# Resource: Variables, Declarations & Assignment Statements

Three interrelated programming concepts are variables, declarations and assignment statements.

## *Variables*

The concept of a *variable* is a powerful programming idea. It's called a variable because – now pay attention – it varies. When you see it used in a program, the variable is often written like this

```
r = 255;
```

(`r` is the variable and the whole thing is the *assignment* statement). You get the idea from such usage that a variable is like an unknown in algebra. But, it's nothing like that. (More about the difference below.) What a variable is is a *container* for a value. So, `r` in the example will "contain" the value `255`. It's a number, of course, but whatever a variable contains is called is *value*.

I used a simple letter for my variable in this example, but variables in Processing (and most languages) can be any string of letters and numbers (and underscore _) as long as they start with a letter, and they are case sensitive. Examples of other variables are:

```
x
area
abc
ABC
really_long_variables_are_annoying_to_type
```

## *Declarations*

The other thing about variables is that they have a *data type*, which is the sort of information they can contain. So, I write,

```
int r = 255;
```

which says that `r` is a variable that will contain an integer, that is, a whole number. Another choice for a variable type would be `float`, which is the programming term for "decimal" numbers, that is, numbers of the form `3.14159`.

The text

```
int g = 0;
```

is called a *declaration statement*, because it declares that the variable will contain an integer; and in this case, the integer starts out as the value 0. I want to make one more declaration,

```
int b = 0;
```

which, as you now know, declares b to be container for integers, starting by containing 0. Also, it is possible to declare a variable without giving it an initial value, as in

```
int temp;
```

There are other data types besides int and float that need to be declared in Processing. Find them at http://processing.org/reference/ in the index headed **Data**. It's also is a good place to get further explanation.

## Assignment

To make a variable vary, we *assign* a new value to it. This is done, unsurprisingly, in an *assignment statement* such as

```
b = r;                    // Set b to have the value r has
```

which says, the contents of r (recall, it's a container) are assigned to or replace the contents of b. The flow of information is from right to left, from r to b, in this case. Some programmers say, "b gets r", meaning b is replaced by the value of r. Notice that we are discussing information here, so nothing happens to the value of r. It is unchanged. It's still there. We simply get a copy of that same value in b.

## Examples

Here are some declarations

```
int area, long_side, short_side;              // declaring 3 integer variables
float radius, force, mass, acceleration;      // declaring 4 float variables
float pi = 3.1415962;                         // declare a float initialized to π
…
area = pi * (radius * radius);                // compute area of circle
force = mass * acceleration;                  // compute a physics law
long_side = area/short_side;                  // compute a fact about rectangles
```

We like to have the declarations at the top of the program (or the start of a function definition, as explained later).

## Assignment (continued)

An important fact about variables is that declaring them only says *how* they are used. They don't start out with any value unless initialized, and they don't get any value until you assign one. So in the examples in the last section the assignment statements, such as

```
area =  pi * (radius * radius);
```

are meaningless unless the variables first are assigned values. pi is, but how are we supposed to square the radius (that is, multiply it times itself) unless I know its value (that is, what float it contains)? I assumed that variables were assigned values in the program where the ellipsis (…) is given.

We use the equal sign (=) to indicate assignment, but as stated above, it doesn't mean "equals;" it means "transfer the value from the right hand side to the variable on the left hand side." This difference is extremely important, so let's say more about it.

## *Computing Is Not Algebra*

The assignment symbol is not the "equality" of algebra, and variables are not the unknowns of algebra. Here's how we know.

Suppose I write

```
int x = 0;
```

telling the Processing language that I want the variable x to have whole numbers as values, and that the first whole number it has will be 0. And then later in my program I want x to be 1 larger than it is presently. In programming, I write

```
x = x + 1;
```

which means that x is to be assigned the value it already has, increased by 1. That is, its value becomes one larger as a result of the assignment statement. Other examples might be assignments in a program that simulates basketball in which

```
score = score + 3;
shot_clock = shot_clock – 1;
```

is written to record the result of a 3-point basket, or elapse of a second on the shot clock. As before, the assignment means add three to score's current value and make the result the value of score. That's how assignment works.

But in algebra, the equal sign means that the values on both sides are the same. So the way you know computing isn't algebra is that

```
x = x + 1
```

is meaningless in algebra. No number equals itself plus one. That is a contradiction. So, when you see assignment statements in programming, realize that they mean to transfer the value from the right side to the left side.  They don't mean equality.