# Vocabulary

**Function**: A section of code that has one specific purpose. It can be used over and over again. To call (or use) a function you simply write the name of the function followed by an open and closed parenthesis. Any information that you need to give to the function is passed as an argument (see below). The actual function code is written somewhere else, in a library or in a different section of the code and is probably a lot longer than the length of the function name. It's a lot easier to write the function name (also known as "calling the function") each time you want to use this code instead of writing it all out again.

**Passing an argument**: This is the primary way that information is passed to a function. To pass an argument you will write a number, character or word inside of the parenthesis that comes after a function name. Some functions don't need to have arguments passed to them, some won't work if you don't pass them arguments, others need multiple arguments to be passed to them (separated by commas).

**Library**: A library is another way to reuse code. They often use an object to group functions together, which is handy when you have functions with the same name that do different things, such as println(). The Sparki object has a println() function that prints to the LCD, but there's also a built in Serial.println() that lets us print back to the computer's serial port.

**Compiler**: Takes the code we write and turns it into a small and efficient package that the processor can run.

**Datatype**: To be the most efficient, we need to tell the compiler what kind of data we're working with. For example, the char, or character, type uses 1 byte per character stored, even a space takes 1 byte. An int, or integer, uses 2 bytes no matter what value you store, but can store any whole number from -32,768 to 32,767 in that 2 bytes, where -32768 would take 6 bytes in a char!

**Statement**: A generic term for part of the code that does something but isn't a function or variable.

## Code from examples explained

```
1 #include <sparki.h>
```

This is a command to include the file named Sparki.h in the code that is uploaded to Sparki. Sparki.h is a header file that contain a whole bunch of functions and other code which makes Sparki work. This is where you can find the actual code inside the functions you called above.

```
1 //include Sparki library
```

This is a comment. Anything with two // doesn't effect your code. It's just there to explain things to another human being who is reading your code. You can write silly things in here, take credit for your work or use two // at the beginning of a line of code to take the code out of the program Sparki runs but leave it visible to another programmer or yourself. You can also use /* at the beginning of a bunch of lines you wish to comment and */ at the end of the lines you're commenting out. Code can get confusing. Comments are your best friend when you're trying to keep track of what's going on or share your code with others.

```
void setup(){
}
```
This is a little section (or function!) of code that happens exactly once before `loop()` takes over and starts happening over and over and over and over. It's empty right now, but anything you want to happen exactly once should go inside this function's curly braces { }.

```
void loop()
{
 //code goes here
}
```
This is the primary function in Sparki's code. Whatever is inside this function happens over and over until Sparki's batteries die or it gets turned off. Let's go over the parts of this code so you understand what each part does.

**void**– if this function were returning any information, the datatype of that information would be listed here. The word "`void`" indicates that `loop()` doesn't return any information.

`loop`– this is the name of the function. You can name your own functions basically anything as long as there is no space in the name. Some function names, like `loop` are "reserved" though. This means that the word is special to the program or system and it is already used so you can't just use it for whatever you like. However, `loop` and `setup` cannot be renamed and also must exist or the code won't compile.

( )– if there was any information (arguments) that was being passed to the loop function the information (or variable) type would be inside of these parenthesis along with the name of the variable. Loop does not get any information passed to it so these parenthesis are empty. *Note, parenthesis are also used for different purposes in code, function arguments is one of their purposes.

{ }– all the code that happens (or executes) when the `loop()` function is called needs to be inside of these curly braces. *Note, curly brackets are also used for different purposes in code.

```
sparki.something
```
Whenever you see `sparki` followed by a dot, that's the sparki object. When we `#include <sparki.h>` the code included creates this object for us and we then use it to access the sparki library.

```
if()
while()
```
These are examples of flow control statements. `if()` lets you make decisions, you place conditions inside its parentheses and follow it with code that only gets run if the condition is true. If you want more than one line of code to run, place it inside curly brackets {} just like a function. `while()` lets you repeat code, you place the conditions inside the parentheses and follow it with code that gets repeated. Also use {} if you have multiple lines. When you see `while(1)` that mean to repeat forever because in code anything not zero is true.