# Writing a Swap Function?

Students often ask whether it is possible to write a Ruby method that swaps two variables. The tl;dr is to do this:

x, y = 3, 4

y, x = x, y

The longer answer is that you cannot write a swap **method** in Ruby. There is no way to get access to the name of the arguments outside a function, so we can't rebind them. For instance if we could write

x, y = 3, 4

swap(x, y)

# Now x = 4, y = 3

that would mean that swap somehow knows that the names of the arguments passed in are x and y. But that's not how methods work. The arguments are evaluated before the swap method is called, and then references to the objects (not the names of the variables) are passed into the method.

Take another example that helps explain why object references are passed to functions: what is swap(3, 4) supposed to do? Here we're passing in **immediate** values; they aren't even stored in variables.

Other languages allow you to write swap variables. Here's an example from C:

void swap (int\* xp, int\* yp) {

int z = \*yp;

\*yp = \*xp;

\*xp = z;

}

void main () {

int x = 3;

int y = 4;

swap(&x, &y);

// Now x == 4 and y == 3.

}

In main, we ask C to create space for two ints, and to store 3 and 4 in them. The expression x returns the value stored in the memory allocated for x. The expression &x returns a **pointer**, which is the memory address which the value for x lives in.

The swap function takes in two **pointers** (that's what int\* means); it does **not** take two int **values**. The code of swap pulls the value stored at yp (\*yp means to pull the value out of the memory address yp) and stores this in memory for the variable z. Next, we pull out the value at memory address xp (\*xp) and copy it into memory address yp (\*yp = ...). Lastly, we copy the value of z into memory address xp.

Because swap changes the memory that underlies the x and y variables in main, the swapping is visible outside swap.

Swapping can be accomplished other ways in other languages. For instance, in Lisp, swap could be written as a macro. Macros in Lisp don't evaluate their arguments. The raw, unevaluated Lisp expression is passed into the macro, which can then manipulate it