Return Value

Get to know about how to return a value from a function. Also, discover recursion which is a very important concept in programming.

We'll cover the following

- Recursion

Let's consider another example, one where we want our function to return a value. Let's write a function to compute Fibonacci numbers (Wikipedia). Fibonacci numbers are defined as:

```
\begin{equation} F_{n} = F_{n-1} + F_{n-2} \end{equation}
```

where $F_0 = 0$ and $F_1 = 1$.

```
#include <stdio.h>

int Fibonacci(int n) {
    if (n==0) return 0;
    else if (n==1) return 1;
    else return Fibonacci(n-1) + Fibonacci(n-2);
}

int main() {
    int n = 10;
    int Fn = Fibonacci(n);
    printf("Fibonacci(%d) = %d\n", n, Fn);
    return 0;
}
```

Here we define the return type as `int`, as we want our function to return an integer. We define one input argument, called `n`, which is also an `int` type. Then in the body of the function, we do our calculations.

Unlike in some languages such as Python, Matlab, and R, in C, functions can only return a single value. There is a way to achieve the same result however, which is to return a **pointer** to a complex data type such as an array, or a structure. We will

talk about complex data types in the next section.

Recursion

Note in the Fibonacci example above, that in the body of our function, if the value of the input argument `n` is not `0` or `1`, then the function ends up calling itself (on line 6 of the code listing). When a function calls itself, this is called **recursion**, or a **recursive function call**. Recursion allows for very compact code, and for intuitive definitions. You may see recursion used in mathematical functions, and also in algorithms like sorting and searching. The cost of recursion is that sometimes the overhead involved in the computer repeatedly calling functions over and over again, can be costly, but this really depends on the nature of the algorithm. For Fibonacci numbers, recursion is OK for small `n` but once `n` becomes large, it is really slow. As an exercise, you could try to re-code the Fibonacci function using a loop instead of recursion. Another thing to try is **memoization** (look it up, I didn't mis-spell it).

With functions, we always have to be careful. Whether we're calling a function recursively, or passing arguments into the function, there is a chance things might go wrong. How? Find out in the next lesson.