Variables, #Define & storage

In addition to doing things rapidly, one area where computers can easily best your average human, is memory. Computers can trivially recall things quickly and precisely; as long as they were first asked to remember it.

Let's say you are building an automatic oven, which can measure the temperature of the food it is cooking. The user inputs the desired 'cooked' temperature. The oven needs to periodically check the temperature and compare it to the 'cooked' temperature. Hence, the controller needs to remember the 'cooked' temperature.

Programming languages use ***Variables*** for remembering things. They are just symbols which stand in for any arbitrary value, similar to variables in Algebra. The value can be changed, or vary, hence the name 'variable'. Variables names can most anything, though in the language we are using there are a few rules :

- Can consist of letters a-z, upper or lowercase, numbers 0-9, and the underscore ("_")
- Cannot begin with a number
- Cannot be certain ***reserved words***[1]
- Generally no maximum length, though some systems may cap at 64-256 characters.

Assigning a value to a variable is easy. If we wanted the variable 'foo' to hold the value 17, we could just write:

```
foo = 17;
```

Once assigned, anywhere we use `foo`, we will be actually using the value 17. At least until we set `foo` to some other value; again that's the variable part.

In C/C++/Arduino, variables can store most anything — within a certain range, or ***type***, which they are initially specified to be using. In programmer speak, this means that the variables are ***strongly typed***, and must be pre-defined.

What this means in practice is that at some point *before* we could use `foo = 17;`, we would need to have told the compiler that we intended to use a variable called `foo` *and* that `foo` was going to hold a number, specifically an integer. This is done by simply inserting the following line at the start of our code:

```
int foo;
```

This is called ***declaring*** a variable. If you put this line in the `setup()` area, then the variable will only be available for use in the `setup()` area, likewise, if you put it in the `loop()` area, it will only be available there. If you put it above and outside both areas, it will be available everywhere; variables declared this way are said to be ***global***. In many situations global variable use should be limited or avoided entirely. However, in the Arduino IDE they are used heavily. You will soon learn about blocks, and in fact variables can be declared inside any block, and are available only inside that block. This is referred to as their ***scope***.

There are other frequently used types of variables. `int` only allows for storing whole numbers. To store, say `3.12`, you might use `double`. To store the letter 'a' you would use `char` (short for character), which can hold any letter, number, or symbol.

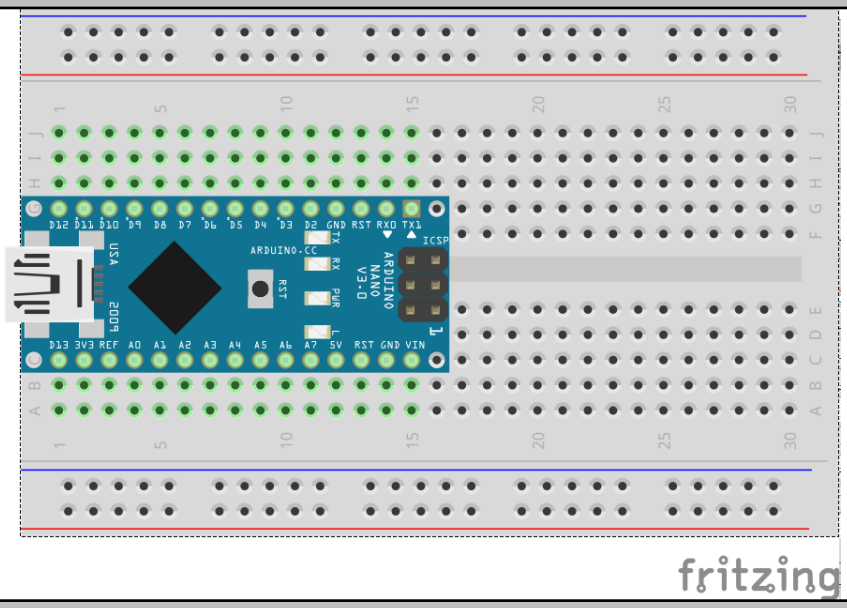
These sort of variables are often called **scalar** variables, since they represent a single value. There are other variables which we will examine later which can represent multiple values. The term scalar comes from linear algebra, where a scalar contrasts with a matrix or vector. This exposes programming and Computer Science's root in mathematics, but don't worry, knowledge of linear algebra is not required for most programming!

Some programming languages do not require that you pre-define or specify an initial type for a variable you will use. These are called ***dynamically typed*** languages.

In addition to these normal variables, there's one other special construct available in the C/C++/Arduino environment. The first step of compiling involves invoking what's called the ***preprocessor***, which looks for lines in the program that start with the #, or pound character. The word that comes after the # is a command. One command, `#define` (pronounced: pound define) is a simple substitution system that the compiler does before anything else. It will search your code for the line `#define XXX YYY`, then it will search the rest of the program and look for occurrences of XXX, and replace any that it finds with YYY . This is not really a variable, as it cannot be changed, and it cannot be referenced as such. Also, confusingly, these lines do not require a semicolon, as they are handled by a different process, and technically done before the compiling begins in earnest. We will make minimal use of `#define`, but many packages and example programs you find will use them, so it is important to understand how they work.

[1]Reserved words are words that are used to define other instructions to the compiler. A list is provided in the appendix.

Experiment: It Varies.

Components	Wiring Diagram
✓Computer ✓Arduino IDE	

Connection Instructions	
No connections required, we will be using the internal LED for this example.	
Sketch(es)	notConstant.ino ***(This is not in the examples. You will duplicate and rename previous completed project to create this sketch)***

Analysis Questions

Does your code seem a bit redundant once you have completed the task?
How did you determine what the appropriate delay times would be?

Programming Tasks

In this experiment we are going to modify the previous project.

In the window of your previous project, Select All, then use the Copy function to copy it.

Start a new project in the Arduino Application. In the window which opens Select All again and hit the delete key.

Now that you have a blank window for the new project, Paste in the contents of your old project.

Replace all the `delay()` millisecond values with variables. You will probably need at least 2, one for the short or ‘dot’ delay, one for the long, or ‘dash’ delay, and possibly a third for the delay between each letter’s flash sequence.

Once that’s working, replace the value you assigned to one of your variables to with a #DEFINE