

Variable-Arity Procedures

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Experiments with `display-line`
 - Exercise 2: Extending `display-line`
 - Exercise 3: Determining the number of arguments
 - Exercise 4: Experiments with `display-separated-line`
 - Exercise 5: A Clicker
 - Exercise 6: Multiple Separators

Exercises

Exercise 0: Preparation

- Please scan through the reading on variable arity procedures.
- Start DrScheme.

Exercise 1: Experiments with `display-line`

Here is the `display-line` procedure from the reading.

```
;;; Procedure:
;;; display-line
;;; Parameters:
;;; 0 or more values
;;; Purpose:
;;; Displays the strings terminated by a carriage return.
;;; Produces:
;;; Nothing
;;; Preconditions:
;;; (none)
;;; Postconditions:
;;; The standards
(define display-line
  (lambda arguments
    (let kernel ((rest arguments))
      (if (null? rest)
          (newline)
          (begin
             (display (car rest))
             (kernel (cdr rest))))))))
```

a. Try out some other calls to `display-line` to check what it prints. For example, try the following:

```
(display-line "going" "going" "gone")
(display-line "countdown:" 5 4 3 2 1 "done")
(display-line) ;; apply display-line to no arguments
```

b. Explain your results.

Exercise 2: Extending `display-line`

The current version of `display-line` prints all text together without spaces. Modify the code, so that one space is printed between any two adjacent values supplied as arguments to `display-line`. For instance, after your modifications, the example from the reading will change. It will now be ...

```
> (display-line "+--" "Here is a string!" "--+")
+-- Here is a string! --+
```

You may not use `display-separated-line` in your answer to this question.

Exercise 3: Determining the number of arguments

Define and test a procedure named `call-arity` that takes any number of arguments and returns the number of arguments it received (ignoring their values):

```
> (call-arity 'a #\b "c" '(d))
4
> (call-arity 0.0)
1
> (call-arity)
```

Exercise 4: Experiments with `display-separated-line`

- What happens if you invoke `display-separated-line` without giving it any arguments?
- What happens when you give it only one argument?
- What happens when you give it two arguments?
- What happens when you give it three arguments?

Exercise 5: A Clicker

Define and test a procedure `clicker` that takes one or more arguments, of which the first must be an integer and each of the others must be either the symbol `'up` or the symbol `'down`. `clicker` should start from the given integer, add 1 for each `'up` argument, subtract 1 for each `'down` argument, and return the result:

```
> (clicker 17 'up 'up)
19
> (clicker -12 'down 'up 'down 'down 'down)
-15
> (clicker 100)
100
```

Exercise 6: Multiple Separators

In writing, we often separate the last element of a list using a different separator than for the prior elements. For example, we might separate the all but the last element with commas and the last element with "and". Extend `display-separated-line` so that it requires two parameters (the default separator and the final separator) and supports as many the client provides.