Estrategias para representar datos

- Implementación: Se refiere como se guardan los datos en el computador

int: cadena 32 bits
string: array de cadenas de 8 bits (ASCII extendido)
doudle: cadena de 64 bits
¿Donde la damos el significado?

- Interfaz: Es lo que ve el programador, operaciones y formas de representación de los datos, int: 1 2 3 ...

TAD: Tipo abstracto de dato: Soporta el cambio de implementación (el programador no se da cuenta de ello)

Cuando usted diseña tipos de datos (TAD) con este esquema LAS FUNCIONES DEL PUNTO DE VISTA DEL PROGRAMDOR NO CAMBIAN.

1) Define los constructores
2) Define los observadores
   - Definir predicados para cada variante del tipo de dato
   - Definir extractores por cada parte del tipo de dato

Dos ejemplos: lista y otro con los ambientes.

```
<enviroment> ::=  '()
                  empty-env()
                ::= <list id> <list val> <enviroment>
                  extend-env(lid lval env)

 (extend-env
            '(x y z)
            '(1 2 3)
            (extend-env
                  '(a b c)
                  '(4 5 6)
                  (empty-env)
            )
 )
```

Sintaxis concreta o representación externa

```
def funcion():
  ...
 return x

public static void main(String args[])
        ....
```

Es lo que escribe el programador

Sintaxis abstracta, representación interna

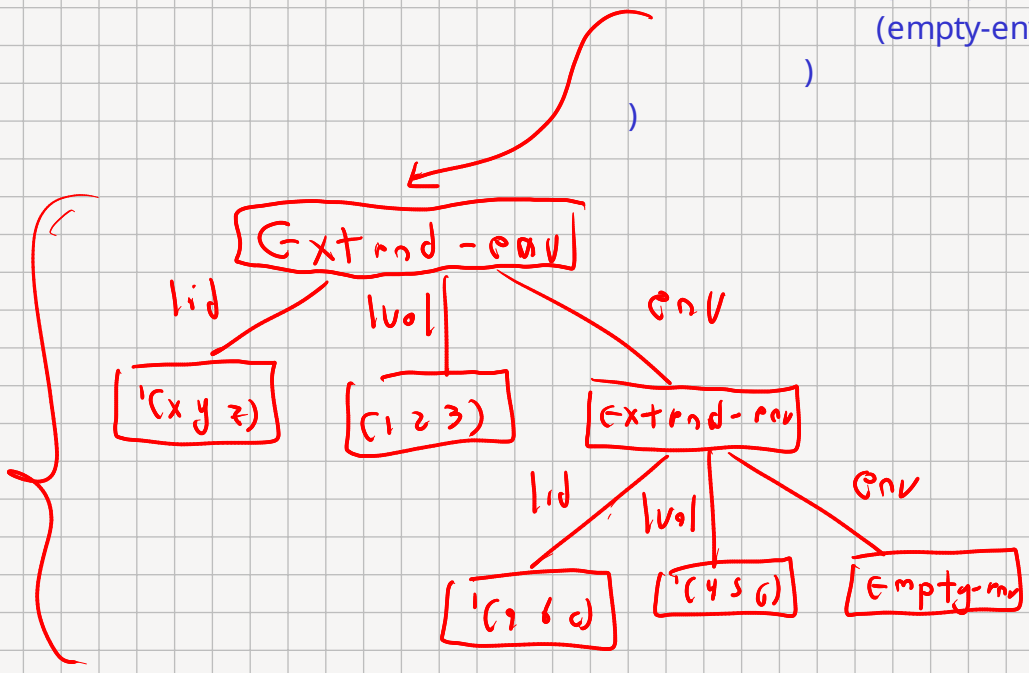
Lo que el lenguaje de programación representa.

Permite hacer optimizaciones de código.

Arboles de sintaxis abstracta: Relacionan la gramática con el código escrito, permiten revisar si algo esta BIEN ESCRITO, porque si esta mal escrito, NO PODEMOS GENERAR EL ARBOL

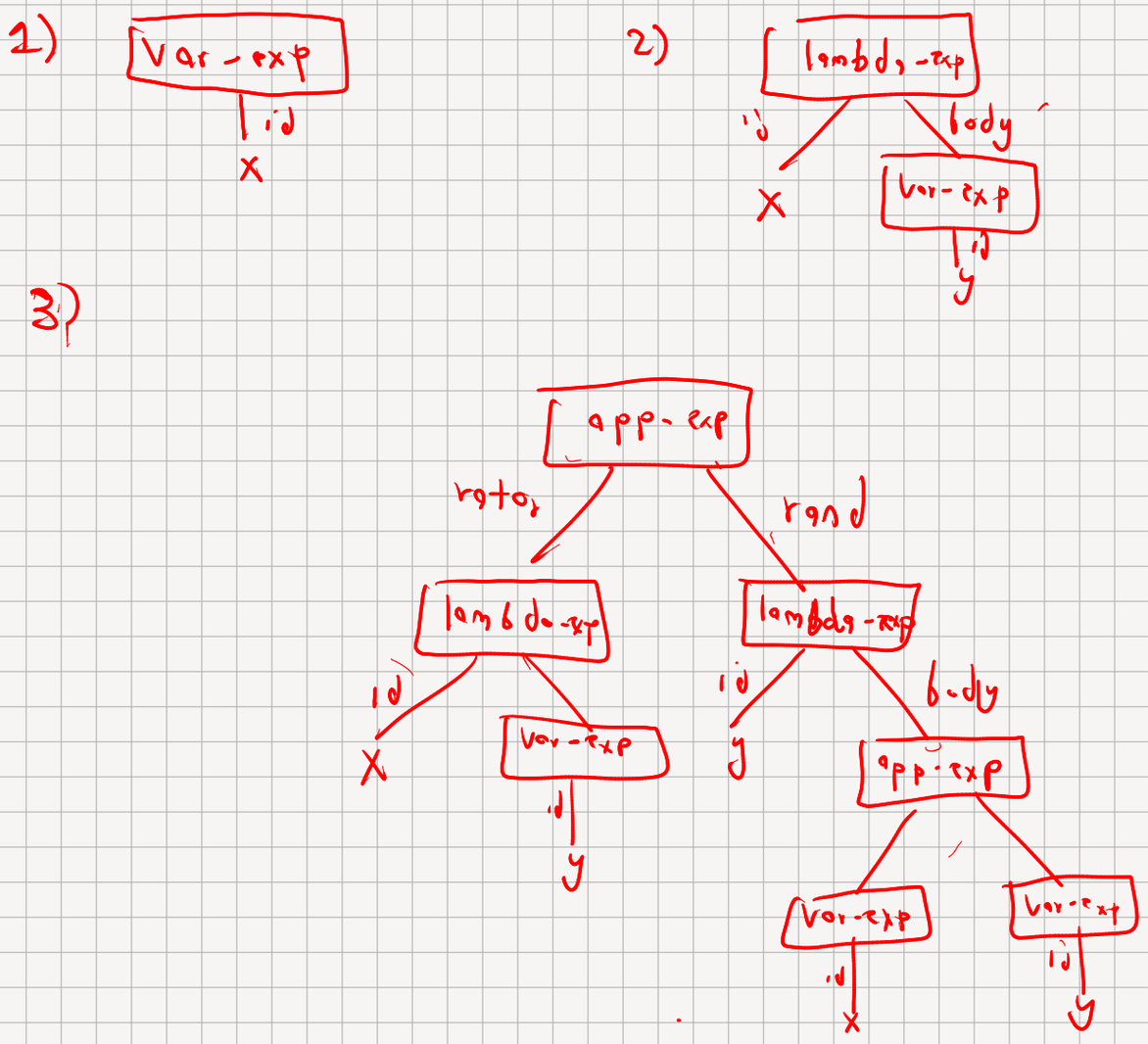
```
<enviroment> ::=  '()
                  empty-env()
                ::= <list id> <list val> <enviroment>
                  extend-env(lid lval env)
```

```
(extend-env
        '(x y z)
        '(1 2 3)
        (extend-env
                '(a b c)
                '(4 5 6)
                (empty-env)
        )
)
```




```
<lc-exp> ::= <identifier>
                var-exp(id)
             ::= "lambda" <identifier> <lc-exp>
                lambda-exp(id body)
             ::= <lc-exp> <lc-exp>
                app-exp(rator rand)
```

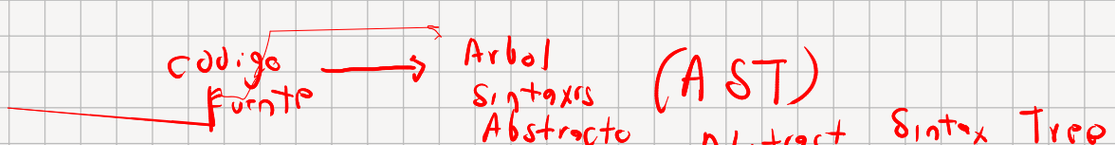
```
1) 'x
2) lambda (x) y
3) (
        (lambda (x) y)
        (lambda (p) (x y))
   )
```




```
(define-datatype <nombre-tipo> <nombre-predicado>
   (<nombre-variante> (<nombre-campo> <tipo-campo)*)+
)
```

Parser: Convertir de sintaxis concreta a sintaxis abstracta

Unparser: Convertir de sintaxis abstracta a sintaxis concreta




```
<lc-exp> ::= <identifier>
               var-exp(id)
         ::= "lambda" <identifier> <lc-exp>
             lambda-exp(id body)
         ::= <lc-exp> <lc-exp>
             app-exp(rator rand)
```