Using Functions in Clojure

Zachary Bennett
Software Engineer

@z_bennett_ zachbennettcodes.com

Same inputs, same outputs!

Function Structure

Name
What is the function called?

Parameters
What are the function’s inputs?

Body
What does the function do?

```
; Basic function syntax

(defn make-name

      [first-name]

      (str first-name “ Bennett”))
```

◀ **Function name**

◀ **Function parameters**

◀ **Function body**

Key Points

Clojure is a functional programming language!

Functions can be used as inputs and outputs

Endeavor to make your functions pure!

User-defined Functions

User-defined Functions

Named

Functions attached to a symbol to facilitate reuse

Anonymous

One-off functions, usually passed as arguments

```
;; Named Functions


(defn say-hello

  [first-name]

  (str "Hello " first-name))


(say-hello "Zachary")
```

◂ ”defn” assigns the function to a given name – a combination of “def” and “fn”.

◂ The parameter list is a vector

◂ The function body

◂ Invoking a named function looks like this

All about reuse!

```clojure
;; Anonymous Functions


(fn [first-name]

  (str "Hello " first-name))


; Can’t invoke later on



; Example usage
(map (fn [num] (+ num 1)) [1 2 3])

; Short-form usage
(map #(+ % 1) [1 2 3])
; => (2, 3, 4)

(#(+ %1 %2) 1 2)
; => 3
```

◀ **”fn” creates an anonymous function**

◀ **The parameter list is a vector**

◀ **The function body**

◀ **Using an anonymous function looks like this**

◀ **There is a shorter way to write it!**

◀ **Multiple parameters are counted**

Multi-arity and Variadic Functions

Multi-arity Functions

Multi-arity functions are functions that can take a differing number of parameters. For each “arity” there is a unique function implementation.

Variadic Functions

Variadic functions are functions that can take a differing number of parameters. There is only one function implementation.

A good example is “println”

```
;; Multi-arity Function


(defn my-printer

  ([] (my-printer "No parameters!"))

  ([one] (println one))

  ([one, two] (println one two)))


(my-printer)

(my-printer "One")

(my-printer "One" "Two")

(my-printer "One" "Two" "Three")
```

◂ First, defined a named function

◂ You can then define a separate implementation of the function by “arity”

◂ One “arity” can invoke another “arity”

◂ Each invocation with a valid arity will invoke the corresponding implementation

◂ An invalid invocation results in an ArityException being thrown!

```clojure
;; Variadic Function


(defn foo [first & rest]

  (println first)

  (doseq [arg rest] (println arg)))


(foo "First" "Second" 3 4 "Fifth")



#(println %1 %&)
```

- A variadic function is defined using defn. An ampersand marks the start of the variable number of parameters
- The variable parameters are put inside of a list which can be evaluated in the function body
- Logs each variable given to the console on a new line
- Anonymous, variadic functions can be created

Recursion

Recursion

In computer science, recursion is a means of solving a problem by breaking the problem up into smaller versions of itself.

Functions calling themselves!

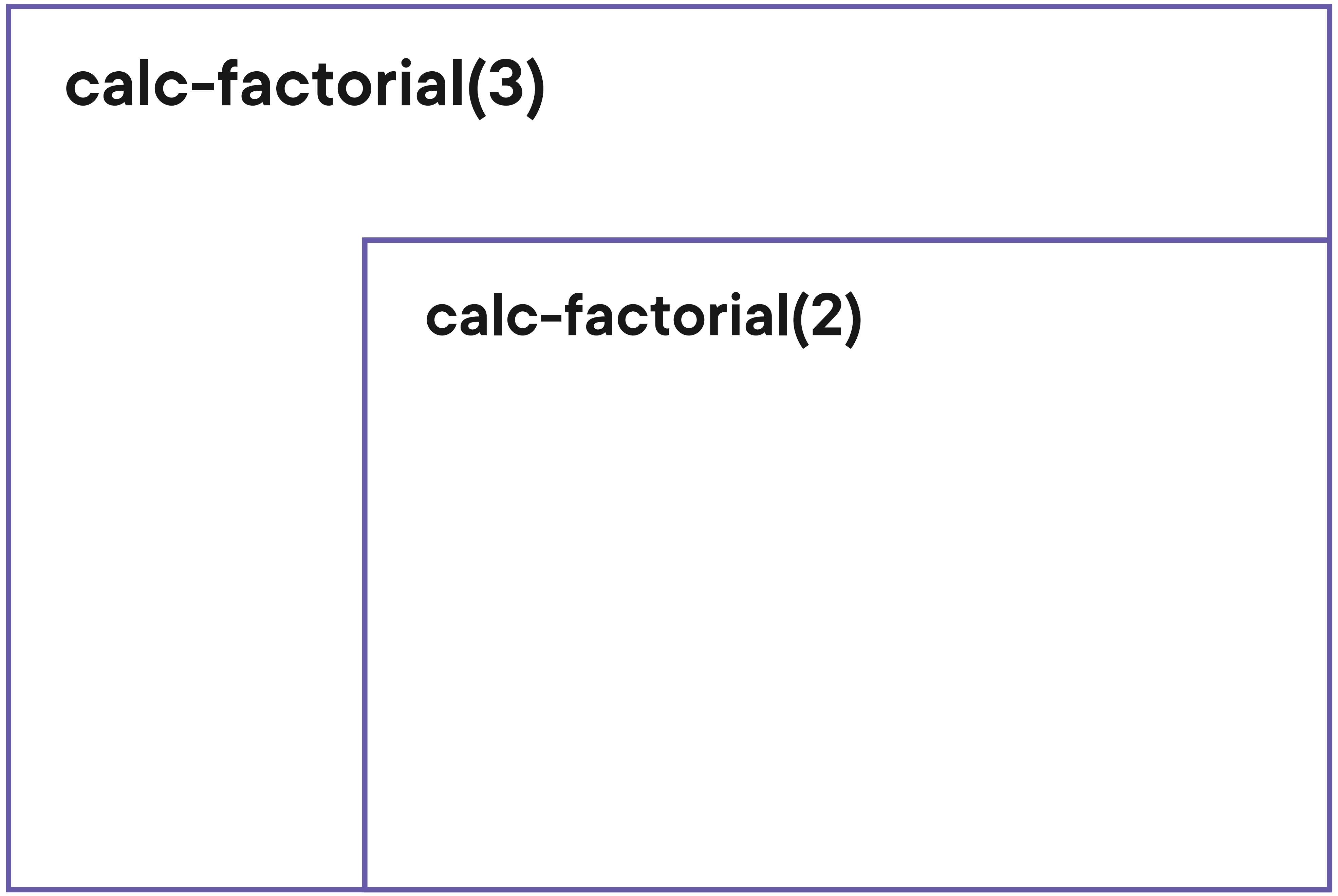
```clojure
;; Recursive Function

(defn calc-factorial [num]

  (if (zero? num)

    1

    (* num (calc-factorial (dec num)))))
```

- This function calculates the factorial of a number. For example, the factorial of 3 is 6 since 3 * 2 * 1 = 6.
- Here is an example of a “base case” that triggers the end of a recursive call.
- This final line of code is an example of recursion – the calc-factorial function is called within the calc-factorial function itself!

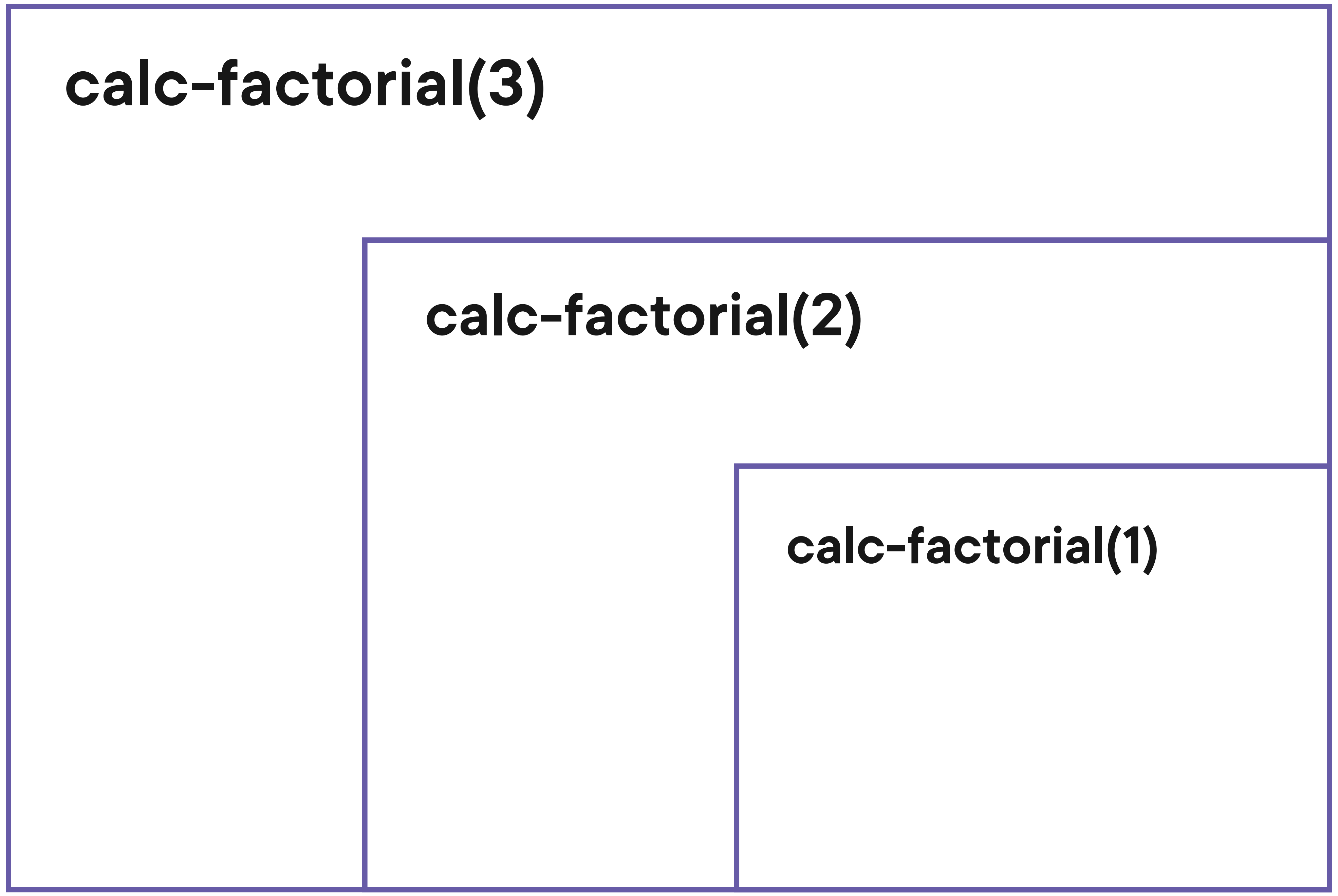
Recursion Example

calc-factorial(3)

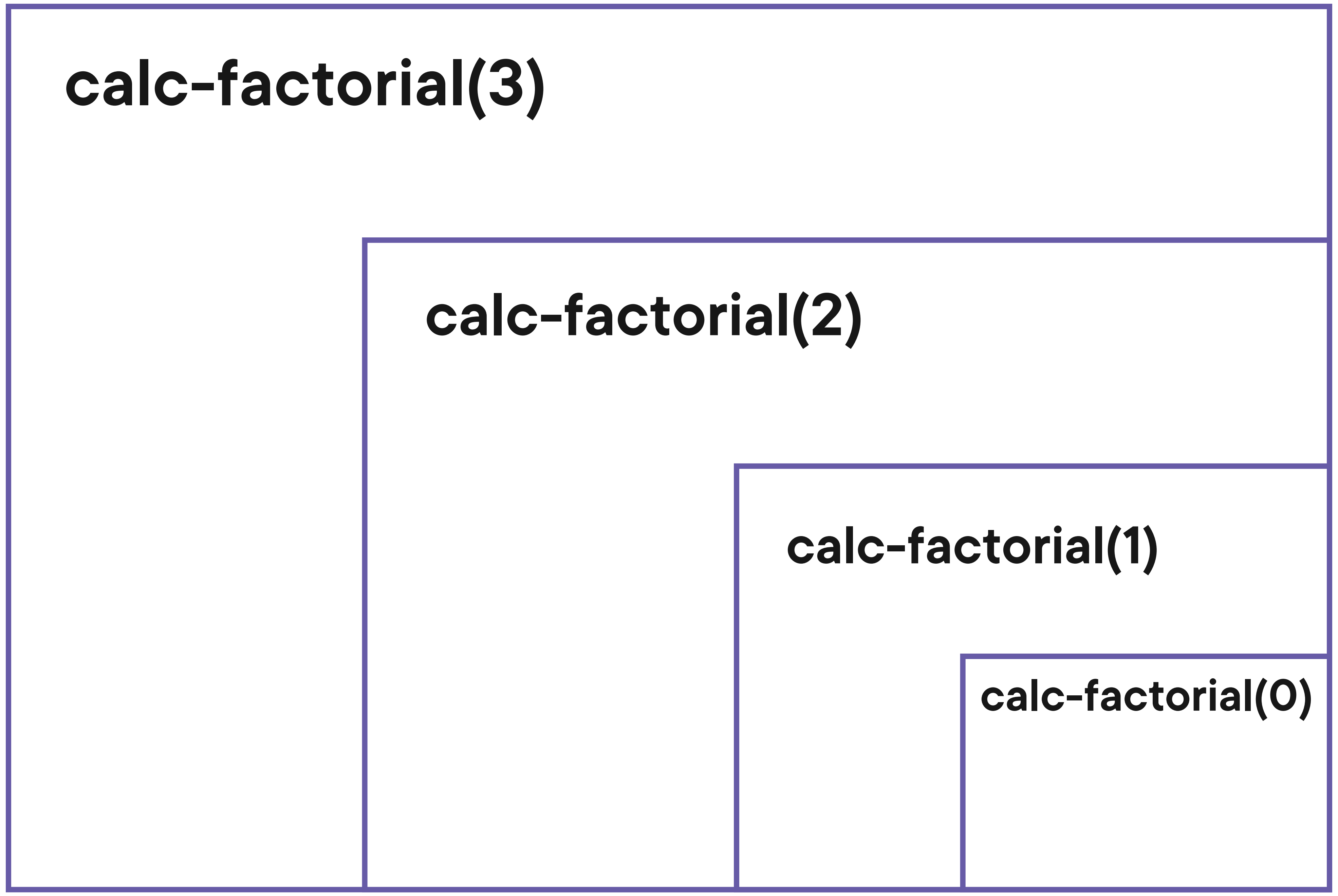
Recursion Example



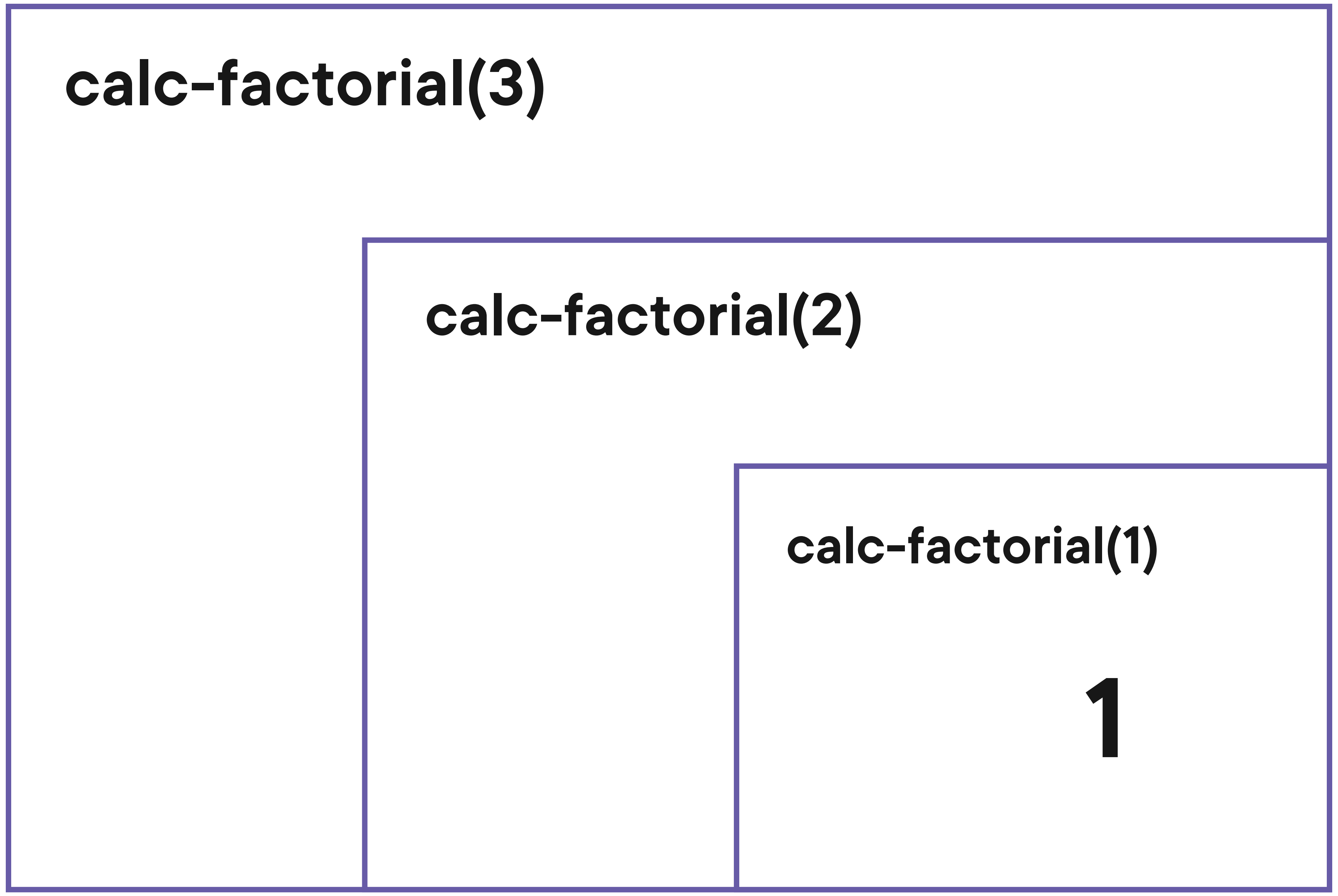
Recursion Example



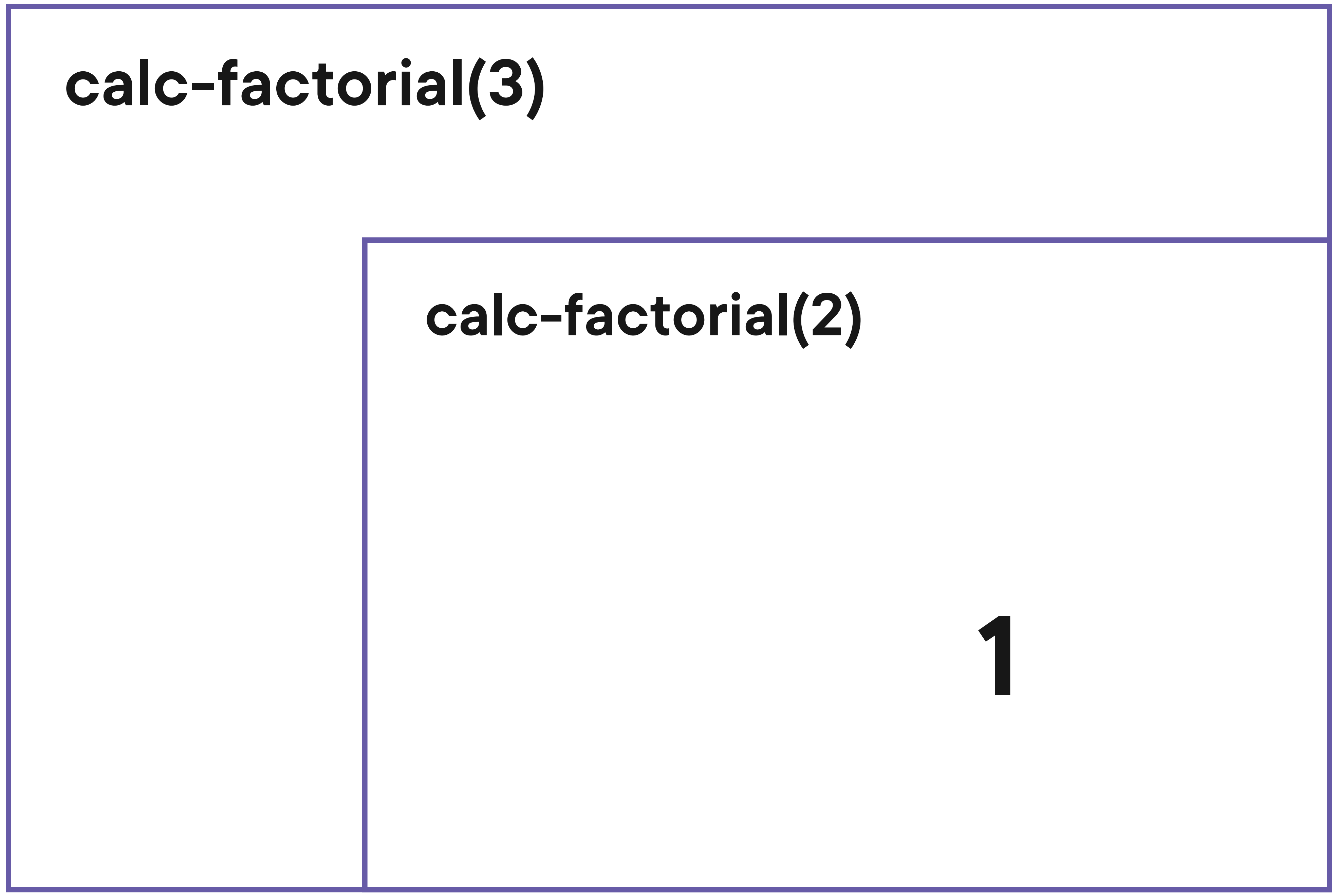
Recursion Example



Recursion Example



Recursion Example



Recursion Example

calc-factorial(3)

2

1

Recursion Example

calc-factorial(3)

2 * 1

Recursion Example

calc-factorial(3)

2

Recursion Example

$$3 * 2 = 6$$

Demo

Functions

- Named
- Anonymous
- Multi-arity
- Variadic