Unrestricted Lambda
The procedure `(lambda (x y z) ....)` takes 3 arguments. `(lambda (x y z w) ...)` takes 4. Sometimes we want to write a function that takes an indeterminate number of arguments. For example, we might want to have an average procedure that averages its arguments:

```
(avg 4) returns 4
(avg 3 4) returns 3.5
(avg 3 4 5) returns 4,
```
and so forth.
If we write a lambda expression with one parameter, without parentheses around this parameter, as in

(lambda args ...)

then when this procedure is call all of the actual arguments are collected in a list which is bound to the parameter args.

Here is our function avg:

(define avg
 (lambda args
   (lambda args
     (let ([sum (apply + args)]
           [n (length args)])
       (/ sum n)))))
You may have noticed that +, <, max, and other operators are defined as procedures in Scheme, but and is a form (a kind of expression), not a proc. This means that we can't apply and in a recursion. Here is a procedure-version of and:

```
(define and-proc
  (lambda args
    (cond
      [(null? args) #t]
      [(car args) (apply and-proc (cdr args))]
      [else #f]]))
```