Continuations

On the Web and in your OS

Jay McCarthy
PLT @ Brigham Young University

- **What are Continuations?**
- **How are they used on the Web?**
- **How many fit on the end of pin?**

What is a continuation?

What is a continuation?

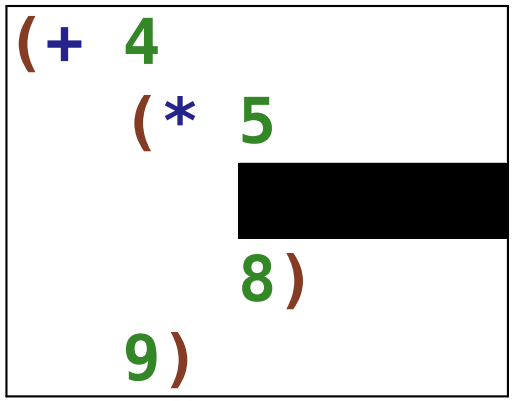
The 'rest' of the computation

```
(+ 4
   (* 5
      (- 6 7)
      8)
   9)
```

Result: -27

```
(+ 4
   (* 5
      (- 6 7)
      8)
   9)
```

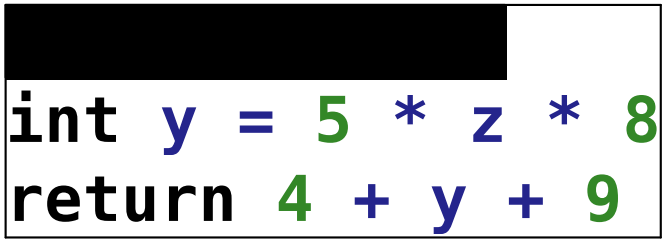
Result: -27



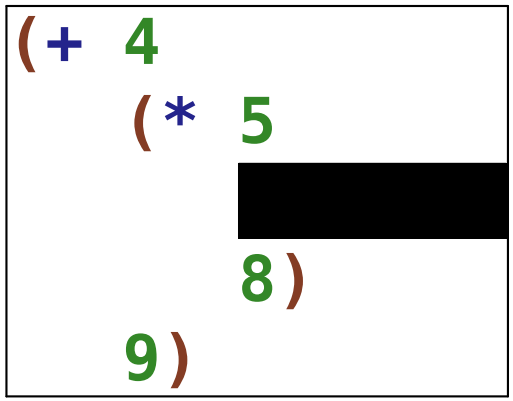
Result: -27

```
int z = 6 - 7
int y = 5 * z * 8
return 4 + y + 9
```

Result: -27



Result: -27



Result: -27

```
(λ (z)
  (+ 4
     (* 5
        z
        8)
     9))
```

Result: #<procedure>

```
(+ 4
   (* 5
      (let/cc k
        (- 6 7))
      8)
   9)
```

Result: -27

```
(+ 4
   (* 5
      (let/cc k
        (k (- 6 7)))
      8)
   9)
```

Result: ?

```
(+ 4
   (* 5
      (let/cc k
        (k (- 6 7)))
      8)
   9)
```

Result: -1067

```
(+ 4
   (* 5
      (let/cc k
        (k (- 6 7)))
      8)
   9)
```

Result: -27

```
(λ (z)
  (+ 4
     (* 5
        z
        8)
     9))
```

Result: #<procedure>

```
(λ (z)
  (exit
   (+ 4
      (* 5
         z
         8)
      9)))
```

Result: #<procedure>

```
(+ 4
   (* 5
      (let/cc k
        (k (- 6 7)))
      8)
   9)
```

Result: -27

```
(+ 4
   (* 5
      (+ 4
         (* 5
            (- 6 7)
            8)
         9)
      8)
   9)
```

Result: -1067

```
(+ 4
   (* 5
      (exit
       (+ 4
          (* 5
             (- 6 7)
             8)
          9))
      8)
   9)
```

Result: -27

UNdelimited

```
(exit
 (with-prompt
  (+ 4
     (* 5
        (let/cp k
                (k (- 6 7)))
        8)
     9)))
```

Result: -1067

```
(exit
 (with-prompt
  (+ 4
     (* 5
        (let/cp k
                (* 2 (k (- 6 7))))
        8)
     9)))
```

Result: -2147

```
(exit
 (with-prompt
  (+ 4
     (* 5
        (let/cp k
                (k (* 2 (k (- 6 7)))))
        8)
     9)))
```

Result: -85867

```
(exit
 (with-prompt
  (+ 4
     (* 5
        (let/cp k
                (if (positive? (k (- 6 7)))
                  -1
                  0))
        8)
     9)))
```

Result: 13

```
(exit
 (* -1
    (with-prompt
     (+ 4
        (* 5
           (let/cp k
                   (if (positive? (k (- 6 7)))
                     -1
                     0))
           8)
        9))))
```

Result: -13

Continuations

- The future of the computation
- Can be represented as a function

Continuations

- The future of the computation
- Can be represented as a function
- Undelimited (the entire future)
- Delimited (the future up to a point)

Continuations

- The future of the computation
- Can be represented as a function
- Undelimited (the entire future)
- Delimited (the future up to a point)
- Languages like Racket give you all of this

- What are Continuations?
- How are they used on the Web?
- How many fit on the end of pin?

```
(printf
 "Sum is: ~a"
 (+ (prompt "Enter first number:")
    (prompt "Enter second number:")))
```

Enter first number:

```
(printf
 "Sum is: ~a"
 (+ (prompt "Enter first number:")
    (prompt "Enter second number:")))
```

Enter first number: 2

```
(printf
 "Sum is: ~a"
 (+ (prompt "Enter first number:")
    (prompt "Enter second number:")))
```

Enter first number: 20

Enter second number:

```
(printf
 "Sum is: ~a"
 (+ (prompt "Enter first number:")
    (prompt "Enter second number:")))
```

Enter first number: 20

Enter second number: 2

```
(printf
 "Sum is: ~a"
 (+ (prompt "Enter first number:")
    (prompt "Enter second number:")))
```

Enter first number: 20

Enter second number: 22

Sum is: 42

```
(web-printf
 "Sum is: ~a"
 (+ (web-prompt "Enter first number:")
    (web-prompt "Enter second number:")))
```

```
(web-dispatch
 ["/"
  (web-prompt&go-to
   "Enter first number:"
   "/sum/")]
 ["/sum/$first"
  (web-prompt&go-to
   "Enter second number:"
   "/sum/$first/")]
 ["/sum/$first/$second"
  (web-printf
   "Sum is: ~a"
   (+ first second))])
```

```
(define (continuation1 answer1)
  (define (continuation2 answer2)
    (web-printf
     "Sum is: ~a"
     (+ answer1
        answer2)))
  (web-prompt&go-to
   "Enter second number:"
   continuation2))
(web-prompt&go-to
 "Enter first number:"
 continuation1)
```

```
(define (web-prompt display-text)
  (let/cc server-continuation
    (define new-url
      (store-in-dispatch-table!
       server-continuation))
    (send-to-client
     (web-prompt&go-to
      display-text
      new-url))))
```

Plus a lot of details...

Refer to OOPSLA 2010, ICFP 2009, and HOSC 2007 papers for most of them.

```
(define (web-server some-servlet)
  (while true
    (define conn (wait-for-connection))
    (define req (get-a-request conn))
    (define resp (some-servlet req))
    (display-on-wire conn resp)))
```

```
(define (web-server some-servlet)
  (while true
    (define conn (wait-for-connection))
    (while (connected? conn)
      (define req (get-a-request conn))
      (define resp (some-servlet req))
      (display-on-wire conn resp))))
```

```
(define (web-prompt display-text)
  (let/cc server-continuation
    (define new-url
      (store-in-dispatch-table!
       server-continuation))
    (send-to-client
     (web-prompt&go-to
      display-text
      new-url))))
```

```
(define (web-server some-servlet)
  ....
  (define conn1 (wait-for-connection))
  (define req1 (get-a-request conn1))
  (define resp1 (some-servlet req1))
  (display-on-wire conn1 resp1)
  ....
  (define conn2 (wait-for-connection))
  (define req2 (get-a-request conn2))
  (define resp2 (some-servlet req2))
  (display-on-wire conn1 resp2)
  ....)
```

```
(define (web-server some-servlet)
  (while true
    (define conn (wait-for-connection))
    (while (connected? conn)
      (define req (get-a-request conn))
      (define resp
        (with-prompt
         (some-servlet req)))
      (display-on-wire conn resp))))
```

```
(define (web-prompt display-text)
  (let/cc server-continuation
    (define new-url
      (store-in-dispatch-table!
       server-continuation))
    (send-to-client
     (web-prompt&go-to
      display-text
      new-url))))
```

```
(define (web-prompt display-text)
  (let/cc server-continuation
    (define new-url
      (store-in-dispatch-table!
       server-continuation))
    (send-to-client
     (web-prompt&go-to
      display-text
      new-url))))
```

```
(define (web-prompt display-text)
  (let/cc server-continuation
    (define new-url
      (store-in-dispatch-table!
       server-continuation))
    (abort
     (web-prompt&go-to
      display-text
      new-url))))
```

Plus some tiny details...

Prompts are first-class

- **What are Continuations?**
- **How are they used on the Web?**
- **How many fit on the end of pin?**

  Web Components

```
(define (page header)
  (send/suspend/dispatch
   (λ (embed/url)
     `(p
       (h1 ,header)
       (a ([href
            ,(embed/url
              (λ (req) (page "First")))])
          "First")
       ,(include-counter embed/url)
       (a ([href
            ,(embed/url
              (λ (req) (page "Second")))])
          "Second")))))
```

```
(define (include-counter embed/url)
  (let/cc k
    (let loop ([cnt 0])
      (k
       `(p
         (a ([href
              ,(embed/url
                (λ (req) (loop (sub1 cnt))))])
            "-")
         ,(number->string cnt)
         (a ([href
              ,(embed/url
                (λ (req) (loop (add1 cnt))))])
            "+"))))))
```

- **What are Continuations?**
- **How are they used on the Web?**
- **How many fit on the end of pin?**

Event-based Network Servers

```
(define req1 (read-request fd))
(define req2 (read-request fd))
(define res1 (compute-answer req1 req2))
(send-answer fd res1)
(free-resources req1 req2 res1)
```

```
(define (evloop evts)
  (sync
   (for/list ([e (in-list evts)])
     (e
      (λ (new-evts)
        (append
         new-evts
         (remove e evts)))))))
```

```
(evloop
 (list
  (λ (next)
    (handle-evt
     (read-request-evt fd)
     (λ (req1)
       ....)))))
```

```
(next
 (list
  (λ (next)
    (handle-evt
     (read-request-evt fd)
     (λ (req2)
       ....)))))
```

```
(define res1 (compute-answer req1 req2))
(next
 (list
  (λ (next)
    (handle-evt
     (send-answer-evt fd res1)
     (λ ()
       (free-resources req1 req2 res1)
       (next empty))))))
```

```
(define (read-request fd)
  (let/cc k
    (switch-to-evloop-until
     (read-request-evt fd)
     k)))
```

- **What are Continuations?**
- **How are they used on the Web?**
- **How many fit on the end of pin?**

Operating Systems

```
(define threads empty)
(define (spawn body)
  (set! threads (snoc threads body)))
(define (switch)
  (unless (empty? threads)
    (define next (first threads))
    (set! threads (rest threads))
    (next)))
(define (yield)
  (let/cc k
    (spawn k)
    (switch)))
```

```
(define (looper)
  (for ([i (in-range 5)])
    (displayln i)
    (yield))
  (switch))
(spawn looper)
(looper)
```

```
(struct kernel (threads safe))
```

```
(define (main)
  (define N 5)
  (thread
   (λ ()
     (for ([i (in-range (+ N 2))])
       (printf
        "iter: ~a -> ~a\n"
        i
        (swap (* 2 (add1 i))))))))
  (thread
   (λ ()
     (for/fold ([sum 0])
         ([i (in-range N)])
       (printf
        "adder: ~a -> ~a\n"
        i
        (swap (+ i sum)))
       (+ i sum)))))
```

```
(define (boot main)
  (define initial (kernel (list main) 0))
  (let loop ([ks initial])
    (unless (empty? (kernel-threads ks))
      (loop (step-one-thread ks)))))
```

```
(define (step-one-thread ks)
  (match-define
   (kernel (cons top-thread other-threads)
           safe)
   ks)
  (define syscall
    (run-thread-until-syscall top-thread))
  (execute-syscall
   syscall
   (kernel other-threads safe)))
```

```
(struct syscall (user-context))

(struct syscall:thread syscall (child-thunk))
(struct syscall:end syscall ())
(struct syscall:swap syscall (new-safe))
```

```
(define (execute-syscall call kernel-state)
  (match-define
   (kernel threads safe)
   kernel-state)
  (match call
    [(syscall:thread user-ctxt child-t)
     (kernel (list* user-ctxt child-t threads)
             safe)]
    [(syscall:end user-ctxt)
     (kernel threads
             safe)]
    [(syscall:swap user-ctxt new-safe)
     (kernel (snoc threads
                   (λ () (user-ctxt safe)))
             new-safe)]))
```

```
(define (thread child-t)
  (call-with-composable-continuation
   (λ (user-ctxt)
     (abort-current-continuation
      kernel-prompt-tag
      (syscall:thread user-ctxt child-t)))
   kernel-prompt-tag))
```

```
(define-syntax-rule
  (define-syscall-throw user-id syscall-id)
  (define (user-id . syscall-args)
    (call-with-composable-continuation
     (λ (user-ctxt)
       (abort-current-continuation
        kernel-prompt-tag
        (apply syscall-id user-ctxt syscall-args)))
     kernel-prompt-tag)))
(define-syscall-throw thread syscall:thread)
(define-syscall-throw end syscall:end)
(define-syscall-throw swap syscall:swap)
```

```
(define kernel-prompt-tag
  (make-continuation-prompt-tag 'kernel))
(define (run-thread-until-syscall thread-ctxt)
  (call-with-continuation-prompt
   (λ ()
     (thread-ctxt)
     (end))
   kernel-prompt-tag
   values))
```

```
(struct kernel (threads safe))
(define (boot main)
  (define initial (kernel (list main) 0))
  (let loop ([ks initial])
    (unless (empty? (kernel-threads ks))
      (loop (step-one-thread ks)))))
(define (step-one-thread ks)
  (match-define (kernel (cons top-thread other-threads) safe) ks)
  (define syscall (run-thread-until-syscall top-thread))
  (execute-syscall syscall (kernel other-threads safe)))
(struct syscall (user-context))
(struct syscall:thread syscall (child-thunk))
(struct syscall:end syscall ())
(struct syscall:swap syscall (new-safe))
(define (execute-syscall call kernel-state)
  (match-define (kernel threads safe) kernel-state)
  (match call
    [(syscall:thread user-ctxt child-t)
     (kernel (list* user-ctxt child-t threads) safe)]
    [(syscall:end user-ctxt)
     (kernel threads safe)]
    [(syscall:swap user-ctxt new-safe)
     (kernel (snoc threads (λ () (user-ctxt safe))) new-safe)]))
(define-syntax-rule
  (define-syscall-throw user-id syscall-id)
  (define (user-id . syscall-args)
    (call-with-composable-continuation
     (λ (user-ctxt)
       (abort-current-continuation
        kernel-prompt-tag
        (apply syscall-id user-ctxt syscall-args)))
     kernel-prompt-tag)))
(define-syscall-throw thread syscall:thread)
(define-syscall-throw end syscall:end)
(define-syscall-throw swap syscall:swap)
(define kernel-prompt-tag
  (make-continuation-prompt-tag 'kernel))
(define (run-thread-until-syscall thread-ctxt)
  (call-with-continuation-prompt
   (λ ()
     (thread-ctxt)
     (end))
   kernel-prompt-tag
   values))
```

Jay McCarthy

 @ 

PLT Brigham Young University

Twitter: **@jeapostrophe**

Github: **jeapostrophe**

Blog: **http://jeapostrophe.github.io**