Elements of Programming

Please note: a lot of this text is taken from *Processing: A Programming Handbook for Visual Designers and Artists, by Casey Reas and Ben Fry.*

Functions

Functions allow you to draw shapes, set colors, calculate numbers, and to execute many other types of actions. A function's name is usually a lowercase word followed by a set of parentheses. In Processing, if there is a second word in the function the first letter of the second word is capitalized, like the strokeWeight() function.

The comma-separated elements between the parentheses are called **parameters**, and they affect the way the function works. Some functions have no parameters and others have many.

example of a preset function in Processing:

```
size(400, 400);
```

You can also write your own functions in Processing — we'll learn about that in a few weeks.

Variables

"In software, data is stored as numbers and characters. Examples of digital data include a photograph of a friend stored on your hard drive, a song downloaded from the Internet, and a news article loaded through a web browser. Less obvious is the data continually created and exchanged between computers and other devices. For example, computers are continually receiving data from the mouse and keyboard. When writing a program, you might create a data element to save the location of a shape, to store a color for later use, or to continuously measure changes in cursor position."

The first time a variable is written, it must be declared with a statement expressing its datatype.

Global and Local Variables

Global variables are introduced at the beginning of a program and are accessible within every code block. Local variables are introduced within a certain code block and can only be used within that code block. It is a good practice to only use variables where you really need them. If you only really need a variable within a certain if statement or function, make your variable local; it saves computer space, and you'll get less errors and confusion.

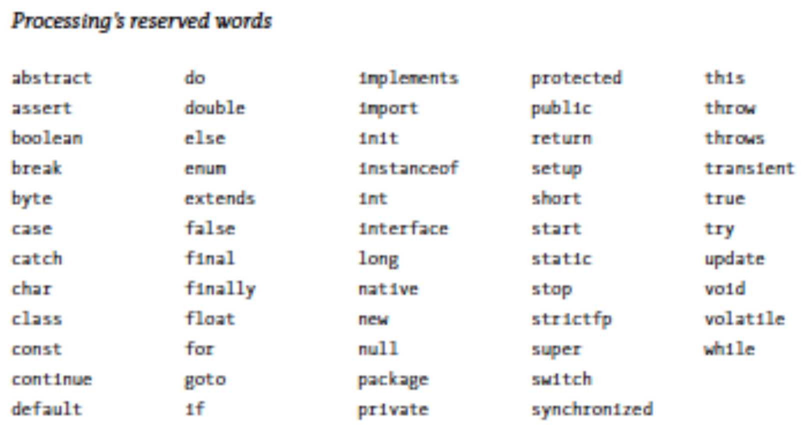
Tips

- Different variables can't have the same name.
- Name variables who words or single letters — not half words that only make sense to you.
- Reserved words: Some words are essential to the Processing language, and their use is restricted to their intended use. For example, the int data type is an integral component of the language. It can be used only to declare a variable of that type and cannot be used as the name for a variable. The following program produces an error:

```
float int = 50; // ERROR! Unexpected token: float
```

Elements of Programming

Here is a list of reserved words that you can't use as variable names:



Processing's reserved words

abstract	do	implements	protected	this
assert	double	import	public	throw
boolean	else	init	return	throws
break	enum	instanceof	setup	transient
byte	extends	int	short	true
case	false	interface	start	try
catch	final	long	static	update
char	finally	native	stop	void
class	float	new	strictfp	volatile
const	for	null	super	while
continue	goto	package	switch	
default	if	private	synchronized	

- Variable Types:

In Processing:

- int
    - Datatype for integers, numbers without a decimal point.

```
int a;
OR
int a = 23;
```

- float
    - Data type for floating-point numbers, e.g. numbers that have a decimal point.

```
float a;
OR
float a = 1.5387;
```

- String
    - Data type for a sequence of characters. A String is defined between double quotation marks.

```
String mywords;
OR
String mywords = "This is a sentence";
```

- boolean
    - Datatype for the Boolean values true and false. It is common to use boolean values with control statements to determine the flow of a program.

```
boolean a = false;
OR
boolean a;
```

- mouseX & mouse Y (system variables)
    - The system variable mouseX and mouseY always contain the current horizontal and vertical coordinates of the mouse.

```
line(mouseX, 20, mouseX, 80);
```

Also see:

- PImage, byte, char, color, double, float, long, pmouseX, pmouseY, PFont, PShape

Code Blocks

Code inside a set of braces {} is called a block. The two major code blocks structuring most Processing programs are setup() and draw(). A typical Processing program will hold most code in these two blocks. The structure looks like this:

```
void setup(){
        // code setting up the size and background color goes here
}
void draw(){
        // code that runs continuously goes here
}
```

Conditionals

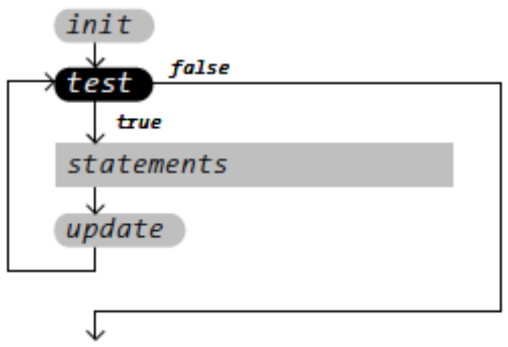
Conditional statements (if/else) in combination with relational expressions (<,>, <=, >=, ==, !=, etc.) and logical operators (&&, ||, !) help control the flow of a program. Building on the work of the logicians Leibniz and Boole, modern
computers use logical operations such as AND, OR, and NOT to determine which lines of code are run and which are not.

```
if (test) {
        statements
}

OR

if (test) {
        statements
} else {
        statements
}

OR

if (test) {
        statements
} else if{
        statements
} else{
        statements
}
```

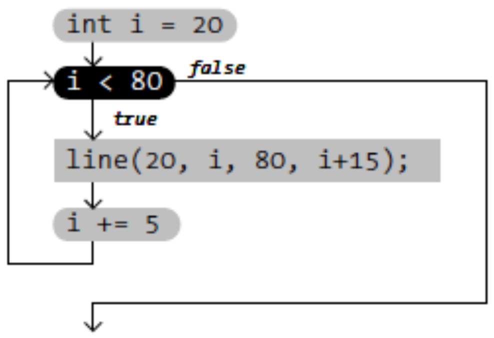
Iteration

For Loops

Iterative structures are used to compact lengthy lines of repetitive code. Decreasing the length of the code can make programs easier to manage and can also help to reduce errors.



```
for (init; test; update) {
            statements
}
for (int i = 20; i < 80; i += 5) {
            line(20, i, 80, i+15);
}
```



Tips

Formatting

Be sure to indent statements inside of code blocks. When you introduce your global variables at the top of your document, skip a line. Aim for clarity and readability. Make sure all your braces and parentheses are closed, and you have semi-colons at the end of your statements. command T on a Mac will auto-format your code.

println()

Use print() and println() to test variables and expressions in the command line. Use it anytime you need to have a better sense of what your code is doing.

Commenting

/* Comment everything. Say what each line of code is doing, or what you think it's doing. If in doubt, comment. If you don't know what a line of code is doing, make a comment above it about what you think it's doing, and why it confuses you. If you think one line of code is the reason your program isn't working, make a comment. Make comments at the top of your code saying what you were aiming for conceptually, and where you actually got technically. Make a comment when you understand perfectly what your code is doing — it's so satisfying to say so. The practice of commenting code helps the programming student to become conscious of what code is doing and logical processes by articulating it in natural language, and it helps other people looking at your code to understand what it's doing quickly. */

Make Mistakes

Make lots of mistakes. This is how you will learn. When you don't understand what your code is doing, email it to your prof (me) or someone who does.