Soluciones Examen LPP
Convocatoria de Junio

Pregunta 1

```
(define (add-numbers frase)
   (cond ((empty? frase) 0)
      ((number? (first frase))
       (+ (first frase)(add-numbers (bf frase))))
       (else (add-numbers (bf frase)))))
```

Pregunta 2

a)

```
>((lambda (x y z) (x (_y__2__z_) z)) * + 3)
> 15
```

b)

```
>(((lambda (a b) b) + __*__) 3 5)
> 15
```

Pregunta 3

```
(define (pref wd char)
   (cond
      ((empty? wd) "")
      ((equal? (first wd) char) "")
      (else (word (first wd) (pref (bf wd) char)))))
```

Pregunta 4

```
(define fecha list)
(define dia car)
(define mes cadr)
(define anno caddr)

(define (fin-mes fe)
      (cond  ((and (or (= (mes fe) 1) (= (mes fe) 3) (= (mes fe) 5) (=
(mes fe) 7) (= (mes fe) 8) (= (mes fe) 10) (= (mes fe) 12)) (= (dia
fe) 31)) #t)
                ((and (or (= (mes fe) 4) (= (mes fe) 6) (= (mes fe) 9)
(= (mes fe) 11)) (= (dia fe) 31)) #t)
                ((and (= (mes fe) 2) (= (dia fe) 28)) #t)
             (else #f)))

Sol2:
(define (fecha dd mm aa)
   (+ (* aa 10000) (* mm 100) dd))
(define (dia fe) (remainder fe 100))
(define (mes fe) (div (remainder fe 10000) 100))
(define (anno fe) (div fe 10000))
```

Pregunta 5

```
(define (max-levels lista)
  (cond
    ((null? lista) 0)
    ((leaf? lista) 0)
    (else (max (1+ (max-levels (car lista)))
               (max-levels (cdr lista))))))
```

Otra:

```
(define (max-levels lista)
  (cond ((null? lista) 1)
        ((pair? (car lista))
           (max (1+ (max-levels (car lista)))
                    (max-levels (cdr lista))))
        (else (max-levels (cdr lista)))))


(define (mayor lista)
  (if (null? (cdr lista))
    (car lista)
    (max (car lista) (mayor (cdr lista)))))
```

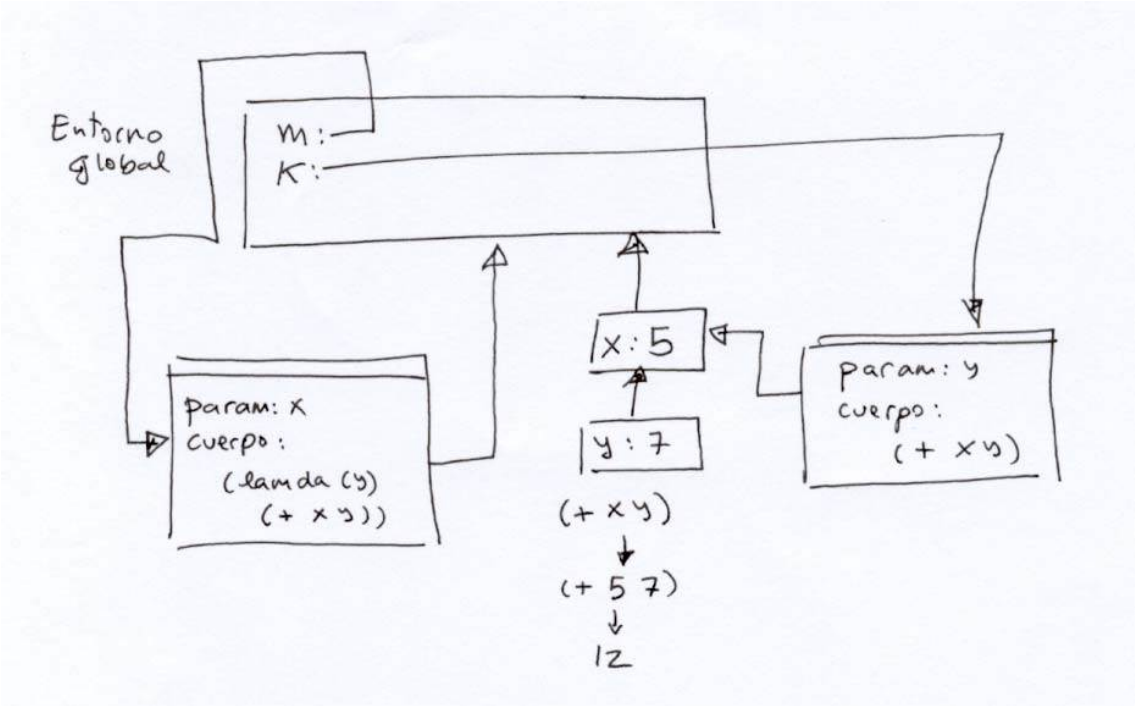
Pregunta 6

```
(define (conversion desde hasta)
      (if (eq? desde hasta)
            1
            (let ((next (get 'convert desde)))
                  (* (contents next)
                  (conversion (type-tag next) hasta)))))
```

Pregunta 7

```
(load "obj.ss")
(define-class (animal)
   (class-vars (animales '()))
   (initialize (set! animales (cons self animales)))
   (instance-vars (vidas 1))
   (method (estado) vidas)
   (method (cambioVidas x) (if (> x vidas) (set! vidas x)
                                          (error "No se puede redefinir
para restar vidas")))
   (method (finVida) (if (> vidas 0)(set! vidas (- vidas 1))
                           (error "Se acabaron las vidas")))
   (method (total-animales) (length animales)))
(define-class (gato)
   (parent (animal))
   (class-vars (num-gatos 0))
   (initialize (set! num-gatos (1+ num-gatos))
               (ask self 'cambioVidas 7))
   (method (total-gatos) num-gatos))
(define-class (perro)
   (parent (animal))
   (class-vars (num-perros 0))
   (initialize (set! num-perros (1+ num-perros)))
   (method (total-perros) num-perros))
```

Pregunta 8



Pregunta 9

(a)

SI NO	(define (swap-cars! x y) (let ((temp x)) (set! x (cons (car y) (cdr x))) (set! y (cons (car temp) (cdr y)))))	**NO**. SET! sólo está cambiando las variables locales “x” e “y”, pero en el entorno global las listas siguen siendo las originales.
SI NO	(define (swap-cars! x y) (let ((temp (car x))) (set! (car x) (car y)) (set! (car y) temp)))	**NO.** El primer argumento de SET! tiene que ser el nombre de una variable. Se produciría error de sintaxis.
SI NO	(define (swap-cars! x y) (set-car! x (car y)) (set-car! y (car x)))	**NO.** Al hacer el primer SET-CAR! perdemos el valor original de (CAR X).
SI NO	(define (swap-cars! x y) (let ((temp (cons (car x) (cdr x)))) (set-car! x (car y)) (set-car! y (car temp))))	**SI.**

(b)

```
(define (swap-cars! X y)
        (set! x (cons (car y) x))
        (set-car! y (cadr x))
        ;; Rellena el espacio en blanco con una
        ;; llamada a set!, set-car! o set-cdr!

        (____SET-CAR! (CDR X) (CAR X)_____))
```