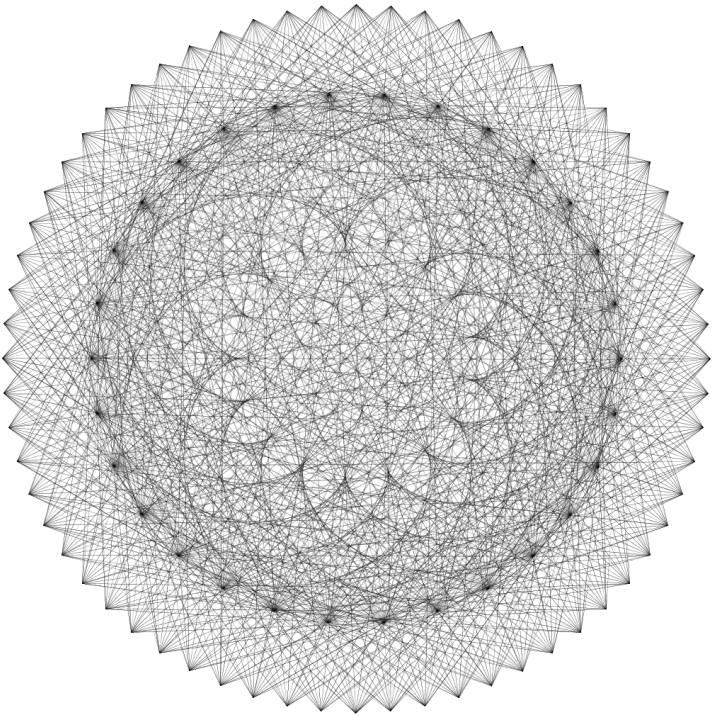


dm[2] / p5

Introduction to programming
v1 / 12.2015

Institute of Architecture and Media
Graz Institute of Technology

Constantinos Miltiadis

drafted by
Constantinos Miltiadis

studioany.com
c.miltiadis@gmail.com

University Assitant
Institute of Architecture and Media
TU Graz

December 2015

This booklet is designed so that it can be printed as an A5 booklet on A4 paper.

It written as reference material for introducing architecture students to programming. It is in no way exhaustive, or “correct” by programming standards. You are encouraged to study more material. Some links for books, tutorials and other references are supplied in the Further Resources section.
One thing to remember is that coding can only be mastered by practice. Use this booklet as technical reference to develop your own ideas.

*this book is bound to have mistakes. If you spot one -or more- inform us.

Cover image: Async circles, 2015.

Contents

Ones and zeros

Computers work in Binary, that is ones and zeros, which is the fundamental computing element. Although it is not necessary to do

Lets calculate an 8-bit binary number

binary: 0 1 1 0 1 0 1 1

Starting from right to left, we translate the numbers to powers of two, with the power being the position of the digit. Therefore the rightmost digit (1) will translate to 1*2^0 = 1. Accordingly the 3rd digit (0) will be 0*2^2 = 0

powers: 7 6 5 4 3 2 1 0
base: 2 2 2 2 2 2 2 2

result:

```
      2^0 *1  = 1
+     2^1 *1  = 2
+     2^2 *0  = 0
+     2^3 *1  = 8
+     2^4 *0  = 0
+     2^5 *1  = 32
+     2^6 *1  = 64
+     2^7 *0  = 0
-------------------
=     107
```

What is the value of the IAM logo?

What is programming?

-What is programming?
-Programming refers to writing code in a programming language in order to create software.

-What is a programming language?
-A programming language is a human readable logical language that is used to write instructions for a computer to execute.

-And then?
-Then the code is translated by a compiler to something that the computer actually understands.

-Why programming?
-There is no clear answer to that. Programming doesn't help you do something specific, but on the contrary it allows you to do anything, by formulating what you need in a programming language.

What is Processing?

Processing -or p5 for short- is a programming language and IDE developed by Ben Fry and Casey Reas at the MIT Media Lab in 2001. It was developed as a tool to introduce designers and non-programmers in general, into programming.

It comes with its own friendly IDE (integrated development environment) for writing and running code.

Processing is free and open-source, and it basically is a large library built on top of Java.

It can be downloaded from: https://processing.org

To have a look at projects developed with processing, and see some of its capabilities go to: https://processing.org/exhibition/

Why Processing?

Some of the reasons are:

- Accessible, free and open-source
- Easy to install
- Cross-platform
- Friendly and easy to use
- The output of the code is visual.
- Easy for programming graphics
- Makes it very easy to export stand alone applications
- Has a huge user community; the odds are for whatever problem or question you have, there is a solution posted online already.
- It offers a large amount of free resources and libraries online
- Learning programming through Processing offers a solid ground for development of your skills further in any other programming environment.
- You can make it talk to other programs or hardware

Installing Processing

To install processing you only need to download and unzip a package. It automatically creates a Processing folder in your Documents folder, where it saves your sketches (programs) and libraries.

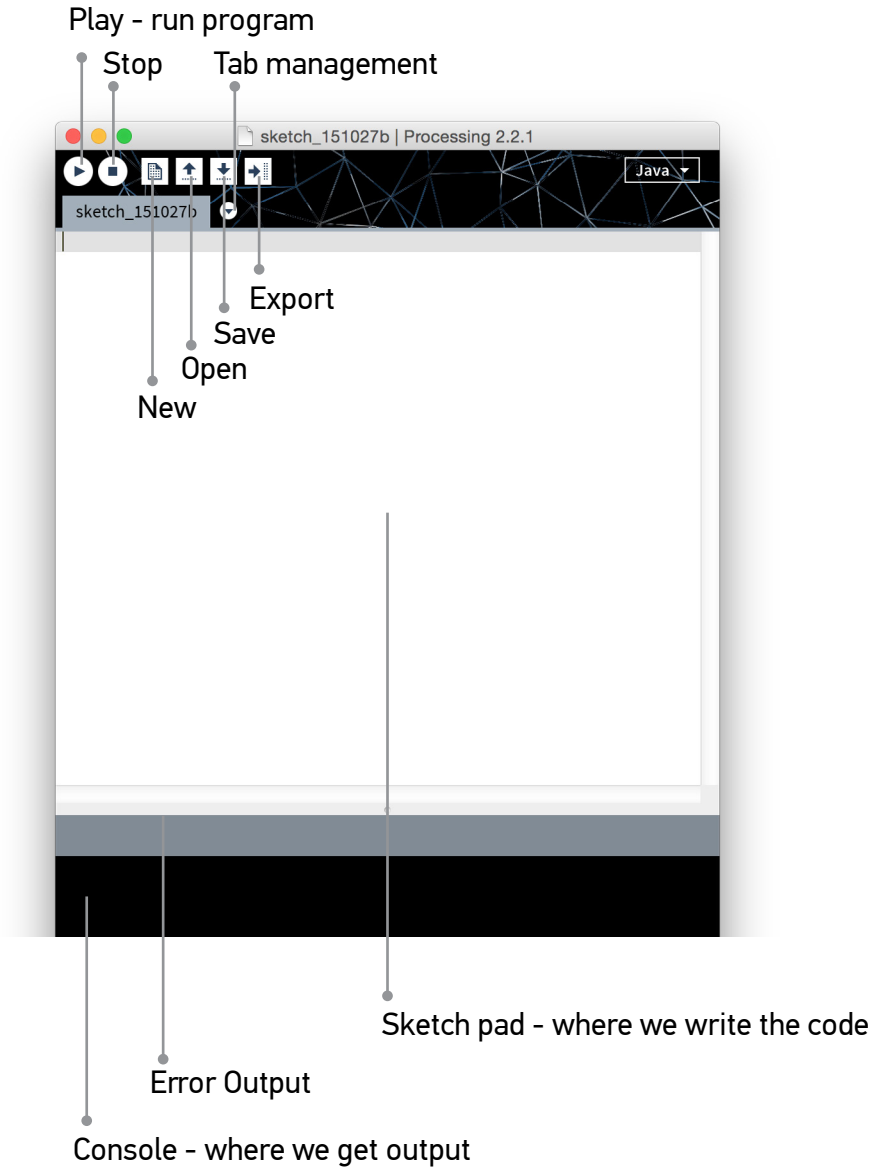
Installing on Windows

1. Go to https://processing.org
2. Click on Download
3. Select if you want to make a donation and then Download
4. Select your platform and wait for the file to download
5. Unzip
6. Copy the contents to a folder of your choice (preferably C:/Develop/Processing)

Installing on OS X:

1. Go to https://processing.org
2. Click on Download
3. Select if you want to make a donation and then Download
4. Select your platform and wait for the file to download
5. The downloaded file is an application, you can go ahead and move it to your Applications folder

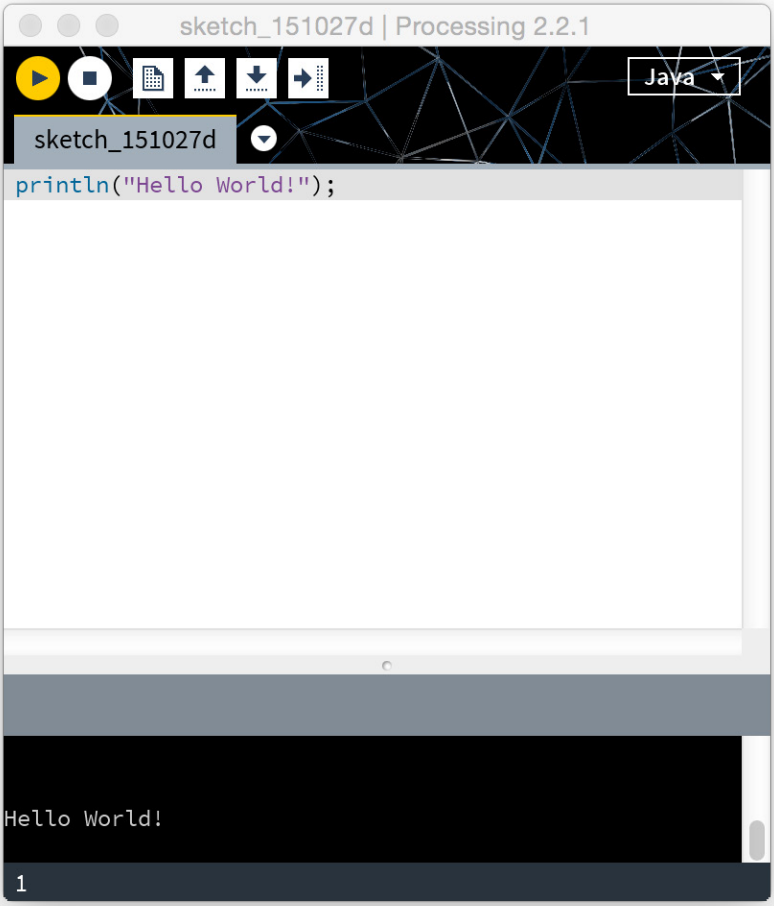
Anatomy of the Processing IDE (v2)



Hello world!

Our very first program. Write the following line and press play!

```
println(“Hello World!”);
```



Canvas

To draw shapes, we have to get accustomed to the canvas. One thing to note, is that the canvas uses Cartesian coordinates, and its origin is the left top corner. So right is X+ and down is Y+. The units are pixels.

Setting the size of the canvas is done in the setup function:

```
void setup(){
        size(500,600);
}
```

We just set the canvas width to be 500 pixels and height 600 pixels.

A lot of times, we need to refer to the canvas size, which is also the maximum drawing space we have. We can do that easily as Processing saves these values as “width” and “height”, and therefore we don’t need to hard-code them.

```
//for example to draw a diagonal line, from the origin, to the
bottom right corner.
line(0,0,width,height);
//therefore eliminating the need to write the actual values, in
this case 500 and 600
```

We can also chose a renderer for our canvas, by passing an extra argument to the size function. By only giving width and height values, we are using the default P2D (processing 2D) renderer, but we can also use P3D and OPENGL by writing:

```
void setup(){
        size(300,300,P3D);
}
```

Colors

Now lets draw some color. Processing allows for different color modes, but we are going to use grayscale and RGB which are the easieast.

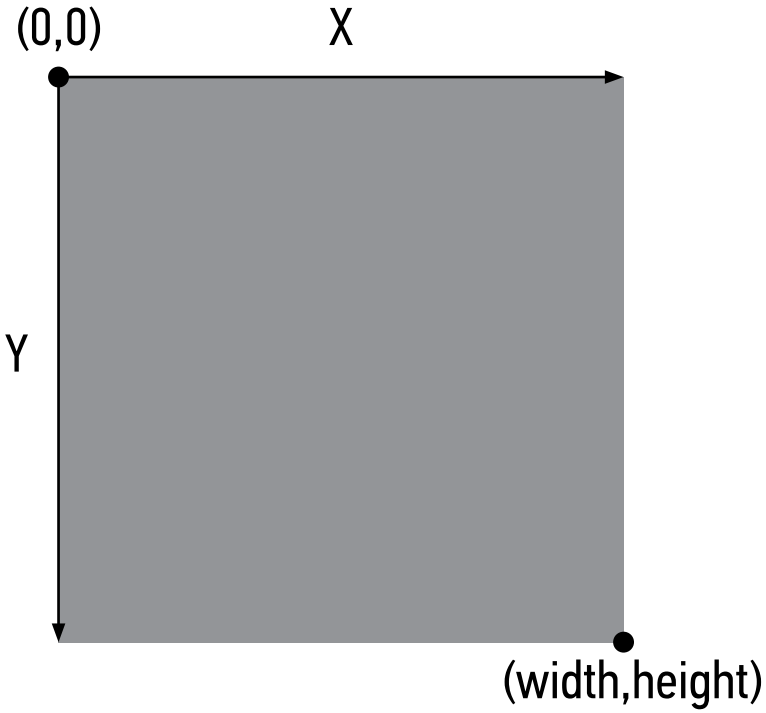
To test colors lets play around with the background function. To do that we use the background function which accepts 1 or 3 parameters, of the range 0-255.

If we only pass 1 parameter it controls the brightness.
If we pass 3 parameters, they are respectively the amount of Red, Green and Blue. Try the following code inside the setup( ) function.

```
background(0); //colors the background black
background(125);//middle grey
background(255); // white

background(255,0,0);// red
background(255,255,255);// white
```

Coordinate System



Shapes

Now we can go ahead and play around with some basic shapes. As we mentioned before they follow the Cartesian coordinates of the canvas.

```
point(x,y);

line(x1,y1,x2,y2);

ellipse(x,y,diameter1,diameter2);

rect(x,y,width,height);

triangle(x1,y1,x2,y2,x3,y3);
```

You can set the line color of these shapes by using stroke and noStroke before you draw your shapes:

```
stroke(0); // for black
stroke(255,0,0); // for red
noStroke(); //to not draw the outline
```

Similarly, you can set the fill color by fill and noFill:

```
fill(255); // for white
fill(0,0,255); // for blue
noFill();
```

You can also add transparency (alpha) to the above, but passing another argument, again from 0-255 to the fill and stroke functions:

```
fill(125,100); //semi transparent grey
fill(255,0,0,150); //semi transparent red
```

The Usual Suspects

Comments

First and foremost are comments. While writing a program, its a good habit to also write comments -sentences, to explain what we are doing, in case we want to return to our code at a later point. We can write comments -text that the compiler ignores- after "//" or multi-line enclosed between "/*" and "*/". Processing colors comments gray.

The semicolon

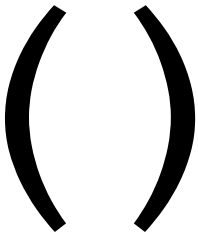
This is called a semicolon, and its used to signify the end of each line of code or statement.
The semicolon is the most used symbol in programming, omitting it the most common typing error.

Curly brackets

Curly brackets are used to enclose a chunk of code. For example the code of a function or a inside a statement:

```
float add(int numA, int numB){ //a function that adds 2 numbers
        return numA+numB;
}
```

Parenthesis



Parentheses, are most notably used to enclose arguments of a function. for example:

```
add(5,6);
//or calling a function with no parameters
noStroke();
```

Square brackets

These are called square brackets, and are used for indexes.

```
myArray[4];
```

Also note that indexing always starts from 0. Therefore to access the first object in an array we need to ask for [0].

Dot operator

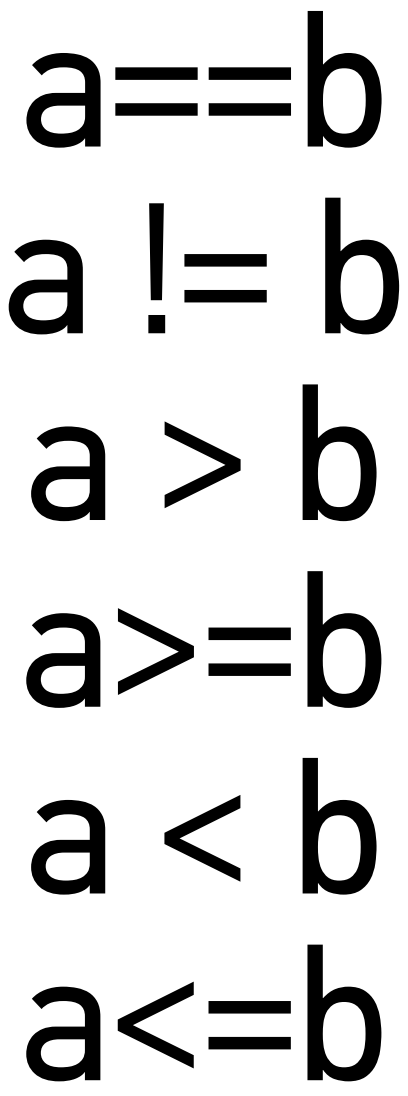
The dot [operator] `.`, is used to access methods or properties inside an object.

```
object.property;
object.function();
```

Mathematical operators

These are the mathematical operations available (addition, subtraction, multiplication, division and modulo).

Logical operators



The most common logical operators used to compare 2 values (a and b). They return a value of true or false (equals, not equal, greater, greater or equal, smaller, smaller or equal).

Basic Program Structure

Processing has 2 built-in functions that we use to structure our programs: setup and draw.

Setup and whatever we place inside it, gets executed only once in the beginning of the program.
On the other hand whatever is inside draw, is repeated multiple times per second.

We do that like so:

```
void setup(){

}

void draw(){

}
```

So the rule of thumb in processing is:
Inside setup we put anything that we want to execute just once, and inside draw, anything that we want to repeat.

These 4 lines are the basis of all the programs we are going to write in the future.

To understand the differences try the short program below

```
//sketch 1
void setup(){
        background(random(255));
}
```

And now the following

```
//sketch 2
void draw(){
        background(random(255));
}
```

By the way random(255) asks for a random number between 0-255. That number we use in the background function, to color the background. (0 - black, 255 - white).

Depending

Variables

Variables allow us to store values in memory.

You can think of a variable as an egg cup. Its a holder for an egg.
Any egg.

You can place an egg in an egg cup, then remove it and put another; as long as it is an egg.

A variable has a name, with which we refer to it, and a type of the things it can hold.

Primitive Data Types

Similarly programming does not deal with eggs, but with data. It allows us to have different types of egg cups for different types of eggs that we will need.

The [most common] data types are:

boolean : a true or false value
int : an integer number //eg 3 5 -100 etc
float : a floating point number //eg 3.1429 -53.002
char : a character // a b c d

some other types that you might see around are:

byte : 8-bit value from -128 to 127
short : integer from -32,768 to 32,767
long : integer in the range from -2^{63} to $2^{63}-1$
double : floating point number of higher presicion than a float

and another special one is

String : that holds text

String is not a primitive type, but a collection of char values. Its called an object -don't worry for that now- and that's why it starts from an uppercase letter instead of a lowercase one.

Declaring & assigning variables

When we want to create a variable, we have to tell the computer, so that it saves a space in the memory for our variable.

Say that I want to make a variable to hold my favorite number. My favorite number is 8, which is an integer, so probably the variable type I am looking for is an int. Furthermore I will need a name for my variable, so that I can recall my favorite number.

Declaring that goes like this:

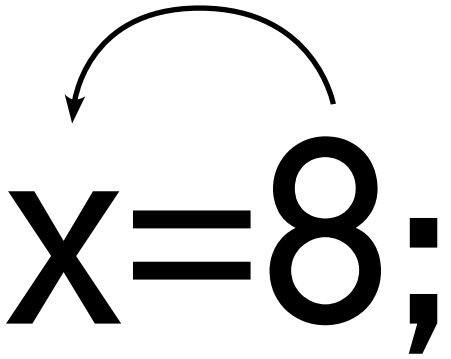
```
int myFavoriteNumber;
```

By writing so, the computer understands that we need a variable (egg cup) of type integer (int) and that we will refer to it later with the name "myFavoriteNumber".

Notes:

- Notice that I wrote "myFavoriteNumber" and not "my favorite number". This is because we cannot have spaces between variable names. They have to be a single word.
- Variable names cannot start with numbers or symbols. Only letters or an underscore "_myVariable".
- Also that it starts from a lower case letter, and every first letter of the next word starts with an upper case. This is called camel casing, and its a convention for java.

Assigning values



Assigning a value to a variable, is one way and always goes from right to left. The value on the Right will be assigned to the variable on the Left. Of course the value on the right (8 in this case) can be replaced by another variable.

In general to declare and assign a variable we write:

```
type name = value;
```

Likewise here are some examples:

```
int myNumber =8;
float myValue = -5.8;
boolean myBool = false;
char myChar = 'c';
String myString = "hello";
```

After declaring a variable, we just call it by name:

```
println(myNumber);
myNumber = -100;
```

The IF Conditional

The IF structure is a conditional statement used to perform actions controlled by boolean condition.

In peudocode it looks like

```
if ( CONDITION) {
        DO ACTION
}else if ( CONDITION 2){
        DO ACTION 2
}else {
        DO ACTION 3
}
```

and in an example

```
int myAge = 20;

if (myAge>=18){
        println("I am an adult");
}else if (myAge>0) {
        println("I am not an adult");
}else {
        println("Negative age entered");
}
```

It starts with an IF, then optionally followed by any number of ELSE-IF, and in can end with an ELSE.
As soon as one if or else if condition if met, the program exits the structure.
The ELSE case is optional and is activated only if none of the other cases (if, else-ifs) were met.

The previous is different from:

```
if (myAge>=18){
        println("I am an adult");
}
if (myAge>=0) {
        println(I am not an adult);
}
```

In this case, if "myAge" is positive, both "i am" and "I am not" will be activated.

We can also stack multiple if structures inside each other.

```
int myAge = 20;

if (myAge>=18){
        println("I am an adult");
}else if (myAge>=0) {
        println("I am not an adult");

        if (myAge<6){
                println("I should be in kindergarden");
        }else if (myAge<12){
                println("I should be in primary school");
        }else {
                println("I should be in high-school");
        }
}else {
        println("Negative age entered");
}
```

Variable scope & Indentation

Scope is a term which refers to the space where a variable is visible or valid. A general rule is that a variable lives only inside the curly brackets in which it was declared.

```
int myNumber =8;

void setup(){
        println(myNumber);

        int myAge = 20;
        println(myAge);
}

void draw(){
        println (myNumber);

        String myAddress = “Inffeldgasse 10”;
        println(myAddress);

        println(myAge) ; //! EXCEPTION ERROR
}
```

“myNumber” is not inside of any curly brackets. Therefore it is valid and can be called from anywhere in our code. Variables like this, we call Global.

“myAge” on the other hand was declared inside the setup function. It is therefore valid (or alive) only in there. Its therefore called a Local variable.

In case we try to execute the above code, Processing will complain that “myAge cannot be resolved to a variable” - meaning that it cannot find any variable with that name.

Indentation is a coding convention that is very helpful to organize our code, and for understanding which variables are valid and where. It refers to the convention of indenting each new block of code to the right by one tab.
Although programming languages and compilers do not require that, it is very helpful in order to read and understand a piece of code and its structure.
Variable scope can easily be shown by an indented piece of code. Inside processing double-click an opening bracket to see where it closes. Whatever is declared inside it, is only alive there.

In Processing you can Auto-Indent with Control+T

```
int myAge=20;

void setup(){

    boolean isAdult;

    if (myAge>=18){

        isAdult =true;

    }else{

        isAdult= false;
        boolean isInSchool;
        if (myAge>=6) {
            isInSchool=true;
        }else{
            isInSchool = false;
        }

    } //--isInSchool not valid after this closing bracket
} //--isAdult not valid after this closing bracket
```

Functions

Writing functions is another very important aspect of programming. It helps us in 2 ways: first break done our code to smaller parts which we can control, organize and reuse; and second make our code modular.

So lets say we are working with right angle triangles and we want to calculate the area and the hypotenuse of a right angle.

```
float sideA = 3;
float sideB = 4;
void setup(){
        //calculate the area
        float area = sideA*sideB/2f;
        println("Area:"+ area);

        //calculate the hypotenuse (a^2+b^2=c^2)
        float hSquared = sq(sideA) + sq(sideB);
        float hypotenuse = sqrt (hSquared);
        println("Hypotenuse:"+ hypotenuse);
}
```

Everything seems fine up to here, but what if we want to compute 2 triangles at a time? We will need to write these formulas again... Or write a function that we can reuse.

To define a function we need a Type, Name and Arguments.
Lets take the Area problem. It takes the length of 2 sides as input and then computes and prints the area. The 2 input numbers are called ARGUMENTS, and in this case should be floating point numbers. Because it just prints the result the Type will be VOID (empty). For name lets use "printTriangleArea".

```
//function definition
void printTriangleArea(float s1, float s2){
        float result = s1*s2/2f;
        println(result);
}
```

Now we just call it as many times we want and it will print the result for us. To call a function we just write its name followed with parentheses which include the input to the function. In this case:

```
//Calling our function
printTriangleArea(3,4);  //3 is taken as s1 and 4 as s2
printTriangleArea(6,8);
printTriangleArea(125,55);
```

Now lets say that we want to get the result instead of printing it. In this case the function has to RETURN a value back to where it was called. It is therefore not a void, but has a return type. Since the area of a triangle is a floating point number, the function will be a float. So the above function definition is transformed to:

```
//function definition
float getriangleArea(float s1, float s2){
        float result = s1*s2/2f;
        return result;
}
```

And now, we can actually get a number for the area, in case we want to use it further in our calculations.

```
//calling the function
float area1 = getArea(3,4); //area1 is 6
float area2 = getArea(6,8);  //area 2 is 24
float area3 = getArea(125,55); //area3 is 3437.5
```

The same for calculating the hypotenuse:

```
//definition
float getHypotenuse(float s1, float s2){
        float result = sqrt( sq(s1) + sq(s2) );
        return result;
}
```

And calling the function - from inside setup () for example

```
float hypotenuse = getHypotenuse(125,55); //=136.365
```

Arrays

While in many cases, having variables to store values is sufficient, in some occasions, we might need to store multiple values in an effective way. With as much as we know, we would go about it like so:

```
String student1 = “Anna”;
String student2 = “George”;
String student3 = “Martin”;
String student5 = “Kristina”;
```

It might seem OK for now, but this way is neither practical nor effective. Also what happens when we have 100 values…?

In these cases we use Arrays. As we mentioned before that we can think of variables a egg cups, we can think of arrays as egg cartons.

Continuing from the last example say we want to save the names and ages of 4 students. We need 2 types of variables: Strings for storing names, and integers for the age.

```
//one way to do it is like this
String [ ] names = {“Anna”, “George”, “Martin”, “Kristina”} ;
int [ ] ages = {18,20,19,22};
```

We can now get the amount of values stored in a

To access the values inside each array, we just need to call the variable name followed by the index of the value we need. An index is a numerical position, written in square brackets.

NOTE: Indexing always starts from 0. So the first position is 0 the second is 1 etc.

To print for example the name and age of the student in the first position in the lists we can do the following:

```
println(“Student 1 : “+ names[0] + “ age ” + ages[0]);
```

We can also declare an array and fill it later:

```
//declare the array
String [] names;

void setup(){
        //set its size
        names = new String [4];
        //and set values
        names[0] = “Anna”;
        names[1] = “George”;
        names[2]= “Martin”;
        names[3] = “Kristina”;
}
```

Loops: FOR

Loops are structures that execute instructions repeatedly.

Say for example you want to repeat the following line of code 100 times:

```
point(random(width), random(height));
```

it wouldn't make sense to hard-code it 100 times. You can rather use a loop:

```
for (int i=0;i<100;i++){
        point(random(width), random(height));
}
```

The above is the most common - FOR Loop. The For loop uses an iterator, in this case "i" which functions as a counter.
Lets examine how it works and break down the part inside the parenthesis.

```
//set a new iterator for the loop, and set it to the starting point
int i=0;
//set an end point
i<100;
//set the iterator step
i=i+1 or i++
```

So from i=0 until i reaches and including 99, repeat what is inside the curly brackets.
We can also use a for loop for counting:

```
//this will print all numbers from 0 to 99
for (int i=0;i<100;i++){
        println(i);
}
```

But also Loops are also very useful for working with Arrays. Say we have the following arrays:

```
String [ ] names = {“Anna”, “George”, “Martin”, “Kristina”} ;
int [ ] ages = {18,20,19,22};
```

To print all of the entries, instead of doing:

```
println(names[0]);
println(names[1]);
//.....and so on
```

We can use a For loop. Arrays have a very helpful property which is the amount of values they hold, which is called length, and can be accessed as:

```
int arrayLength =  names.length; // will print 4
```

We can use the length of the array as the end point of our loop iterator.

```
for (int i=0;i<names.length;i++){
        print(names[i]);
}
```

And because we know both the “names” and “ages” arrays have the same length (4) we can use either.

```
for (int i=0;i<names.length;i++){
        print(“Student in position:”+ i);
        print(“ name :”+ names[i]);
        println(“ age :”+ ages[i]);
}
```

We can do all sorts of stuff with arrays and loops, such as fill a huge array with random numbers. and find the sum, and other properties of it.

```
//create a new array with size 482
float [ ] myArray= new float [482];

//fill the array with random values
for (int i=0;i<myArray.length;i++){
        myArray[i] = random(100);
}

//get the sum of all these values
float sum =0;
for (int i=0;i<myArray.length;i++){
        sum+=myArray[i];
}
//print the sum of all these numbers
println("The sum of all "+myArray.length+" values is "+ sum);
```

Calculate the average (sum/number of elements)

```
println("The average number is: "+sum/myArray.length);
```

Find the largest number in the array. Create a new "max" variable, set it initially to something that is definitely not the maximum - say 0. Iterate through all values and check if each one is larger than "max". If it is, set "max" equal to the current value.

```
float max =0;
for (int i=0;i<myArray.length;i++){
        if (max<myArray[i]) {
                max = myArray[i];
        }
}
println("The maximum value in the array is:"+ max);
```

Mouse Events

Processing offers us access to input provided by the mouse or keyboard.

Regarding the mouse, Processing has 2 variables used to get the mouse position. These are mouseX and mouseY, and can be accessed from anywhere (they are Global variables).

```
void draw(){
        ellipse(mouseX, mouseY, 10, 10);
}
```

We can also get mouse events - when a mouse is clicked, moved or dragged. These are functions that get activated whenever any of the above occurs.

```
void mouseMoved(){
        println("The mouse is moving");
}
void mousePressed(){
        println("Mouse click");
}
void mouseDragged(){
        println("Mouse is dragged");
}
void mouseReleased(){
        println("Click released");
}
```

NOTE that you have to have a void draw function declared in your program, otherwise it will just run once and then terminate - therefore you will not get any mouse or keyboard actions.

Keyboard events

Likewise we can get keyboard input, through functions that get activated whenever something happens on the keyboard.

```
void keyPressed(){
        println(key);
}
void keyReleased(){
        println(key);
}
```

We can get the value of the key that was pressed/released by the local variable “key”. The “key” can be accessed as both a character and an equivalent integer of the ASCII table*.
To check if a particular key was pressed we can use either consecutive statements or a switch structure. The latter might be more economical for this case.

```
//getting a key with an If structure
        if (key=='a'){
                println(“a was pressed”);
        }else if (key == ‘v’ || key =='V'){
                println(“V or v was pressed);
        }else{
                println(“another key was pressed”);
        }
//or with a Switch structure
        switch (key){
                case ‘a’:
                        println(“a was pressed”);
                break;
                default: println(“Another key was pressed”);
                break;
        }
```

*The ASCII table: http://www.asciitable.com

Shorthand notations

It is usually the case in programming that we need to add something to a value of a variable, divide it, or simply increment it (add 1). For these simple operations, there are shorthand notations that are useful and widely used.
Here are the most common full and shorthand notations:

```
//increment (actually post increment)*
x=x+1;
x++;

//decrement
x=x-1;
x--;

//add
x=x+3;
x+=3;

//multiply
x=x*5;
x*=5;

//subtract
x=x-10;
x-=10;

//divide
x=x/600;
x/=600;
```

* https://en.wikipedia.org/wiki/Increment_and_decrement_operators

Processing mathematical functions

Processing offers a lot of mathematical functions. All can be found in the processing reference online. Here are some of the most commonly used.

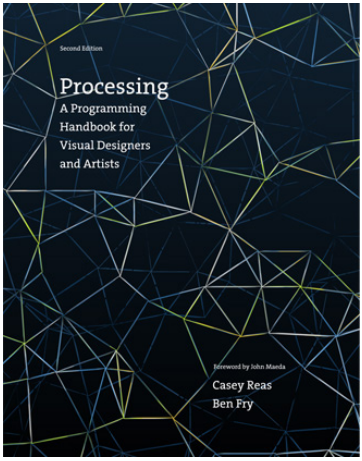
Numerical

```
sq(2); // square of number / will give 4
sqrt(9);// square root of number / will give 3
pow(2,3) ;// number to power / gives 2^3 = 8
abs(-3);// absolute value
ceil(2.1);// rounding up / gives 3
floor(2.9);//round down / gives 2
round(2.1);// round / gives 2
max(2 , 9 ); // maximum of 2 numbers
min(4, 1) ;// minimum of 2 numbers
```

Trigonometrical

```
PI // Processing Constant / gives 3.14
//also TWO_PI, HALF_PI, QUARTER_PI, TAU
sin (PI); // sine of angle (in radians)
cos(TWO_PI); //cosine of angle in radians
tan(radians); // tangent
//also their inverse functions asin, acos, atan, atan2
//angle conversions
degrees(radians); // returns  the angle in degrees
radians(degrees);// returns the angle in radians
```

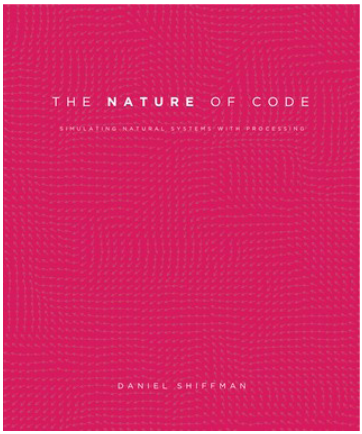
Further Resources



Processing: a programming handbook for visual designers and artists
by [the creators of processing]
Casey Reas & Ben Fry

at IAM library, Inffeld library

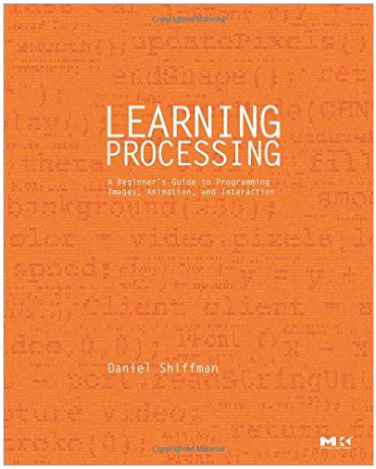
A great and slow paced book, that includes a vast array of examples and also interviews with artists using Processing.



The Nature of Code
by Daniel Shiffman
[processing foundation]

download from
http://natureofcode.com

Focused mostly on Vectors, mathematics and algorithms. Goes from vector math to generative algorithms, cellural automata and more.



Learning processing: a beginner's guide to programming images, animation, and interaction

by Daniel Shiffman
[processing foundation]

online at: http://proquest.techbus.safaribooksonline.de/9780080920061

hard copy at Inffeld library

Links

Processing programming reference: https://processing.org/reference/

Video tutorials

Jose Sanchez Plethora Project Tutorials
http://www.plethora-project.com/education/

Stanford University - Programming Methodology Class
Prof. Mehran Sahami
https://see.stanford.edu/Course/CS106A

Video lectures

Karsten Schmidt - The tower of Babel / Eyeo 2014
https://vimeo.com/70457659

Michael Hansmeyer - Building unimaginable shapes
https://www.ted.com/talks/michael_hansmeyer_building_unimaginable_shapes?language=en

Computerphile Youtube Channel
https://www.youtube.com/user/Computerphile

Audiovisual Projects

Ryoji Ikeda - Test Patterns / Transfinite / Data Patterns
https://www.youtube.com/watch?v=XwjlYpJCBgk
https://www.youtube.com/watch?v=k3J4d4RbeWc

Amon Tobin - ISAM
https://www.youtube.com/watch?v=0zMI-qbmIPk

Readings

Daniel Davis, Jane Burry, Mark Burry - The Flexibility of Logic Programming: Parametrically Regenerating the Sagrada Família
http://www.danieldavis.com/the-flexibility-of-logic-programming-parametrically-regenerating-the-sagrada-familia/

Karsten Schmidt - Jacobs Ladder
https://medium.com/@thi.ng/the-jacob-s-ladder-of-coding-4b12477a26c1#.z11w24vni

General glossary

Programing language - A human readable language used to communicate or pass instructions to a machine. Programming languages are used to write programs (software) that control the computer (hardware). A program written in a specific language is then compiled (translated) to another language that the computer understands and can execute.
Common programming languages are C, C++, Java, Ruby, Swift, Python, C# etc.

IDE - Integrated development environment. A program that provides the environment and tools to develop programs.

Low level / High level programming languages: A relative term that denotes the “distance” of a programming language to the machine that is controlling. C and C++ are low level languages where Java is a high level one. That means that Java has a lot more intermediate layers between the software and the hardware, which makes it slower in comparison to the former.

OOP, Object Oriented Programming: A programming paradigm that allows for the creation of Objects. Objects are data structures that include data and behavior (methods).

Java: Java is a strongly typed object-oriented programming language, that is designed to have minimal platform dependencies, and therefore to a large extend is cross-platform. As of 2007 Java is free and open-source.

JVM, Java Virtual Machine: Is a program that handles the execution of Java programs. It is essentially the key part that makes Java cross-platform.

Open-source: Refers to software that has its code (source code) publicly

accessible. Open-source software also has a certain distribution license. Further reading: Open Source Initiative, Free Software Foundation, GNU General Public License, MIT License, Richard Stallman, Creative Commons, Lawrence Lessig.

Cross-platform: Refers to applications that can run on all types of computer architectures (the same software, for example, can be executed on Windows, OS X and Linux).

Platform dependent / Platform dependencies: Refers to applications or parts of the code that are specific per computer architecture (specific version for each operating system).

Compiler: A program that transforms code written in a specific programming language to instructions that a computer can understand and execute.

Case sensitive: Refers to programming languages that are sensitive with character casing e.g. number ≠ Number ≠ NUMBER

Naming

UPPER_CASE_CHARACTERS

lower_case_characters

camelCasing

Naming Conventions

Generally agreed upon rules for writing code, so that other people can read and understand it easily.

Some of the common Java conventions are:
using camel casing
using lower case for the first character of variable and function names
using an upper case for Class names

Camel Casing: A naming convention for writing compound words in one so that they can be easily readable. In Java we use camel casing which starts from a lower case character. Therefore if we want to name a variable that describes my number we would write it as

Processing glossary

Sketch: A program written in Processing. The are stored in the Sketchbook, in a folder with the same name. They have the extention .PDE

Sketchbook: The folder that contains our processing programs, or Sketches. By default is located at Documents/Processing and its autogenerated when running processing for the first time.

Tab: A tab of code used to split and organize our program. Although Processing stores the contents of a tab to a different PDE file, in practice they are just like parts of a larger text, and as far as code semantics go, they are not distinguishable entities.

PApplet: Although not visible from inside processing, PApplet is the processing framework that we write code in Processing.

Library: A piece of code that can be referenced to our code and provides further functionality. For example, a popular Processing library is PeasyCam which offers a 3D camera to Processing.
The default location for user installed libraries for Processing is: Documents/Processing/Libraries and its automatically generated.

Installing new libraries:
The easy way to install a library is to go to Tools/Add Tool and then search for the library you are looking for.
The traditional way, is to download the library you want from the Processing library Index and Unzip it to the Libraries folder.

Processing library list
https://processing.org/reference/libraries/