Section 5: Functions

Introduction: A function is a subroutine that can be referenced by name. In lecture, we likened this to the chorus of a song: in-between verses you sing or play the same chorus without having to write out the same thing all over again. In a computer program, a function is a set of instructions that execute when you call a function. After the function is done, it will return to the point in the program that it was called from. Optionally, a function may pass a return value back to its caller.

The main benefits of using functions are (1) decomposition of larger problems and (2) code reuse from different places in your program. Code reuse can be *identical,* or *similar* if you use parameters.

Function Definition: Because you are creating a subroutine from scratch, you need to tell Processing (1) what it does and (2) how you plan to use it. The first line of a function definition is the function interface and tells Processing how you will use the function: `return_type function_name(type1 param1, …)`

- `return_type` is the data type of the return value. If there's no return value, use `void` here instead.
- `function_name` is the name of your function. The naming rules follow those of variables.
- The parameter list is contained within the parentheses. This is a comma-separated list of variable types and names. This will declare a set of variables that you can use *only* within your function. If you are not using any parameters, you may leave the parentheses empty (but they are still required!).

After the function interface, the function body (what it does) is found within curly braces (`{}`). Note that a function definition does not require a semicolon! It is good practice to indent your function body so it is easy to tell which statements are part of that function.

Examples:

```
void owl(int s) {
   // s is short for "scale"
   strokeWeight(s/2);
   ellipse(7*s, 7*s, 9*s, 12*s);
   ellipse(5*s, 5*s, 4*s, 4*s);
   ellipse(9*s, 5*s, 4*s, 4*s);
   ellipse(5*s, 5*s, 1*s, 1*s);
   ellipse(9*s, 5*s, 1*s, 1*s);
   triangle(6*s,7*s, 8*s,7*s, 7*s,9*s);
}
```

```
int sum(int x, int y) {
   return x+y;
}
```

Calling Functions: Functions are called by name, followed by an argument list in parentheses. The argument list corresponds to the parameter list from the function definition and provides the *values* that the parameters get initialized to when your function executes.

- Argument values get assigned to parameters *in order*
- Argument values can be written explicitly or taken from variables.

Examples:

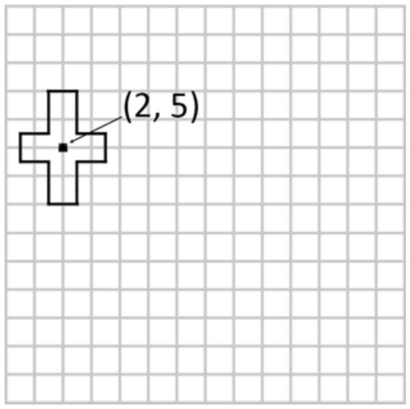
```
int z;
z = sum(5,10); // run sum() with x=5, y=10; store returned value in z
owl(z);         // draw an owl with s=15
```

Exercises:

1) The function `mystery` is defined below. What value is returned by `mystery(1)`? What are *all* of the possible return values of this function?

```
int mystery(int x) {
   return min(10, max(0, 2*x));
}
```

2) We've written a function that draws a cross. The plot below is the result of calling `cross(2, 5, 3, 4)`. On the same plot, draw the result of calling `cross(10, 10, 8, 6)`.



3) Write a Processing function below that computes and returns the average of 3 given numbers.
Hint: this function should take three `float`s as arguments.

4) Write a Processing function below that, when given two coordinates `(x1,y1)` and `(x2,y2)`, draws a line segment between the coordinates, places a point at the midpoint, and returns the length of the line segment.

Hint 1: The commands **`sq`**`()` and **`sqrt`**`()` compute the square and square root of a number, respectively.
Hint 2: What should the data type of the return value be?

5) Go to the course website and get started on the homework titled "Animal Functions." [*individual*]