Splat Operator

Splat Operator

There are few different ways to use the splat (*) operator in Ruby. Let's explore each of them so we can add them to our programming tool belt.

Using splat to accept additional arguments

Ruby methods are pretty strict in that we must pass in the exact number of arguments that a method expects. If we pass in too many, we will receive an error.

Building upon the code above, if we want our method to have the ability to accept at least two arguments with potentially more, we can add a splat parameter. The additional arguments will be gathered into an array for us to use as we see fit:

```
def method(arg_1, arg_2, *other_args)
    p arg_1          # "a"
    p arg_2          # "b"
    p other_args     # ["c", "d", "e"]
end

method("a", "b", "c", "d", "e")
```

If we pass in exactly two arguments, then `other_args` will be an empty array:

```
def method(arg_1, arg_2, *other_args)
    p arg_1          # "a"
    p arg_2          # "b"
    p other_args     # []
end

method("a", "b")
```

As a best practice, we should use splat at the end of the parameter list to avoid confusion.

Using splat to decompose an array

We can also use splat to decompose or unpack elements of an array. Let's say we had an array containing some elements, but we wanted each individual element to become an argument:

```
def greet(first_name, last_name)
    p "Hey " + first_name + ", your last name is " + last_name
end

names = ["Grace", "Hopper"]
greet(*names)    # => "Hey Grace, your last name is Hopper"
```

When using splat to unpack an array, you can imagine that the `*` will remove the brackets ([]) that enclose the array. This leaves us with a simple comma separated list, perfect for passing in arguments. If you imagine `*` as removing the brackets around an array, we can figure out some other creative ways to use this tool:

```
arr_1 = ["a", "b"]
arr_2 = ["d", "e"]
arr_3 = [ *arr_1, "c", *arr_2 ]
p arr_3 # => ["a", "b", "c", "d", "e"]
```

Using splat to decompose a hash

We can use a double splat (**) to perform a similar unpacking of a hash's key-value pairs. Double splat will only work with hashes where the keys are symbols:

```
old_hash = { a: 1, b: 2 }
new_hash = { **old_hash, c: 3 }
p new_hash # => {:a=>1, :b=>2, :c=>3}
```