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

2D ANIMATION & INTERACTION COURSE
WEEK 2 QUICK REFERENCE

VARIABLES

A *variable* is a little named object that can hold a value, such as a number We can use the name of the object to refer to its value We can *assign* a variable a new value at any time Its name always refers to the value most recently assigned to it..

You can stores two different kinds of numbers, each in its own type of variable:

`int`	A whole number with no fractional part
`float`	A number that may have a fractional part

Rules for variable names:

1. Use only upper-case letters (A-Z), lower-case letters (a-z), digits (0-9), and the underscore character (_).
2. Start with a letter.
3. Start all variable names with a lower-case letter (for now).
4. Combine words into a single name in two ways:
   a) Use an underscore: `number_of_bananas`
   b) Use camel case: `numberOfBananas`

Using variables:

Assign a value to a variable using the equals sign (=) You can assign a number, or the result of some arithmetic using numbers and/or variables, using the operators +, –, *, and / Use parentheses to group operations together if there's even the slightest risk of confusion.

```
int numberOfApples;
float pricePerApple;
float totalValueOfApples;
numberOfApples = 5;
applePrice = .75;
totalValueOfApples = numberOfApples * applePrice;
```

You can save a little bit of typing by assigning a value to a variable immediately after declaring it, if you like.

```
int numberOfApples = 5;
float applePrice = .75;
float totalValueOfApples = numberOfApples * applePrice;
```

PRINTING

Use the function `println()` (short for "print line") to print something It will print the result to the output window at the bottom of the Processing environment, and will follow it with a return character so anything else you print will start on a new line.

To print text, include it in double quotation marks To print the value of a variable, just name it as an argument to `println()` To print out multiple things on a single line, join them together with the plus character (+).

```
println(priceOfGrapefruit);
println("I'm about to draw the red box");
println("numberOfApples is now "+numberOfApples);
```

PROGRAM STRUCTURE

Programs begin with a `setup()` routine, where you typically create the graphics window and then tell Processing to draw the graphics nicely.

Then you provide a `draw()` routine Processing calls this automatically when it's ready to create a new frame When `draw()` is finished, Processing copies the completed image to the graphics window It then waits until it's time for a new frame, when it calls `draw()` again Typically you'll start `draw()` with a call to `background()` to clear your drawing area to a solid color.

Except for microscopic programs, you'll always want to provide a `setup()` and a `draw()`.

```
void setup() {
  size(1000, 800);  // Make the graphics window
                    // 1000 pixels wide, 800 high
  smooth();         // Make everything look great
}

void draw() {
  background(220, 200, 22);  // light yellow
  // create variables, do stuff with them
  // draw things
}
```

COMMENTS

There are two forms of comments.

Single-line comments

Start a new comment with two slashes (no space between them!), like this: // Everything until the end of the line is ignored This is useful for one-line comments, or for adding a short note after a line of code.

```
 // Now we start drawing in red
int kiwiHeight = 50;  // draw 50-pixel-high kiwi
```

Multi-line comments

Start a new comment with the characters /* (no space between them!) Everything that follows will be ignored, up to and including the characters */ (again, no spaces between these two characters) This is useful for multi-line comments, or for temporarily “hiding” big chunks of your program without actually deleting them You cannot put one multi-line comment inside another; once the computer sees the characters */ that’s the end of all commenting until you start a new comment.

On the other hand, you can use the multi-line comments around lines that have single-line comments in them.

```
/* This is a short multi-line comment */
/* Temporarily disable the next two lines
fill(255, 0, 0);  // fill with super-bright red
noStroke();       // turn off strokes
*/
```

ANIMATION

The system variable `frameCount` tells you what frame number you're about to draw The first time the system calls `draw()`, the value of `frameCount` is 1 The next time it's 2, then 3, and so on You can use `frameCount` to create graphics that change over time.

Important: never assign a new value to `frameCount`! It's maintained for you by Processing and updated automatically.

```
void draw() {
  background(200);  // fill window with light gray
  // draw a growing ellipse centered at (400, 400)
  ellipse(400, 400, 2*frameCount, frameCount);
}
```

SKELETON PROGRAM

Here's a skeleton program that you can use as a starting point for your own programs, including your homework It starts with a typical `setup()` routine that merely creates a graphics window and calls `smooth()` All the real work is done in `draw()`, which is called automatically by Processing each time it's ready for you to provide it with a new frame.

```
void setup() {
  size(1000, 800);  // Window of 1000W x 800H
  smooth();         // Make everything look great
}

void draw() {
  background(200);  // fill window with light gray
  // draw a growing ellipse centered at (400, 400)
  ellipse(400, 400, 2*frameCount, frameCount);
}
```