are only suitable for use with a subset of boards, such as the Leonardo or Esplora boards. Note that if the keyboard functions are always in use, it may become difficult to program the board, and so keys should only be sent where you are ready to handle them. Perhaps use a kill switch or other condition in your program to prevent the board from always sending keys.  
  
Acceptable key codes are either single characters like ‘c’ or specific modifiers like “KEY\_SPACE”, For a list of acceptable key modifiers see: https://www.arduino.cc/en/Reference/KeyboardModifiers

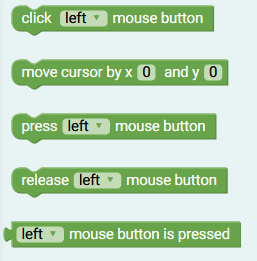
Press Key: puts the given key in the “down” position, as though it were held down on the keyboard, so that it always sends keys. Combine multiple keypresses to create key functions (“KEY\_LEFT\_CTRL” + ‘C’), etc.  
Release Key: puts the given key in the “up” position, as though not held on the keyboard, so that it sends no data.

Release All Keys: disables all pressed keys, so that no more keypress data is sent until a new key is pressed.

Send Keystroke: sends a single momentary keystroke of a given key, as though the key was pressed once on the keyboard and then released.

## Mouse

Use these blocks to create programs that simulate a mouse over USB. The arduino mouse library is not supported for all types of boards, and so these blocks are only suitable for use with a subset of boards, such as the Leonardo or Esplora boards. Note that if the mouse functions are always in use, it may become difficult to program the board, and so mouse input should only be sent where you are ready to handle it. Perhaps use a kill switch or other condition in your program to prevent the board from always sending mouse input.

Click Mouse: Send a momentary press of the chosen mouse button, as though the button was pressed and released.

Move Mouse: Adjusts the mouse position, based on its previous position. Use this to move the mouse cursor around the screen.

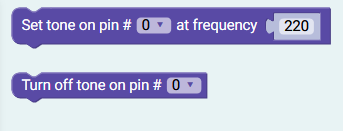
Press Mouse: Send a mouse pressed signal, as though the button was being continuously pressed.

Release Mouse: Send a mouse release signal, so that no more mouse press events are sent.

Is Pressed: Returns true if the given mouse button is pressed, false otherwise.

Audio

Use these blocks to send audio signals to buzzers attached to your device.

Set Tone: generates a constant tone on a specified pin at a set frequency

Stop Tone: Stops a tone from playing on a pin

Motors

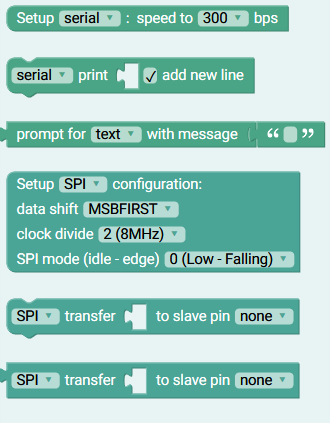
Use these blocks for projects that perform automated tasks with attached servo motors or stepper motors.

Set Servo: Sets the angle on the chosen servo motor. Tells the motor to actuate and turn to a specific angle.  
  
Read Servo: Reads the current angle of the servo motor attached on the chosen pin.

Setup Stepper: Initialises a stepper motor setup. The number of pins required to control the motor and the steps per revolution will vary with the motor. From here you can set the speed of the motor.

Move Stepper: Turns a stepper motor a designated number of steps.

Comms  
For more information about SPI (Serial Peripheral Interface) configuration please refer to the official arduino documentation at https://www.arduino.cc/en/Reference/SPI

Setup Serial: Creates a new serial connection with a specified baud rate.

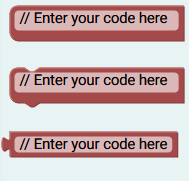
Serial print: Sends a specified value over the serial connection.

Serial Prompt: Sends a message to the local serial console and gets input to send back to the board.

Setup SPI: Creates a new SPI (Serial Peripheral Interface) configuration with the specified settings.

SPI Transfer: send data over a SPI connection.

Raw Code

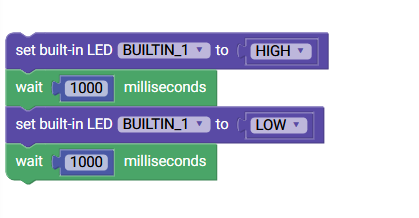
Use these blocks to write raw Arduino code in an otherwise block-based project. These blocks are used internally for transition from code back to blocks.

Raw comment: Use this to insert a comment. Comments are not executed but allow you to clarify for anyone who reads the code, what your program is doing.   
  
Raw code: Raw code block. Use this to insert any valid Arduino code into the block editor.

Raw value: Raw code block used for values.

# Sample Programs

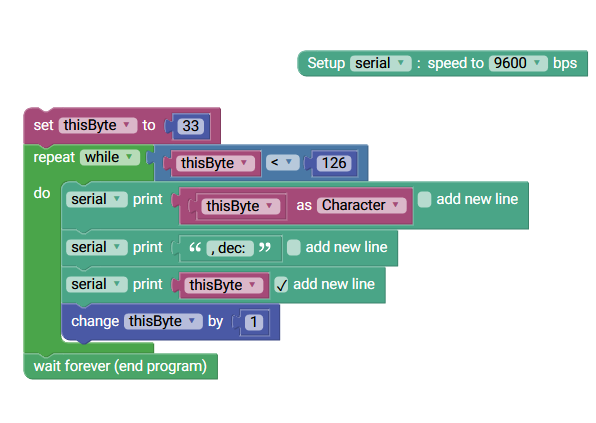
## Blink



This program, selectable in the file menu under examples, turns the built-in LED light on the board on for 1 second, then off for 1 second, and repeats.

## Blink Code

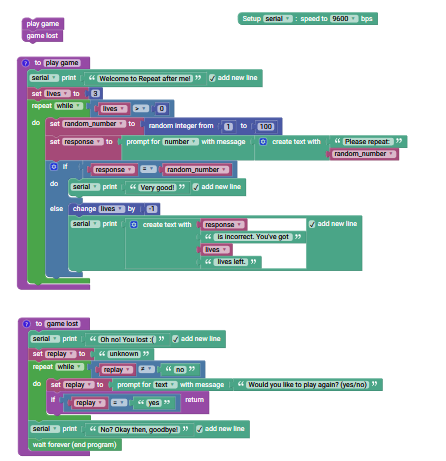
|  |
| --- |
| void setup() {  pinMode(13, OUTPUT);  }  void loop() {  digitalWrite(13, HIGH);  delay(1000);  digitalWrite(13, LOW);  delay(1000);  } |

Serial Repeat  
  
This program prints a series of ascii characters and their numeric representation to the serial console.

## Serial Repeat Code

|  |
| --- |
| int thisByte;  void setup() {  Serial.begin(9600);  }  void loop() {  thisByte = 33;  while (thisByte < 126) {  Serial.print((char)(thisByte));  Serial.print(", dec: ");  Serial.println(thisByte);  thisByte += 1;  }  while(true);  } |

## Serial Repeat Game

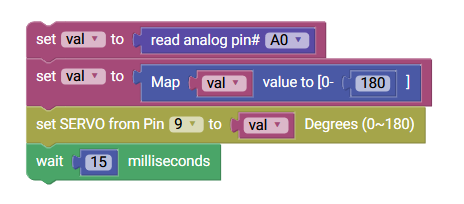


This program implements a small game that sends and receives input using the serial console. The program chooses a random number between 1 and 100 and prints it to console. The user is asked to repeat the number back. The program demonstrates the use of multiple functions and conditional branching structures: if the user responds incorrectly three times, they lose the game, triggering the ‘game lost’ function to run until the user chooses to play again, which returns control to the ‘play game’ function.

## Serial Repeat Game Code

|  |
| --- |
| int thisByte;  void setup() {  Serial.begin(9600);  }  void loop() {  thisByte = 33;  while (thisByte < 126) {  Serial.print((char)(thisByte));  Serial.print(", dec: ");  Serial.println(thisByte);  thisByte += 1;  }  while(true);  }int thisByte;  void setup() {  Serial.begin(9600);  }  void loop() {  thisByte = 33;  while (thisByte < 126) {  Serial.print((char)(thisByte));  Serial.print(", dec: ");  Serial.println(thisByte);  thisByte += 1;  }  while(true);  } |

## Servo Knob

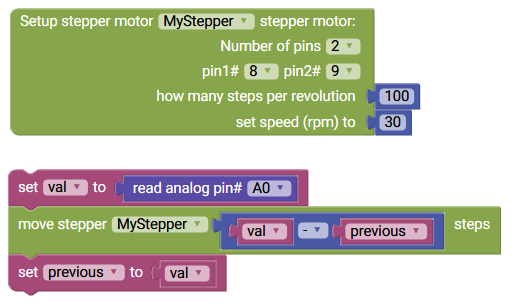


This program accepts analog input from an attached potentiometer knob or other analog input on pin A0 and outputs a corresponding angle to a servo motor on pin 9.

## Servo Knob Code

|  |
| --- |
| #include <Servo.h>  int val;  Servo myServo9;  void setup() {  pinMode(A0, INPUT);  myServo9.attach(9);  }  void loop() {  val = analogRead(A0);  val = (map(val, 0, 1024, 0, 180));  myServo9.write(val);  delay(15);  } |

## Stepper Knob



This program accepts analog input from an attached potentiometer knob or other analog input on pin A0 and outputs a corresponding number of steps to a stepper motor on pins 8 and 9.

## Stepper Knob Code

|  |
| --- |
| #include <Stepper.h>  int val;  int previous;  int MyStepper[2] = {8, 9};  Stepper stepper\_MyStepper(100, 8, 9);  void setup() {  stepper\_MyStepper.setSpeed(30);  pinMode(A0, INPUT);  }  void loop() {  val = analogRead(A0);  stepper\_MyStepper.step((val - previous));  previous = val;  } |