CSSE 304 Day 27

Warm-up for `call/cc`:

1. A **receiver** is an argument (which happens to also be a procedure) passed to a procedure, with the intention that the procedure will eventually pass values to that argument. In some situation, receivers are referred to as "callbacks".

   a. The continuations that we pass to CPS procedures (in the scheme-procedure representation) are receivers.

   b. The consumer procedures that we pass `to call-with-values` are receivers.

   c. `call-with-output-file` is another example of a procedure that expects a receiver as an argument.

2. Pretend that we have a procedure `escape-+` that adds its arguments and returns this sum as the final answer, no matter what the context.
   a. `(* (escape-+ 5 6) 3)` ➔

   b. `(escape-+ (escape-+ 2 4) 5)` ➔

3. More generally, suppose that we have a procedure `escaper` that takes a procedure as an argument and returns an equivalent escape procedure.
   a. (escaper +) creates a procedure that is equivalent to `escape-+`

   b. `(+ 3 ((escaper +) 4 5))` ➔

   c.
```
(+ ((escaper (lambda (x)
               (- (* x 3) 7)))
    5)
   4)          ➔
```

   d. A slide gives details of how you can experiment with `escaper` in *Chez* Scheme.

4. Let *p* be a procedure. If an application of *p* abandons the current continuation and does something else instead, we call *p* an ***escape procedure***.
   a. An example of a Scheme escape procedure that we have already used:

   b. Is `escaper` an escape procedure?

5. We consider the `dining-out` procedure from the slides, along with a reading from the Springer and Friedman book. Your takeaway: In future discussions, I will often refer to "taking a photograph", "saving the photograph", and "rubbing a photograph." You should leave today's class knowing what those things mean.

Definition of `call/cc` and simple examples:

6. Note the detailed definition of call/cc that is on the slide whose title is CALL/CC.
   Here is a summary of that definition:

   (call/cc receiver) ➔ (**receiver continuation)**,
   Hence the name: call-with-current-continuation.
   Rephrasing it: What is that continuation?

   If **c** is a procedure that represents the execution context of this application of **call/cc**, then the continuation is equivalent to **(escaper c).**

7. call/cc example: **(+ 3 (call/cc (lambda (k) (* 2 (k 5)))))**
   a. The receiver is
   b. The context c of the call/cc application is
   c. The continuation is
   d. Thus (+ 3 (call/cc (lambda (k) (* 2 (k 5))))) is equivalent to

8. Another call/cc example : **(+ 3 (call/cc (lambda (k) (* 2 5))))**
   a. The receiver is
   b. The context c is
   c. The continuation is
   d. Thus (+ 3 (call/cc (lambda (k) (* 2 5)))) is equivalent to

9. Another call/cc example: **(+ 3 (call/cc (lambda (k) (k (* 2 5)))))**
   a. The receiver is
   b. The context c is
   c. The continuation is
   d. Thus (+ 3 (call/cc (lambda (k) (k * 2 5)))) is equivalent to

10.
```
(define xxx #f)
  (+ 5 (call/cc (lambda (k)
                  (set! xxx k)
                  2))) ; xxx is equivalent to?
   (* 7 (xxx 4))
```

11. **(call/cc procedure?)**