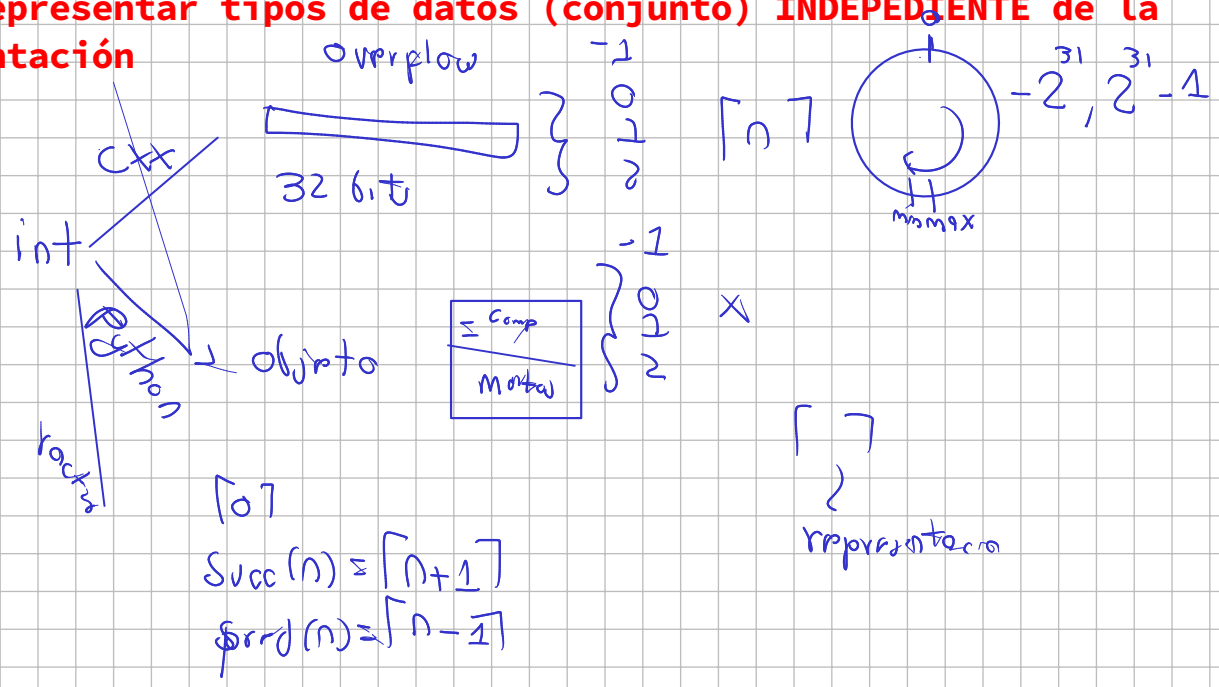
Representación de datos

TAD: Tipo abstracto de datos

Poder representar tipos de datos (conjunto) INDEPEDIENTE de la implementación



Receta TAD

- **Interfaz (es lo que interactua el programador)**
- **Implementación (debajo, no lo ve el programador)**

Interfaz

- **Constructores**
- **Observadores: Predicados y extractores**

- **Recursiva (succ y pred), Listas de #T, (cons q l)**
- **Listas y procedimientos**

Receta para la construcción de TADs

- **Implemente un constructor para cada variante en la gramatica**
- **Implemente un predicado para cada variante en la gramática**
- **Implemente un extractor para cada parte de cada variante en la gramática**

```
<arbol> ::=   '()
              arbol-vacio()
          ::= <int> <arbol> <arbol>
              arbol-int(k, hiz, hder)
          ::= <symbol> <arbol> <arbol>
              arbol-symbol(k, hizq, hder)
```

Bignum

Base

(list an an-1 ... a1 a0)

lsb

msb

16

( )

16^0 16^1 16^2 ....

(3 5 8)

16^0 16^1 16^2

$3 + 5\times16 + 8\times16^2$

(4 5 8)

(1 1 1)

(0 0 0 1)

```
(define plus
  (lambda (x y)
    (if
     (zero? y)
     x
     (succ (plus x (pred y))))))
```

plus pred

```
(plus 5 3)

(succ (plus 5 2))
(succ (succ (plus 5 1)))
(succ (succ (succ (plus 5 0))))
(succ (succ (succ 5))
```

```
5 + 3
succ (5)
succ (5)
succ (5)

(succ 5) + 2
(succ (succ 5)) + 1
(succ (succ (succ 5)))
```

5×3

5+5+5