Blocks

Higher order functions are one of the key components of functional programming.

A higher order function is a tool that takes other functions as parameters or returns them as a result.

Blocks are nameless methods that can be passed to another method as a parameter.
Passing a block to a method is a great way of data abstraction.

Blocks can either be defined with a keyword `do ... end` or curly braces `{ ... }`.

Example:

a). Passing a block to a method that takes no parameter

CODE

```
def call_block
   puts "Start of method."
   yield
   puts "End of method."
end
call_block do
   puts "I am inside call_block method."
end
```

OUTPUT

```
Start of method.
I am inside call_block method.
End of method.
```

In this example, a block is passed to the *call_block* method.
To invoke this block inside the method, we used a keyword, `yield`.
Calling `yield` will execute the code within the block that is provided to the method.

b). Passing a block to a method that takes one or more parameters.

CODE

```
def calculate(a,b)
   yield(a, b)
end

puts calculate(15, 10) {|a, b| a - b}
```

OUTPUT

```
5
```

In this example, we have defined a method *calculate* that takes two parameters a and b.
The `yield` statement invokes the block with parameters a and b, and executes it.

Task

You are given a partially complete code. Your task is to fill in the blanks (`_____`).

The factorial method computes: `n!` { n x $n-1$ x $....2$ x 1 }.