```
#lang racket

(define (our-if c1 t f)
  (printf "Begin function\n")
  (if c1
      t
      f))

; Does it work?

(our-if (= 4 4) 5 6)

(our-if (= 4 4) (printf "true!\n") (printf
"false!\n"))

(our-if (= 4 4) (+ 1 2) (+ 3 4))

(if (= 4 4) (printf "true!\n") (printf
"false!\n"))

;; How do we hold off on evaluating
arguments to functions?
;; With functions!

(our-if (= 4 4)(lambda ()(printf
"true!\n"))(lambda ()(printf "false!\n")))

(define (our-if2 c1 t f)
  (if c1
      (t)
      (f)))

(our-if2 (= 4 4)(lambda () (printf
```

```
29 "true!\n"))(lambda () (printf "false!\n")))
30
31 ;; Why does this work?
32 ;; Functions do two things: abstract over
32 data and delay evaluation
33
34 (printf "Hello world!\n")
35
36 (lambda () (printf "Hello world!\n")) ;
36 Operation packed up, awaiting deployment
37
38 ( (lambda () (printf "Hello world!\n")) ) ;
38 Function application -> operation is
38 triggered
39
40
41 (define (foo x y z w v)
42   (printf "Hello world!"))
43
44 (foo (+ 1 2) (printf "y") (printf "z") (-
44 10 1) (printf "v"))
```