# Office Hours

# CS-151

Define Activity Structure
- Descirption, Day-of-Week-
  Location

```
(define-type Calendar (Listof Activity))
```

Activity is a list

List of integers

```
(define my-list (list 2 3 4 5))
```

```
(: act1: Activity)
(define act1 (Activity "class1" 'Monday
                                    "online"))

   (: act2: Activity)
(define act2 (Activity "class2"
                       'Wednesday
                       "online"))
```

Create Point with these values

```
(define my-point (Point 1.0 2.0))
```

constructor

```
(define cs-151 (list act1 act2
                     act3 act4))
```

(define cs-151) (list (Activity "class1"

Problem 2:

T is a tree

[ max-tree-descandant-height (T) = max (heigh-of T, max-cone-descendut-height(C )) ]

C is the pine cone of the tree T.

· P is my pine-cone

max-cone-descendant-height (P) = max-tree-descendant-height (T1)

T1 is the tree contained in P.

<u>Base case</u>

Pine-Cone has Pine-Tree, it's height is 0.

```
(: max-tree-descendant-height :
   Pine-Tree → Real)
(define (max-tree-descendant-height
                t)
```

```
 (cond
[
(symbol? (Pine-Tree-seed t))
  (Pine-Tree-height t)]


[else (max (Pine-Tree-height t)
           (max-cone-descendant-height
             Pine-Tree-seed t))])


(define (max-tree-descendant-height
                 t)  ✓
(match (Pine-Tree-seed t)
['nothing (Pine-Tree-height t)]
[(Pine-Cone _ _) (max (Pine-Tree-height t)
   (max-cone-descendant-height
      (Pine-Tree-seed t)))]))
```

3.

n is odd if and only

n-2 is odd.

Base-0 even: False

Bas 1- odd. True.

my-odd? (4) = my-odd? (2)

= my-odd? (0) ⇒ False

my-odd? (5) = my-odd? (3)

= my-odd? (1) ⇒ True.

Recursive formula

my-odd? (n) = my-odd? (n-2)

my-odd? (0) = false

my-odd? (1) = True.

```
(pred (pred num))

(: my-odd? : Nat → Boolean)
(define (my-odd? n)
 (match n            → 0
  ['zero #f]            ✓
  [(Succ 'zero) #t]         ✓
 [(Succ (Succ m))           ]
(define-type Nat (U 'zero Succ))
 (define-struct Succ
    ([nat : Nat]))
```

(my-odd? 'zero) ⇒ #f

(my-odd? 'zero) ⇒ error-message

(Succ 'zero)

$$\boxed{\text{'zero}}$$

One is a structure of type Succ which contains the symbol 'zero.

'one

Two is structure of type Succ which contains One.

$$\boxed{\boxed{\text{'zero}}} \neq \text{'}\underline{\text{two}}$$

$$(a > b) \Longleftrightarrow (a-1 > b-1)$$

$$(4 > 3) \Longleftrightarrow (3 > 2)$$

// $\underline{\text{true}}$

$$\Longleftrightarrow (2 > 1) \Longleftrightarrow (1 > 0)$$

(: my-gte :

my-gte (a, b)
= my-gte (a-1, b-1)
if b == 0 (b 'zero)
then it is true.

my-gte (3, 4)
= my-gte (2, 3)
= my-gte (1, 2)
= my-gte (0, 1) ⇒ base case!
<u>false</u>

if n1 is 'zero
    but n2 is not 'zero
    then false

if n2 is 'zero
    then true

my-gle (4,4) = my-gle (3,3)
    = my-gle (2,2)
= my-gle (1,1) = my-gle (0,0)
    ⇒ true

a == b
if and only if
(a-1) == (b-1)

if a == 0 and b == 0
true

if a == 0 and b != 0
false

if a != 0 and b == 0
false

equal?(3,3)
=equal?(2,2)
=equal?(1,1) = equal?(0,0)
⇒ true.

equal?(2,3)
= equal?(1,2)
= equal?(0,1) ⇒ false

[ ]

```
(check-expect (predsubint
  (succ 'zero)) 1)
```