Standard Arduino program format:

Header - comments
Declare variables:
int x = 0;
int count = 0;

void setup()
{
// Initialize I/O here
// Include libraries
}

void loop()
{
// This is your main program!
}

Compiled by Tom Thoen, PCC

Arduino programs (sketches) are composed of **3 main parts**:

Variables - values or characters that change in the program
Loops / Branches – decisions based on comparing conditions or variable values
Functions* - built-in or user defined subroutines

These all work with **hardware I/O** (inputs and outputs) that **interface to the real world** to make an **embedded system**

* There are many functions built into the **Arduino library**

Variables:

	Examples:
int : covers large range of +/- values (+/ -32,767)	int loopCount;
boolean : either a zero or a one	boolean test;
float: large values, typically fractional	float bigValue;

Structures:

if **condition** (branch): tests for a condition and will run code if condition is true; if not skips code:

```
if (x > 5)
{
 // Do this code
}
// Otherwise continue down here
```

* Can also use an *else* condition

while **loop:** tests for condition and will run code until condition changes:

```
while (x  ==  1)          // note the double = sign is used to test a condition, not assign a value!
{
 //  Do this code as long as x = 1  :  Note:  does not change the value of x!
}
```

For loop: Advantages: resets to starting value, can use variables to modify starting and ending points.

```
for ( initial condition;  test;  action)
{
        Do this code
}
```

Example:

```
void loop()
{
 for (int i = 0;  i < 15; i++)                  // Notice that the variable is declared inside the structure!!
  {
     digitalWrite (LedPin, HIGH);
     delay(2000);
     digitalWrite (LedPin, LOW);
     delay(2000);
  }
}
```

Functions:

Functions are subroutines that are used repeatedly. They are the key to creating well-structured programs.

Types	Examples
built-in:	delay(500); millis(); *and lots of others we haven't talked about yet!!*
User defined:	user created, can be re-used in your own programs

Functions have different ways of using data:

```
Void:  void setup();                                  // nothing passed, nothing returned
Passed values:  delay(1000);                          //  The value 1000 is passed to the function
Returned values:  x = count();                        // the result x is returned
Passed and returned:  x = square(value);              // value is passed, x is returned
```

The function above is written like this:

```
 int square(int number)          //  the function is int since it returns a value.  It is passed number
{
     int x = number * number;    //   set x to equal number x number  ( * = multiplication in C)
     return x;                   //  return the value to the main program
}
```

Serial Communication:

```
if (Serial.available() > 0 )          // Check to see if a character is available from the keyboard;

Serial.print ("Hello People!");   // Print a character or variable without carriage return:
Serial.println("Hello again");    // Print a character or variable with carriage return:
ch = Serial.read();               //  Get a character from the serial monitor
x = Serial.parseInt();            //  Get a number (int) from the serial monitor
```

Digital I/O:

```
void setup()      // Configure inputs and outputs in setup:
{
  pinMode (3, OUTPUT);          // Use pin 3 as digital output
  pinMode (12, INPUT);          // Use pin 12 as digital input
}

digitalWrite(4, HIGH);          // turns pin 4 on.  Or, use a variable instead of 4

value = digitalRead (inputPin); // Read an input on pin inputPin
while (value == 1)              // Do something while input value = 1
```

Pullup configuration: For inputs configure the internal pullup resistor (switch to ground):

```
void setup()
{
   pinMode (5, INPUT);          // Setup pin 5 as input
   digitalWrite (5, HIGH);      // Configure with an internal pullup to 5V
}
```

You can also write data to pins 0 – 7 as a byte using PORTD command:

```
a = 75;
PORTD = a;                      // Write 75 in decimal to pins 0 - 7
```

ANALOG I/O - INPUTS:

Inputs: **Note 1:** Input value is scaled from 0 – 1023 for a 0 – 5V input:
Use Analog pins A0 – A5 for input **Note 2:** Nothing is required in Setup!

```
int sensorVal = 0;
const int analogInPin = A2;  // Analog input pin that the voltage signal

void loop()
{
   sensorVal = analogRead(analogInPin);   // Read the voltage from Pin A2 and store in sensorVal
```

ANALOG I/O - OUTPUTS:

Outputs:
Use Digital pins with ~ symbol

Note 1: Output value is scaled from 0 – 255 for a 0 – 5V output:
Note 2: Nothing is required in Setup!

```
int outputVal = 0;
const int analogOutPin = 10;      // Define what output pin used for an analog voltage output

void loop()
{
  analogWrite(analogOutPin, outputVal);          // Write the value of outputVal to pin 10
}
```

SCALING OF VARIABLES

y = map(*x*, *minimum value of x*, *max value of x, min value of* y, *max value of* y);

scales *x* to *y*

This allows the programmer to scale different variable sizes to fit into other sizes.

Example: You have an input sensor that reads from 0 to 1023, but you want the output to be scaled from 0 to 10 to turn on 10 led's over the full sensor range.

output = map (sensor, 0, 1023, 0, 10);

Switch / Case: A very powerful control structure - allows running different code based on a variable value

```
switch (x)          // x is the variable that will effect what happens
{
   case 1:
     //  Do this code if x = 1.  This can also be a function call!
     break;              //  jump out of structure if this happens
   case 2:
     //  Do this code if x = 2.
     break;
   default:
   // do this if x doesn't equal any of the previous variables.  Note, this is optional but good to include
}
```