Simulating a Web Server

HTTP protocol is like calling a function:

```
(define total 0)

(define (a)
  `(("Current value:" ,total)
    "Call a2 to add 2"
    "Call a3 to add 3"))

(define (a2)
  (set! total (+ total 2))
  (a))

(define (a3)
  (set! total (+ total 3))
  (a))
```

Simulating a Web Server

Stateless variant is functions with arguments:

```
(define (b)
  (do-b 0))

(define (do-b total)
  `(("Current value:" ,total)
    "Call b2 with " ,total " to add 2"
    "Call b3 with " ,total " to add 3"))

(define (b2 total)
  (do-b (+ total 2)))

(define (b3 total)
  (do-b (+ total 3)))
```

Simulating a Web Server

For complex data, use `remember` and `lookup` to make a simple key:

```
(define (c)
  (do-c "*"))

(define (do-c total)
  (local [(define key (remember total))]
    `(("Current value:" ,total)
      "Call c2 with " ,key " to append \"hello\""
      "Call c3 with " ,key " to append \"goodbye\"")))

(define (c2 key)
  (do-c (string-append (lookup key) " hello")))

(define (c3 key)
  (do-c (string-append (lookup key) " goodbye")))
```

Simulating a Web Server

```
(define table empty)

(define (remember v)
  (local [(define n (length table))]
    (begin
      (set! table (append table
                          (list v)))
      n)))

(define (lookup key)
  (list-ref table key))
```

Direct Interactive Programs

But normally we write code more like this:

```
(define (d)
  (do-d 0))

(define (do-d total)
  (begin
    (printf "Total is ~a\nAdd 2 next?\n" total)
    (do-d (+ total
             (if (read) 2 3)))))
```

Direct Interactive Programs

Or like this:

```
(define (f)
  (do-f 0))

(define (num-read prompt)
  (begin
    (printf "~a\n" prompt)
    (read)))

(define (do-f total)
  (do-f (+ (num-read
            (format "Total is ~a\nNext number...\n"
                    total))
           total)))
```

We'd like to have a **`web-read`**...

Interactive Web Programs

Can we make this work?

```
(define (g)
  (do-g 0))

(define (web-read prompt)
  `(,prompt
     "To continue ..."))

(define (do-g total)
  ... (web-read
        (format "Total is ~a\nNext number...\n"
                total))
  ...)
```

`web-read` should not be specific to **`g`**

Interactive Web Programs

```
(define (g)
  (do-g 0))

(define (web-read prompt)
  (local [(define key (remember ...))]
    `(,prompt
      "To continue, call resume with" ,key "and value")))

(define (resume key val)
  ...)

(define (do-g total)
  ... (web-read
       (format "Total is ~a\nNext number...\n"
               total))
  ...)
```

What should we remember?

Interactive Web Programs

```
(define (g)
  (do-g 0))

(define (web-read prompt total do-g)
  (local [(define key (remember (list total do-g)))]
    `(,prompt
      "To continue, call resume with" ,key "and value")))

(define (resume key val)
  (local [(define l (lookup key))]
    ((first l) ... (second l) ... val ...)))

(define (do-g total)
  (web-read
   (format "Total is ~a\nNext number...\n"
           total)
   total
   do-g))
```

How should `(second l)` and `val` be combined?

Interactive Web Programs

```
(define (g)
  (do-g 0))

(define (web-read/k prompt cont)
  (local [(define key (remember cont))]
    `(,prompt
      "To continue, call resume/k with" ,key "and value")))

(define (resume/k key val)
  (local [(define cont (lookup key))]
    (cont val)))

(define (do-g total)
  (web-read/k
   (format "Total is ~a\nNext number...\n"
           total)
   (lambda (val)
     (do-g (+ total val)))))
```

Interactive Web Programs

```
(define (h)
  (+ (num-read "first number")
     (num-read "second-number")))
```

$\Rightarrow$

```
(define (h)
  (web-read/k "First number"
              (lambda (v1)
                (web-read/k "Second number"
                            (lambda (v2)
                              (+ v1 v2))))))
```

But what if we want to use **h** twice (to add two pairs of numbers)?

Interactive Web Programs

```
(define (h)
  (+ (num-read "first number")
     (num-read "second-number")))
(define (i)
  ; works fine
  (begin (h) (h)))
```

```
(define (h)
  (web-read/k "First number"
              (lambda (v1)
                (web-read/k "Second number"
                            (lambda (v2)
                              (+ v1 v2))))))
(define (i)
  ; first call is useless
  (begin (h) (h)))
```

Continuation-Passing Style

If a function uses **`web-read/k`**, then to make it composable, it must always take a continuation

```
(define (h) (do-h identity))
(define (do-h cont)
  (web-read/k "First number"
              (lambda (v1)
                (web-read/k "Second number"
                            (lambda (v2)
                              (cont (+ v1 v2)))))))

(define (i) (do-i identity))
(define (do-i cont)
  (do-h (lambda (sum)
          ; web-pause/k is like web-read/k,
          ; but with no particular result
          (web-pause/k sum
                       (lambda ()
                         (do-h cont))))))
```

Continuation-Passing Style

```
(define (web-pause/k prompt cont)
  (local ((define key (remember cont)))
    `(,prompt
      "To continue, call p-resume/k with" ,key)))

(define (p-resume/k key)
  ((lookup key)))
```