Scheme : Parentheses

- everything is parentheses

Python	Scheme
`func(par1, par2)`	`(func par1 par2)`
`x + 1`	`(+ x 1)`
`x == 2`	`(= x 2)` (numbers only)
`x.equals(y)`	`(equal? x y)`
`x == y`	`(eqv? x y)`
`x = 2`	`(define x 2)`
`def x():` `  return 2`	`(define (x) 2)` `(define (x y z) (return (y+z)))`
`Link(1)` `Link(1, Link.empty)`	`(cons 1 nil)`
`Link(1, Link(2))`	`(cons 1 (cons 2 nil))`
`s.first`	`(car s)`
`s.rest`	`(cdr s)`
`s.rest.first`	`(car (cdr s))`

`if <predicate>:`
`  <consequence>`
`else:`
`  <alternative>`
	`(if pred conseq alter)`
	`(if pred`
	`  conseq`
	`  alter)`

```
if <predicate 1>:
    <consequence 1>
elif <predicate 2>:
    <consequence 2>
  :
else:
    <alternative>
```

```
(cond ((pred1) cons1)
      ((pred2) cons2)
       :
      (else alternative)
)
```

- Some useful operators

even? odd? null? modulo equal? eq?

null? → if lst is Link.empty()

- 0 & nil are truthy!!!

Warm Up:

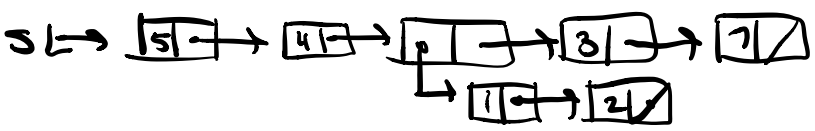
A:
```
(define x (+ 1 2 3))
```

B:
```
(define (x) (+ 1 2 3))
```

A:
```
x = 1 + 2 + 3
```

B:
```
def x():
    return 1+2+3
```

```
(define s '(5 4 (1 2) 3 7))
```




```
(car (cdr (cdr (cdr s))))
```

s.rest.rest.rest.first

Q2:

```
(define (fib n)
    (if (< n 2)
        n
        (+ (fib (- n 1))
           (fib (- n 2)))
    )
)
```

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8

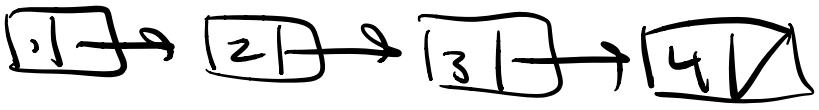
alt:

```
(define (fib n)
  (cond ((= n 1) 1)
        ((= n 0) 0)
        (else (+ (fib (- n 1))
                 (fib (- n 2))
              )
        )
  )
)
```

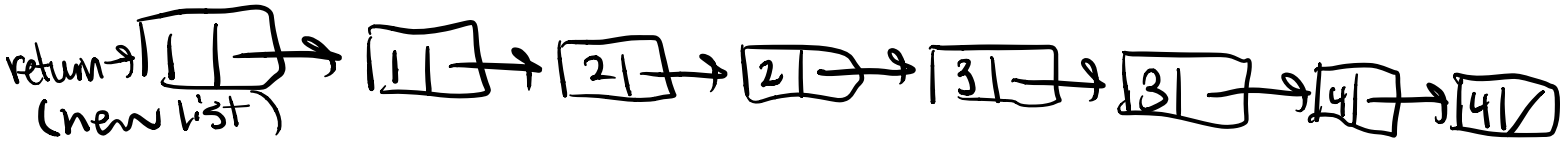
Q5

```
(define (duplicate lst)
  (if   (null? lst)
        nil
        (cons (car lst)
              (cons (car lst)
                    (duplicate (cdr lst))
              )
        )
  )
)
```

given { (1 2 3 4)



return → (new list)



(mutation doesn't really exist)

```
def duplicate(lst):
    if lst is Link.empty:
        return Link.empty

    else:
        return Link(lst.first, Link(lst.first, duplicate(lst.rest)))
```