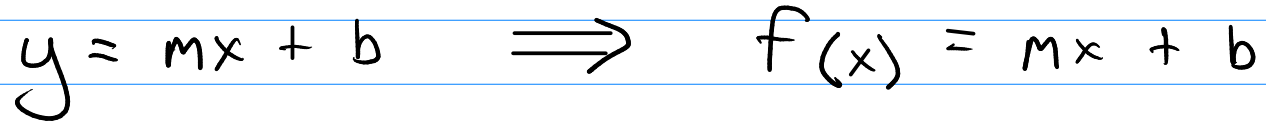
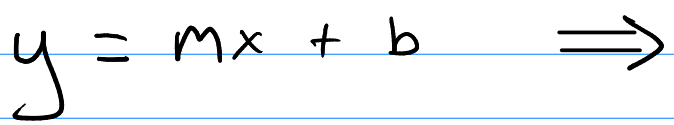
Methods in Java are very similar to functions in algebra! The most basic function in algebra is the good old polynomial function: y = mx + b, where m is the slope, and b is a constant. In this context, y is the function that depends on the value of x, assuming m and b are always constant (let's not go too far into Calculus).



A method in Java is very similar to algebraic functions in that a method in Java will perform some data manipulation, and possibly output a value. I say "possibly" because methods are SIMILAR to algebraic functions, but not synonymous.




```
double y = calcLinearEquation(slope, x, constant);
```

A method's signature typically goes in this syntax:

```
<access level> <extra keywords> <return type> <method name>(<parameters>) {
    // Code goes in this block
}
```

For example,

```
<access level> <extra keywords> <return type> <method name>  (<parameters>)
public static void main (String[] args) {
    System.out.println("Hello World!");
}
```

Note: You can have multiple keywords in a method's signature!

The most common keyword is static.

Others include:
final
synchronized
native
etc.

Important things to know about methods:
1) The execution of a method ALWAYS ends when it reaches the return statement.
2) If a method's signature has a return type, the return statement MUST return a value of that type.
3) The access level of a method (or class field) determines what other classes may access that method (or class field). For example, public gives everything and everyone access to that member; private gives member access only to other members in the same class.
4) The number of parameters in a method signature should ALWAYS be the least possible. I.E. If you wanted to write a method that calculates exponents, you ONLY need two parameters: the base, and the power; nothing more.
5) The method's name should be as descriptive as possible according to its purpose, but should also not be too lengthy. For example,

fetchDataFromInputFile()
would be too long,

in()
would be too short,

but
fileInput()
would be just right.

Always use camel-casing!