Comments:

```
// single line

/* multiple
line */
```

Syntax:

```
{}     encapsulate code
;      line ending
```

Variable Declaration:

```
#define constantName 42
const type constantName;

//forced unsigned long
const int constantName = 32767ul;

type variableName;
type variableName = value;

// can also use 1 and 0
boolean variableName = false;

// for letters only, see also "string"
// SIGNED byte, -128 to 127
char variableName = 'A'; //equivalent.
char variableName = 65;

// 0 to 255
// binary number declaration
byte variableName = B10010;

// -32,768 to 32,767
// i.e. -2^15 to (2^15) - 1)
// hexadecimal declaration shown
int variableName = 0x7B;

//0 to 65,535
//i.e. (2^16) - 1)
unsigned int variableName = 42000;
word variableName = 42000;

// -2,147,483,648 to 2,147,483,647
// i.e. -2^32 to (2^31) - 1)
long variableName = i++;

//0 to 4,294,967,295
//i.e. (2^32 - 1)
//shows returning function
unsigned long variableName = millis();

//3.4028235E+38 and as low as
//-3.4028235E+38
//(32 bit but with only 6-7 decimal
//places of precision for _both_ floats
//and doubles)
float variableName = 3.1459;
double variableName = 3.1459;
```

Arrays:

```
//arrays are 0 indexed.

// will be an array of 6 items
const int myArrayLength = 6;
type myArray[myArrayLength];

// an array 6 long, all positions full
type myArray[] = {2, 4, 8, 3, 6, 9};

// will be an array 6 long,
// positions 5 and 6 will be empty
const int myArrayLength = 6;
type myArray[myArrayLength] =  {2, 4, 6, 9};

some standard uses:
int i;
for (i = 0; i < myArrayLength; i = i + 1) {
  Serial.println(myArray[i]);
}
```

Function Declarations:

```
void myFunction(){
  //do something
}

//function that returns it’s own parameter.
//in this case types must match!
type myFunction(type myParameterName){
 type returnValue = myParameterName;
 return returnValaue;
}
```

Basics Operators

```
Comparison Operators

== (equal to)
!= (not equal to)
< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

Boolean Operators

&& (and)
|| (or)
! (not)

Bitwise Operators

& (bitwise and)
| (bitwise or)
^ (bitwise xor)
~ (bitwise not)
<< (bitshift left)
>> (bitshift right)

Compound Operators

++ (increment)
-- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)

&= (compound bitwise and)
|= (compound bitwise or)
```

Control Structures

different ways to use if...

```
if (x >= 120 || x <= 30) digitalWrite(LEDpin, HIGH);

if (x > 120 && y != 6)
digitalWrite(LEDpin, HIGH);

if (!x){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}


if (boolean test condition)
{
}
else if (other boolean test condition)
{
}
else //default to...
{
}
```

for loops

```
for (int i=startValue; i <= endValue; i++){
       // statement(s)
}

for(int x = 2; x < 100; x = x * 1.5){
println(x);
}

int x = 1;
for (int i = 0; i > -1; i = i + x){
      analogWrite(PWMpin, i);
       // switch direction at peak
      if (i = 255) x = -1; delay(10);
}
```

while and do while

```
while(boolean test condition){
  // statement(s)

  //then if you need to bail out
  if (some other test condition){
       break;
    }
}

do
{
 // statement block always runs tales once
} while (boolean test condition);
```

Digital I/O examples:

pinMode()

```
 for (byte i = 0; i <= myPinArrayLength; i ++) {
    pinMode(pinArray[i], OUTPUT);
  }

for (byte i = 0; i <= mySwtchAryLength; i ++) {
    pinMode(switchArray[i], INPUT);

    // for high impedance usage…
    //(looking for 0 not for 1)
    digitalWrite(switchArray[i], HIGH);
  }
```

digitalWrite()

```
digitalWrite(ledPin, HIGH); //true, 1
delay(1000);
digitalWrite(ledPin, LOW); //false, 0
delay(1000);
```

non blocking toggle snippet:

```
void blinkIt(int myLED, int myBlinkPeriod) {
  if
  ((myBlinkPeriod) < (currentMillis- blinkFlipTime)) {
    blinkState ? blinkState=false : blinkState=true;
    blinkFlipTime = currentMillis;
  }
    digitalWrite(myLED,blinkState);
}
```

digitalRead()

```
variable = digitalRead(inPin);
```

dependency snippet:

```
void pickLED() {
  int toggleButtonState;
  toggleButtonState = digitalRead(toggleButtonPin);
  if (toggleButtonState == HIGH) {
    currentLED = ledPinOne;
    otherLED = ledPinTwo;
  }
  else {
    currentLED = ledPinTwo;
    otherLED = ledPinOne;
  }
}
```