The Imaginary Institute
www.imaginary-institute.com
2Dcourse@imaginary-institute.com

2D ANIMATION & INTERACTION COURSE
WEEK 3 QUICK REFERENCE

COLOR VARIABLES

To create a color, call `color()`, and save the result in a variable of type `color`.

```
color lightGray = color(220);
color lightBlue = color(10, 20, 30);
color transparentRed = color(255, 0, 0, 128);

// define a color in HSB
colorMode(HSB); // colors are HSB from now on
color lightRed = color(10,20,90); // new HSB color
colorMode(RGB); // always return to RGB
```

The numbers you hand to `color()` are interpreted as RGB or HSB depending on the current mode. The mode only matters when defining the color; when using a color, the mode is irrelevant. To use a color variable, hand it to any routine that's expecting a color.

```
background(lightRed);
```

To get the RGB values of a color, hand it to `red()`, `green()`, or `blue()`. To get the HSB values, hand it to `hue()`, `saturation()`, and `brightness()`. To get the transparency, hand it to `alpha()`.

Important: Do not treat colors as numbers! Don't try to multiply them or add them or do any kind of arithmetic with them. To combine colors in any way, pull them apart into components (RGB or HSB), modify the components, and then hand those new values to `color()` to create a new color.

To blend two colors, hand them to `lerpColor()` along with a float from 0 to 1. `lerpColor()` produces different gradients depending on the current color mode (RGB or HSB).

```
newColor = lerpColor(a, red, green);
```

©The Imaginary Institute, 2013. All rights reserved.

SHORTHAND MATH

`a++`	add 1 to a after the current line is completed
`a--`	subtract 1 from a after the current line is completed
`++a`	add 1 to a before the current line starts
`--a`	subtract 1 from a before the current line starts

`a += b;`	shorthand for `a = a+b;`
`a -= b;`	shorthand for `a = a-b;`
`a *= b;`	shorthand for `a = a*b;`
`a /= b;`	shorthand for `a = a/b;`

FUNCTIONS

Functions are blocks of code that can take *arguments* (or *parameters*) as inputs, and can return a value as an output.

```
// we take two floats in, and return a float
float addNumbers(float a, float b) {
  float c = a+b;       // compute sum of inputs
  return(c);           // return our work
}                      // end of function
```

To use a function, hand it values for the arguments. If the function returns a result, assign that to a variable of the same type:

```
float sumOf3And5 = addNumbers(3, 5);
```

If a function does not return a value, then don't include a `return()` statement, and instead declare the function to have the placeholder type `void`. If it takes no inputs, you still need the parentheses following the function name, but nothing goes between them:

```
void printHello() {   // no inputs and no return
  println("Hello!");  // do a little something
}                     // end of function
```

LOCAL AND GLOBAL VARIABLES

Local variables are declared inside the curly braces of a function, and they can only be referred to from within that function. *Global* variables are declared outside of the braces of any function, typically at the very start of the program. They can be used or changed anywhere in the program. Generally speaking, it's a good habit to make your variables local whenever possible, and only use globals when necessary.

It's not required, but I find it helpful to start the names of global variables with a capital letter, so it takes only a glance to distinguish them from local variables, which I always start with a lower-case letter. Note that Processing does not share this convention, and starts most of its own global variables with a lower-case letter.

BOOLEANS

Variables of type `boolean` may take on only one of two pre-defined keyword values: true and false. They can be computed by testing the relationship between two objects, typically using these tests:

<	less-than
<=	less-than or equal-to
>	grater-than
>=	greater-than or equal-to
==	equal to
!	not (applied to a boolean)

For example,

```
boolean needRaincoat = isRaining && goingOutside;
boolean got3dogs = numberOfDogs == 3; // equal?
```

boolean variables may be combined logically in two ways:

&&	and
||	or

For example,

```
boolean ripeAndBlue = colorIsBlue && berryIsRipe;
boolean ripeOrBlue = colorIsBlue || berryIsRipe;
```

Note that these operators are made of two ampersands or two vertical bars, one after the other with no spaces (using just one ampersand or vertical bar tells Processing to use different operators that are just similar enough to cause errors that can be hard to track down).

You can build up collections of tests. Here we test to see if a vegetable is corn by saying that it must be either yellow or white, and that it must be either on the cob or in a can. The parentheses make the tests unambiguous both to us and the computer.

```
boolean isCorn = (isYellow || isWhite) &&
                 (onCob || inCan);
```

Note that it's perfectly okay to insert a return and start a new line if your lines get too long. I usually try to line things up vertically so I can see at a glance what's going on.

IF STATEMENT

The simple *if statement* tests a boolean value. If it is true, the following statement (or block of code) is executed. If it is not true, that code is skipped.

```
if (berryIsRipe) eatBerry(); // do one thing

if (appleIsRipe) {
  // note that we started a block just
  // now by using an open curly brace
  removeSticker(); // take off the PLU sticker
  washApple();     // clean it off
  eatApple();      // Yum!
}   // end the block with closing curly brace
```

When using multiple statements within curly braces like this, I recommend that you indent all of the statements between the braces by one tab stop.

You can also provide an alternative block of code to execute if the test is false. Following the curly brace of the first block (or the semicolon of the first statement), include the keyword `else` and another statement or block.

```
if (appleIsRipe) {
  // go through the steps to eat an apple
} else {
  // put the apple back to let it ripen
}
```

The contents of the if statement can be any boolean variable, or a set of one or more tests that produce a boolean result:

```
if ((ballIsBouncing && ballIsRubber) ||
    (ballIsCrashing && ballIsCrystal)) {
  // bouncing rubber and crashing crystal are blue
  fill(blueBallColor);
} else {
  // all other balls are red
  fill(redBallColor);
}
```

THE KEYBOARD

Each time you press a key on the keyboard, Processing will call your function named `keyTyped()`, if you've written one. The global variable `key` is a character variable that holds the character that was just pressed. Typically you'll test this variable to respond differently to different keys. To test for a character, wrap it in single quotes.

```
void keyTyped() { // called when a key is pressed
  if (key == 'r') RedVal--; // r? decrease RedVal
  if (key == 'R') RedVal++; // R? increase RedVal
  if (key == 's') {         // s? Start a block
    // One or more lines here.
    // They're all executed if the key
    // was the letter s
  } // End of the block
```

Typically `keyTyped()` will change one or more global variables, with the intent of affecting what `draw()` will produce the next time it's called.