Lecture 4

Configure Dr Racket

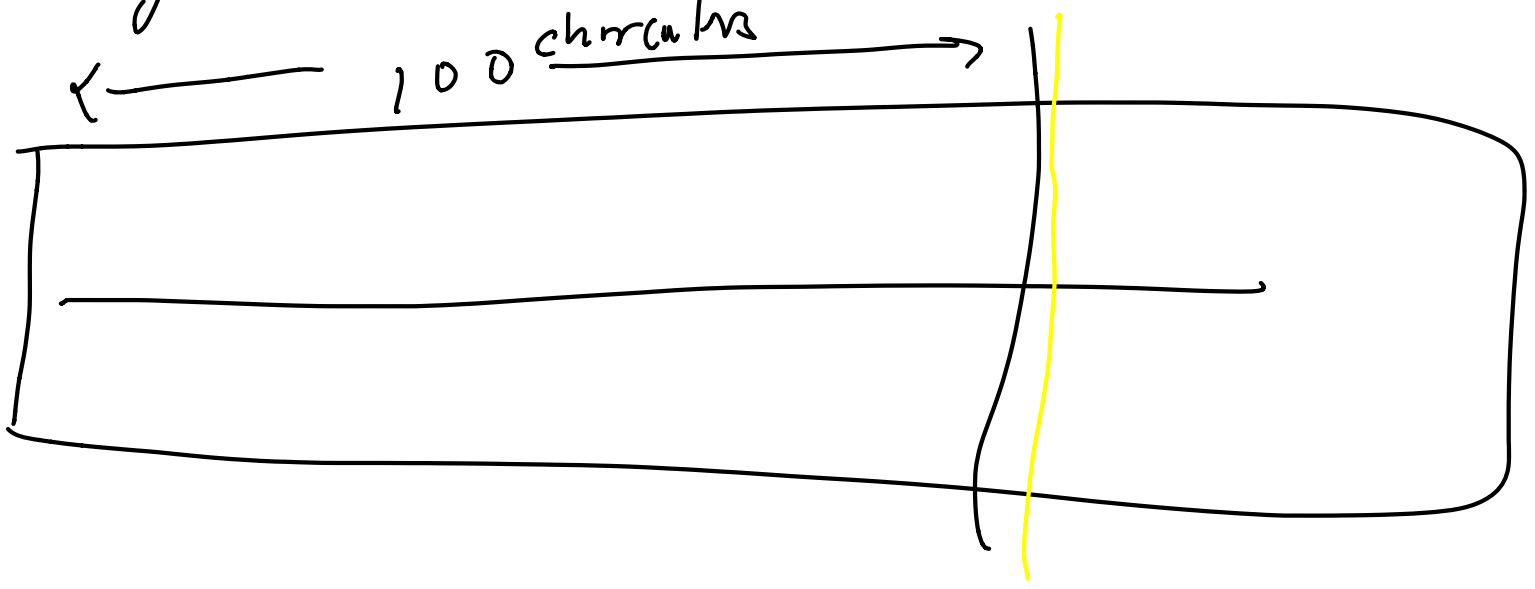
Error-message display line number

Dr Racket
On Top menu
Edit → Preferences → Editing
→ General Editing
☑ Show line numbers

Line number will be displayed

80 : 100
↑ ← col:
Line number

☑ Maximum Character width guide 100



— Type-Checker errors

Unions are prone to Type-checker errors.

Homework 3
Problem 2a.

(a) (: clothing-color : Clothing → Color)

```
(define (clothing-color c)
  (cond
   [(Shirt? c) (Shirt-color c)]
   [else (Pants-color c)]))
```

Can anything go wrong with the above code?

Pants-color expects a structure of type Pants

Type checker will try to deduce the type of c.

Type-checker will see c is clothing.
You get a type checker error.

Why there is no problem in the first line
In the first line, type checker will deduce c is Shirt variable. because we are explicitly checking.

How to preven this

```
(define (clothing-color c)
 (cond
   [(Shirt? c) (Shirt-color c)]
   [(Pants? c) (Pants-color c)]
```

[else (error "Invalid")]).

The otherway is to use match which we will discuss today.

Pattern matching

```
(match exp0
  [pat1 exp1]
  [pat2 exp2]
    ⋮
  [pat n expk])
```

Similar to cond.

```
(match (+ 1 1)
 [1 "one plus one is one"]
 [2 "one plus one is two"]
 [3 "one plus one is three"])
```

⇒ "one plus one is two"

↑ output

```
(: is-it-5 : Integer → String)

(define (is-it-5 n)
 (match n
  [5 "It is 5!!"]
  [_ "Not 5..."]))
```

_ is the wild cart characte.
It matches with anything.

(is-it-5 5) ⇒ "It's 5!!"

(is-it-5 10) ⇒ "Not 5..."

-

Variable matching

```
(match (f 2 3)
  [0  0]
  [n  (* n 2)])
```

(f 2 3) ⇒ 5.

5 is matched against 0. false.

5 is matched againts n variable
Important point: variable
matches with anything.
n will be become 5.

The output will be 10.

```
(match (+ 1 1)
  [3  "A"]
  [5  "B"]
  [2  "D"]
  [5  "E"]
  [2  "F"]) ⟹ "D"
```

```
(match  2
 [3   "A"]
 [5   "B"]
 [2   "D"]
 [5   "E"]
 [2   "F"] ⇒ "D"
```

→ Don't put braces around 2.

(2) ⇒ incorrect.

Variable Scope

```
(: x : Shiny)
(define x "I live outside")

(: func : Shiny -> Shiny)
(define (func x)
  (append x " and thank you"))
```

(func "please") ⇒
"please and thank you"

There are two variables
x but they are different.
Inside func x will refer to
"please".
But outside func x will
refer to "I live outside".

Example

(: n : Integer)
(define n 10) → Variable
Definition.
(define (f input)
→ This is fo

functions.

```
(match 5
  [6  11]
  [n  (* n n)]
  [5  6])
```

What will be the output.

The output will be 25.

not 6

The variable n is local to match is not related to the variable n outside.

Match structures

```
(define-struct Point
  ([x : Real]
   [y : Real]))
```

Write a function to add 1 to both x and y coordinate of a point.

```
(: add-one : Point → Point)
(define (add-one p)
  )
```

```
(match p

  [(Point x y) (Point (+ x 1) (+ y 1))
  ])
```

(add-one (Point 3 5))
⇒ 46.

P1 has x and y

x and y of P1 is getting copied to. x and y inside match.
Then create a new point
(Point (+ x 1) (+ y 1))

HW4 : Problem 1 Pine-Cones and Pine trees
Above is useful.

Area example revisited

```
(define-struct Circle
  ([radius : Real]
   [center : Point]
   [color : Color]))

(define-struct Square
  ([side-length : Real]
   [centre : Point]
   [color : Color]))

(define-type Shape (U Circle Square))
```

```
(: area : Shape → Real)

(define (area shape)
  (match shape
[(Circle r _ _) (* pi r r)]
[(Square s _ _) (* s s)]))
```

I can't write this

(Circle r) ⟹ illegal

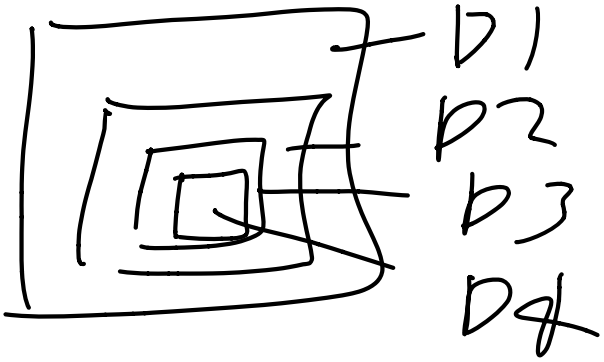
Recursive Types

Russians Dolls

Dolls containing Dolls.

```
(define-struct Doll
 ([inches-tall : Exact-Rational]
  [color : (U 'blue 'red 'yellow)]
  [inside : (U 'nothing Doll)]))
```

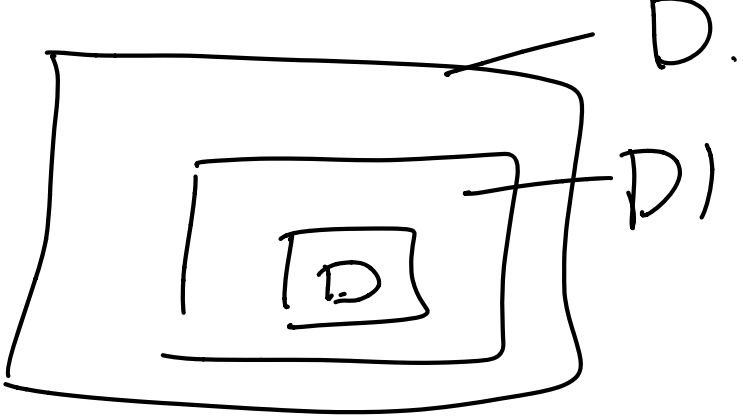
Write a program that takes a Doll as input outputs total number of Dolls inside including itself.



Side Remark:
Color is a user defined data type.

If you want use Color as a type you have to define it.

Idea (Use recusion).
Let D be a Doll.
let D1 be the first doll inside D.



Recursively count for D1.

Total number of Dolls for D
= 1 + Total number of Dolls
for D1.

What is the Base case?

If D contains nothing
then output 1.

Style 1

```
(: count-dolls : Doll → Integer)

(define (count-dolls d)
  (match (Doll-inside d)
    [(Doll _ _ _) (+ 1 (count-dolls
                        (Doll-inside d)))]
    ['nothing 1]))
```

This will get me the the Doll inside d

→ Base Case

Style 2

```
(: count-dolls : (U 'nothing Doll)
                 → Integer)
(define (count-dolls d)
  (match d
    ['nothing 0]
```

```
[(Doll _ _ sd) (+ 1 (count-dolls sd))]
))
```

sd is a variable name.

The Doll inside d (inside field of d) will be stored in sd).

Ex1

```
(check-expect (count-dolls (Doll
  1 'blue 'nothing)) 1)
```

Ex2

```
(check-expect (count-dolls (Doll 1
  'blue (Doll (/ 1 2) 'red 'nothing)))
    2)
```

Mutual Recursion

Example of Hofstadter

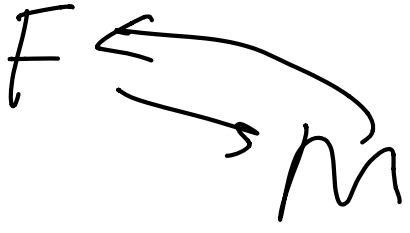
Female and Male sequence

(function) (in a mathematic sense)

$$F(0) = 1 \qquad M(0) = 0$$

$$F(n) = n - M(F(n-1)), \quad n > 0$$

$$M(n) = n - F(M(n-1)), \quad n > 0$$



	0	
F:	1	1
M:	0	0

$$F(1) = 1 - M(F(0))$$

$$= 1 - M(1) = 1.$$

$$M(1) = 1 - F(M(0))$$

$$= 1 - F(0)$$

$$= 0$$

$F(2)$

$M(2)$

This is an example

of mutual recursion

Racket function for this sequence

```
(: M : Integer -> Integer)

(define (M n)
  (cond
    [(= n 0) 0]
    [else (- n (F (M (- n 1))))]))

(define (F n)
  (cond
    [(= n 0) 1]
    [else (- n (M (F (- n 1))))]))
```

(F 3) ⇒ 2

(M 4) ⇒ 2

HW 4 pines-cones and pine-bres!

User-defined Natural Numbers

Remark: This is different from Racket's implementation of natural numbers.

What are Natural Numbers?

We define zero

We say one is successor of zero.

σ: successor.

$\sigma(0) = 1$.

$\sigma(1) = 2 = \sigma(\sigma(0))$

$\sigma(3) = \sigma(2) = \sigma(\sigma(\sigma(0)))$

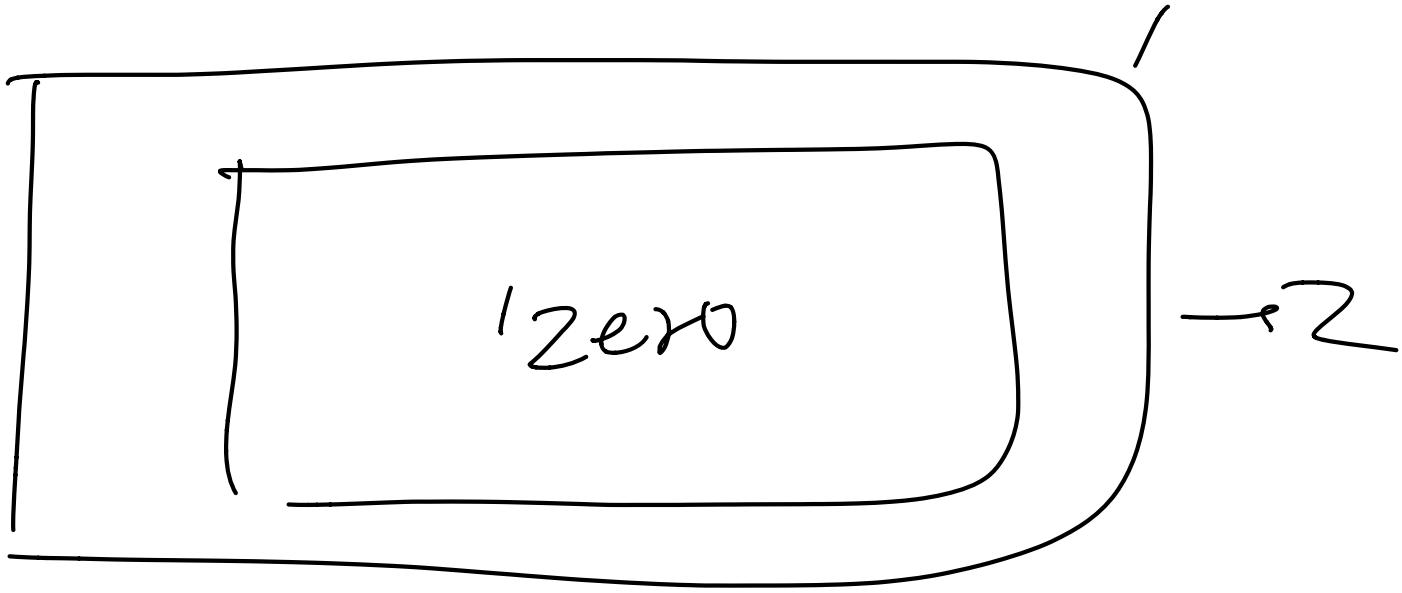
```
(define-struct Succ
  ([nat : Nat]))

(define-type Nat (U 'zero Succ))
```

'zero ⇒ 0

(Succ 'zero) ⇒ 1 ⇒ ≠ output

(Succ (Succ 'zero)) ⇒ 2



Write a function to check if a Nut represents zero.

```
(: zero? : Nat → Boolean)
(match nat
  ['zero  #t]
  [_      #f])
```

Write a function to subtract one from a Nat. (Input: n Output: n-1)

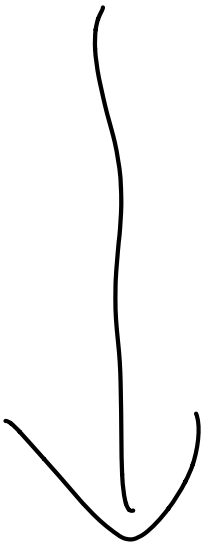
```
(: pred : Nat → Nat)
(define (pred n)
  (match n
    ['zero (error: "pred: not defined")]
    [(Succ m) m]))
```

Write a function add 1 to a Nat.

$$n1 + n2$$
$$= (n1 + 1) + (n2 - 1)$$

(add n1 n2) ≡
(add (add-one n1) (pred n2))

if n2 = 0
n1 + n2 = n1)



Add two natural numbers

Suppose you have written the code add
Want to check = 1+2 = 3

```
(check-expect (add (Succ 'zero)
                   (Succ (Succ 'zero)))
              (Succ (Succ (Succ 'zero))))
```

(Succ 'zero) represents 1

(Succ (Succ 'zero)) represents 2

(Succ (Succ (Succ 'zero))) represents 3.