Ruby - Methods - Arguments

In our previous challenge, we learned about methods and how they help us to abstract similar computations into logical chunks of code that otherwise would be very clumsy and difficult to manage. Methods, in a way, also behave like a *black box* where the programmer should ideally be concerned only with a basic description of the box - what it does, what is it input and what is the expected output, without having to worry about how it does it. This allows us to focus more on the functionality and correctness of program instead of implementation details. In these set of tutorials, we will make the same black box assumptions and focus on building a better understanding of the three aspects described above.

The ability to pass arguments is of critical importance as it determines the complexity and variability of output that we can generate. We have already seen straight forward cases of passing several values as variables to our methods, but however, there's much more to Ruby's methods. Let's examine it with practical scenarios.

Consider a case where you have a method that is invoked from different portions of your code with some variation in one of the arguments. However, you still need to pass a value for the remaining arguments which mostly remain constant through these calls. In such cases, the ability to assign default values to your arguments becomes increasingly useful as it helps you to avoid passing a value for all arguments and decrease chances of making errors. It is quite analogous to an *on-demand* behavior where you pass an argument only when you need it to affect your output, otherwise let the default action go on.

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 figure out what `take` method does using the examples below and implement the method. It should help you understand how to build on implementation through the expected functionality.

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

Note

One can invoke method by simply writing `name('Foolan', 'Barik')` or without the parentheses like `name 'Foolan', 'Barik'`, although the latter convention can be confusing and hence is not recommended.