# More on Functions

Functions are a great tool for helping code be

- understandable
- maintainable

Eg: In lab2, all the intelligence about what levels of HDL are healthy, and when LDL is too high, are put into functions.

- might be used in multiple places in the program
- if it has to be changed, just change it once
- "information hiding" - put all the medical expertise in one place

To check what the interpretation of an ldl level is, we just call the function.

If medical wisdom changes about ldl healthy range, we just change it in one location.

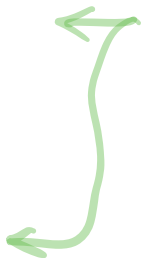
# Scope

In general, in C++ :

```
{

    int x;

}
```



if these are the closest matching { , } that enclose the declaration of variable x then x is only defined within the indicated code block.

```
int specialFunction(int numFarms, int numCities);

int main()
{
    int x=0, y=1;
    int z = specialFunction(x, y);
    return 0;
}

int specialFunction (int numFarms, int numCities)
{
    int w = z;
    w = (x+y)/(numFarms + numCities);
    return w;
}
```

// nonsense code fragment, no modularity

```
cityProduction = f(x, y);
farmAnimals = Y(temp);
cityPollution = g(w, z);
farmTime = farmTime / farmAnimals
farmMod = farmTime + farmAnimals;
cityMax = h(i, z);
R = relate(cityMax, farmMod);
```

// nonsense code, more organized

```
cityProduction = f(x, y);
cityMax = h(i, z);
cityPollution = g(w, z);
. . .

farmMod = farmTime + farmAnimals;
farmAnimals = Y(temp);
farmTime = farmMode / farmAnimals
```

```
R = relate(cityMax, farmMod);
```

```
// nonsense code, more modular

c = doCityStuff(x, y, i, z, w);
f = doFarmStuff(farmTime, farmAnimals, temp);
R = relate(c, f);


int doCityStuff(int x, int y, int i, int z, int w)
{
    cityProduction = f(x, y);
    cityMax = h(i, z);
    cityPollution = g(w, z);
}


int doFarmStuff(int farmTime, int farmAnimals, int Y)
{
  farmMod = farmTime + farmAnimals;
  farmAnimals = Y(temp);
  farmTime =
}
```

# Modularity

Goal: a program that consists of distinct parts, each part being clearly discernable in either

- what it accomplishes
- what "knowledge" it encapsulates
- what data it looks after

The programmer/engineer(s) come to agreement on the **name, parameters and task** that the part will have

The **inner workings** of the part can be hidden from the other parts ... and therefore programmers working on other parts don't have to know or deal with **how** this part does what it does, but get to "use" it (call it) to get it to do what it does.

Lab 3 :