Ruby - Methods - Arguments

In the previous challenge, we learned to use methods to abstract similar computations into logical chunks of code that otherwise would be difficult to manage. Methods, in a way, behave like a *black box*. The programmer works mainly on 1) inputs, 2) expected output, and 3) how it works. We do not have to worry about method internals. In this set of tutorials, we will focus on understanding the three aspects described above.

The ability to pass arguments allows complexity to be hidden from the programmer. We have already seen straightforward cases of passing several values to methods as variables, but there is much more to Ruby's methods.

Consider a case where a method is invoked from different portions of code with a variation in only one of the arguments. All other arguments remain constant. In such cases, it is useful to assign default values to the variables. It allows us to avoid passing a value for every argument, decreasing the chance of error.

For example,

```
def prefix(s, len=1)
  s[0,len]
end

> prefix("Ruby", 3) # => "Rub"
> prefix("Ruby")    # => "R"
```

In this challenge, your task is to determine what the *take* method does. Study the examples below, then implement the method.

```
> take([1,2,3], 1)
[2, 3]
> take([1,2,3], 2)
[3]
> take([1,2,3])
[2, 3]
```

Note

The method can be invoked as `name('Foolan', 'Barik')` or, without the parentheses, as `name 'Foolan', 'Barik'`. The latter convention can be confusing and is not recommended.