Shift and Mask

Bits

Consider the binary representation of a color (what the computer works with)

11111111001100000111001101111111

This is a 32 bit number which contains eight bits for each of Alpha, Red, Green, and Blue.
These values are stuffed together for several reasons:

- It saves memory space (important when you are dealing with millions of pixels)
- It guarantees the parts of a color will never be separated and lost.
- We've always done it that way
- The operating system requires it.

For human convenience, we respell numbers like this in the hexadecimal system. Four bits can be represented by a digit from 0-F.

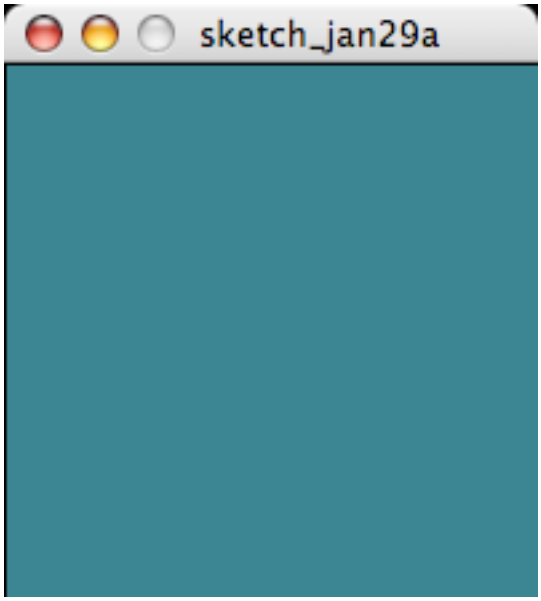
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

With this code the above number (1111 1111 0011 0000 0111 0011 0111 1111) is written FF30737F
Actually, we preface any hexadecimal number with 0x to keep from getting confused with decimal numbers, so 0xFF30737F. You can write numbers like that in Processing.
With experience you can break that up into color components

FF	full opacity
30	some red
73	more green
7F	about the same blue as green

So a sort of pale cyan.

```
void draw(){
 fill(0xFF30737F);
 rect(0,0,200,200);
}
```



RG and of course B

There are two ways to extract the individual RGB values from a color.
R=red(theColor);
G=green(theColor);
B=blue(theColor);

Easy to remember, but slow when applied to millions of pixels.

The faster way is shift and mask.

Mask first

Bit masking uses the bitwise AND operator, which is written &.
0011
&
1010
0010

The rules for logical and (&&) are applied to each individual bit. A bit is one in the result only if the corresponding bits in both operands are one. Another way to look at it is only the bits that are ones in the mask will show in the result.

A mask of 0x000000FF will reveal the last eight bits.
0xFF30737F

& 0x000000FF
0x0000007F

If you mask a color with 0x000000FF (0xFF for short), the result will be the blue.

Now shift

Numbers can have their bits shifted right or left. Shifting right is the same as dropping the last bit and sticking a 0 on the left end.
00100000
>>1
00010000

The two > symbols mean shift right, and the number is how far to shift. >>8 shifts by 8 bits. If you start with a color, >>8 will shove the blue out of the way and leave the green in the last eight bits. A mask with 0xFF will eliminate the alpha and red.

>>16 shifts enough steps to leave the red in the right end.

```
R= (theColor>>16) & 0xFF;
G= (theColor>>8) & 0xFF;
B= theColor & 0xFF;
```

To reassemble a color, start with an opaque alpha 0xFF000000
Shift red 16 left, green 8 left and add it all to blue.

```
aColor = 0xFF000000 + (R<<16) + (G<<8) + B;
```

Purists will use the bitwise OR (|) to reassemble the color instead of +;