



# CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

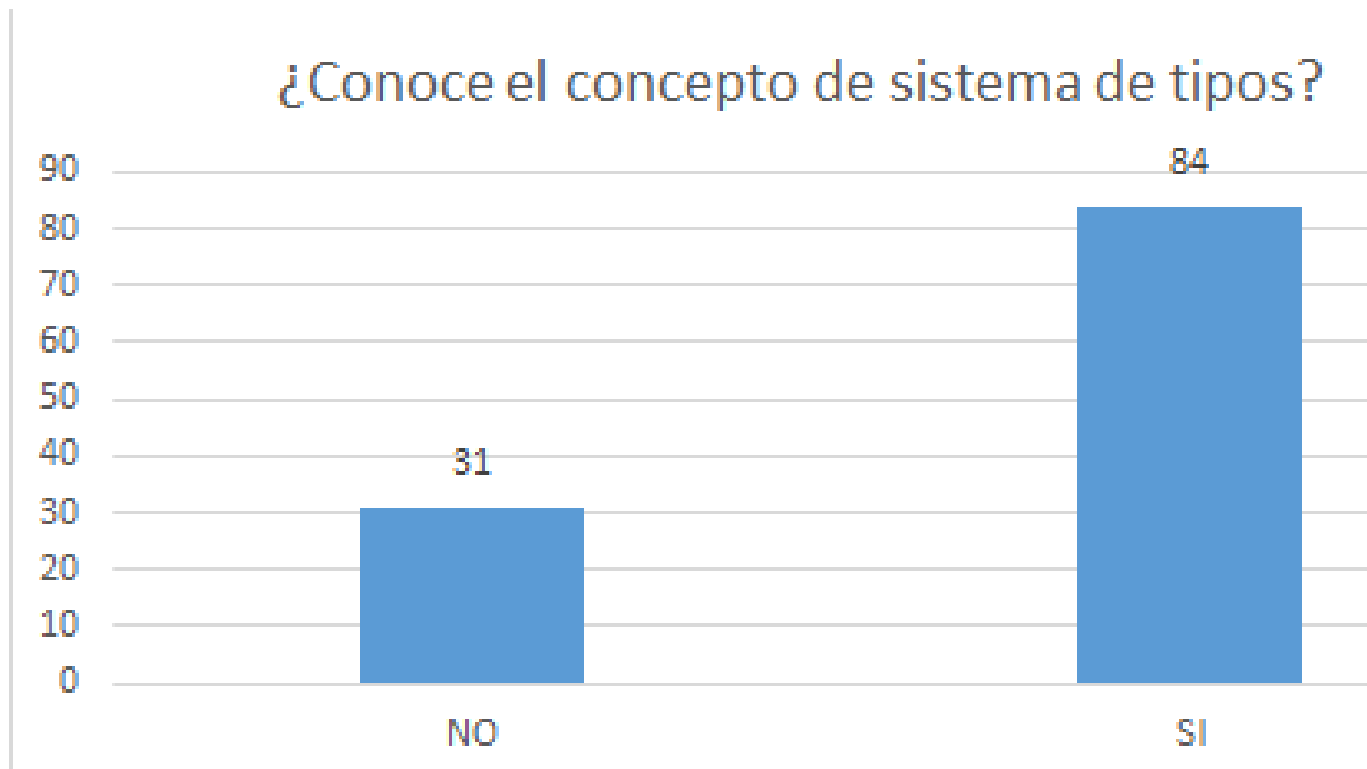
## SISTEMAS DE TIPOS

# SISTEMA DE TIPOS - INTRODUCCIÓN

- ✓ Representa la abstracción de datos en los lenguajes de programación
- ✓ Trata con las componentes de un programa que son sujeto de computación. Esta basado en las **propiedades** de los objetos de datos y las **operaciones** de dichos objetos.
  - Tipos de datos
  - Encapsulamiento y abstracción



# ENCUESTA REALIZADA AL COMIENZO DE LAS CLASES



# SISTEMA DE TIPOS

**Establece el tipo para cada valor manipulado.**

- Provee **mecanismos** de expresión:
  - Expresar tipos intrínsecos o definir tipos nuevos.
  - Asociar los tipos definidos con construcciones del lenguaje.
- Define **reglas** de resolución:
  - *Equivalencia de tipos* – ¿dos valores tienen el mismo tipo?.
  - *Compatibilidad de tipos* – ¿puedo usar el tipo en este contexto?
  - *Inferencia de tipos* – ¿cuál tipo se deduce del contexto?
- Cuanto más flexible el lenguaje, más complejo será el sistema

100100111011



## SISTEMA DE TIPOS

- Conjunto de reglas que estructuran y organizan una colección de tipos.
- El objetivo del sistema de tipos es lograr que los programas sean tan seguros como sea posible.
  - Seguridad Vs Flexibilidad



# SISTEMA DE TIPOS

## TIPADO FUERTE O TIPADO DÉBIL

- Se dice que el **sistema de tipos es fuerte** cuando especifica **restricciones sobre como las operaciones que involucran valores de diferentes tipos pueden operarse.**
- Lo contrario establece un **sistema débil de tipos**

```
a = 2
b= "2"
Concatenar (a,b) //error de tipos
Sumar (a,b) //error de tipos
Concatenar (str(a),b) //retorna "22"
Sumar (a,int(b)) //retorna 4
```

```
a = 2
b= "2"
Concatenar (a,b) //retorna "22"
Sumar (a,b) //retorna 4
```

# ESPECIFICACIÓN DE UN SISTEMA DE TIPOS

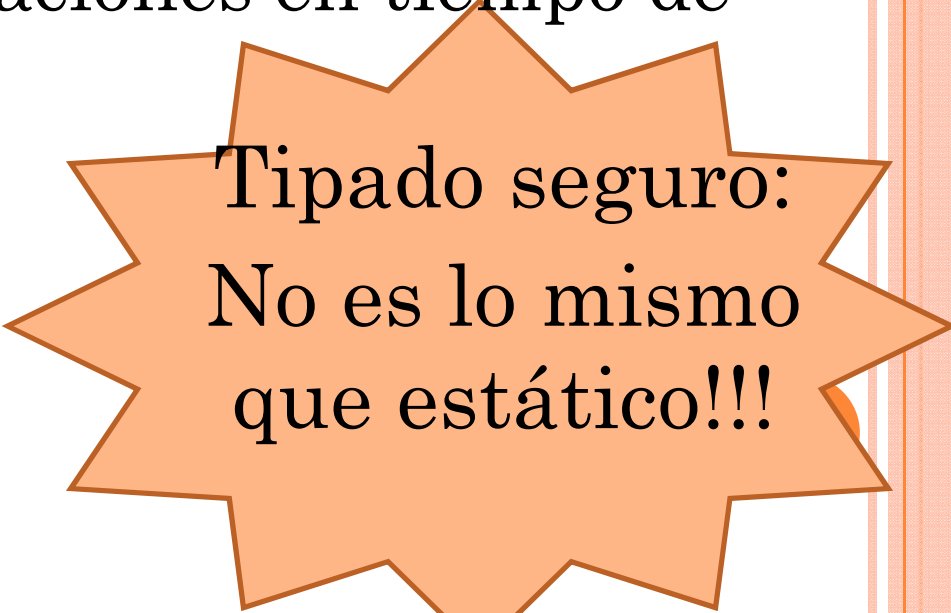
- Tipo y tiempo de chequeo
- Reglas de equivalencia y conversión
- Reglas de inferencia de tipo
- Nivel de polimorfismo del lenguaje



# SISTEMA DE TIPOS - ESPECIFICACIONES

## ○ Tipos de ligadura

- **Tipado estático:** ligaduras en compilación
  - Java, C
- **Tipado dinámico:** ligaduras en ejecución, provoca mas comprobaciones en tiempo de ejecución
  - Php, Phyton, Ruby,



Tipado seguro:  
No es lo mismo  
que estático!!!



# SISTEMA DE TIPOS - ESPECIFICACIONES

## ○ Tiempo de ligadura

- El tipado es **estático** si cada **entidad/variable** queda ligada a su tipo durante la **compilación**, sin necesidad de ejecutar el programa
- El tipado es dinámico si la ligadura de la variable/entidad se produce en tiempo de ejecución



# LENGUAJES FUERTEMENTE TIPADOS

## ENCUESTA DE INICIO DEL CURSO

“Un lenguaje fuertemente tipado es un lenguaje al que por ejemplo hay que **declarar** una variable con su **tipo** y esa variable siempre va a contener datos de ese tipo, **no puede cambiar**.”

87%

“Un lenguaje en el que una vez que se le asigna un tipo de dato a una variable, esta no puede cambiar de tipo durante la **ejecución del programa**.”

“Un lenguaje es fuertemente tipado cuando la declaracion y **conversiones** de tipo son explicitas, impidiendo así la violacion de tipos”

“Reconocer mayusculas y minúsculas”

`“int X;”`

“un lenguaje de programacion fuertemente tipado es aquel que no permite **variaciones** en el tipo de datos que aloja una variable una vez declarada. ”

# LENGUAJES FUERTEMENTE TIPADOS

## Definición:

- Un lenguaje se dice **fuertemente tipado** (**type safety**) si el sistema de tipos impone restricciones que aseguran que no se producirán **errores de tipo en ejecución**

En esta concepción, la intención es evitar los errores de **aplicación** y son tolerados los errores del **lenguaje** (sintácticos o semánticos), detectados tan pronto como sea

Python es fuertemente tipado y  
tiene tipado dinámico



# LENGUAJES FUERTEMENTE TIPADOS

Algunas definiciones que se pueden encontrar:

- Si el lenguaje es fuertemente tipado el compilador puede garantizar la **ausencia de errores de tipo en los programas** (Ghezzi), actualmente esta definición es incompleta
- Un lenguaje se dice **fuertemente tipado (type safety)** si todos los errores de tipo se **detectan**. El problema de esta definición es que no indica en que momento se detectan



# ¿QUÉ ES UN TIPO DE DATO?

- Desde el punto de vista **Denotacional** es
  - Conjunto de valores sobre un *dominio*.
- Desde el punto de vista **Constructivo** puede ser
  - *Primitivo* (built-in o predefinido) provisto por el lenguaje.
  - *Compuesto* (composite o derivado) empleando constructores de tipos.
- Desde el punto de vista de la **Abstracción**
  - Una *interfaz a una representación*.
  - Conjunto de operaciones con semántica bien definida y consistente.

**Al programar los percibimos como  
una mezcla de los tres.**

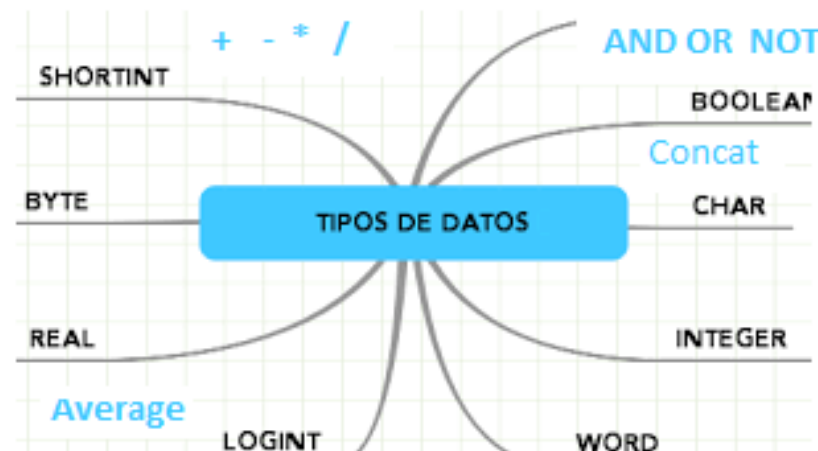


# TIPOS DE DATOS

**Dominio + Abstracción**

**Valores Operaciones**

Los tipos de datos “Capturan la naturaleza de los datos del problema que serán manipulados por los programas”



# TIPOS DE DATOS

## Objeto

(l-valor, r-valor)



## Instancia del tipo

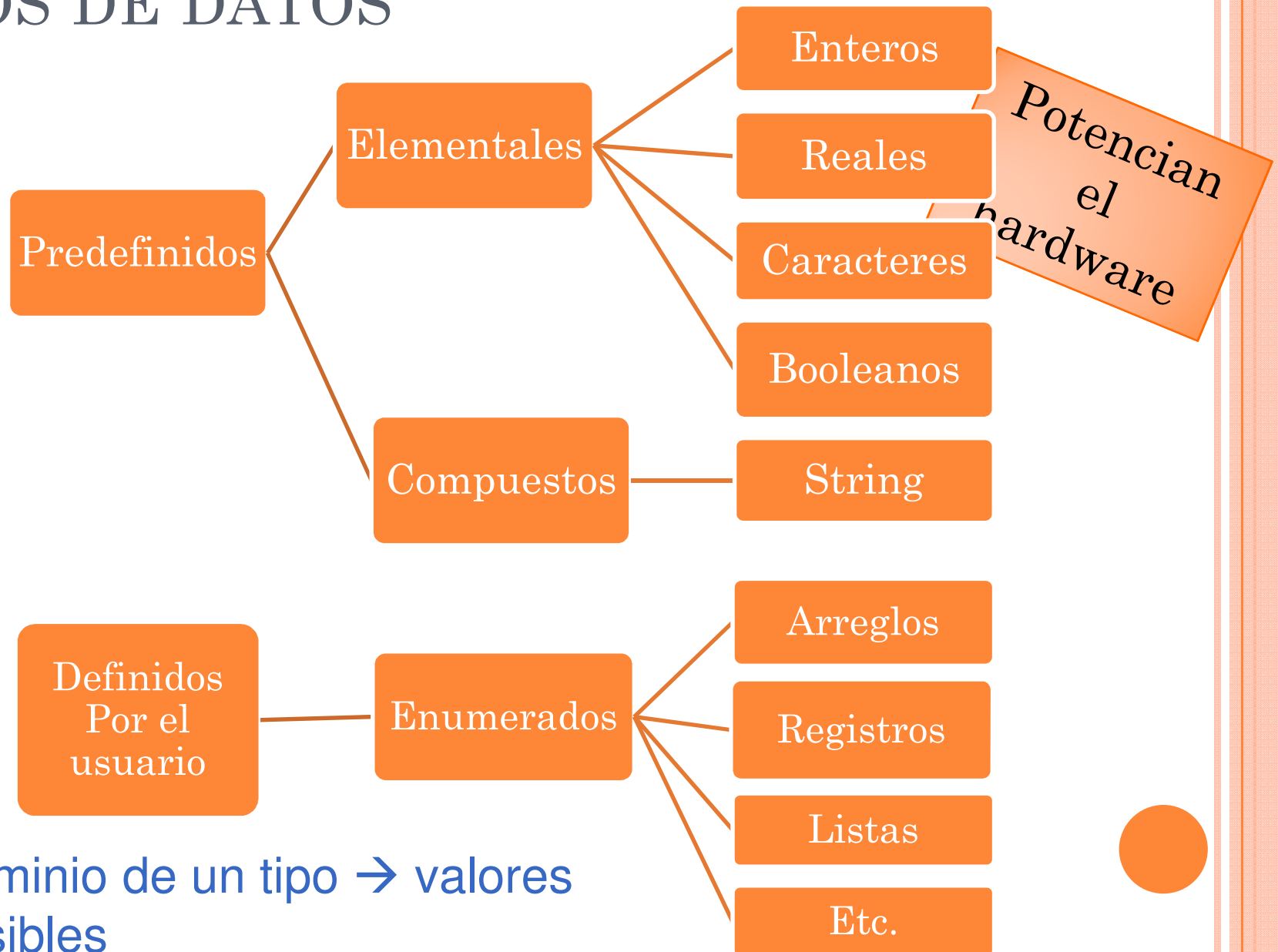


- **Operaciones:** única forma de manipular los objetos instanciados

- **Tipo:** comportamiento abstracto de un conjunto de objetos y un conjunto de operaciones.



# TIPOS DE DATOS



Dominio de un tipo → valores posibles



# TIPOS DE DATOS

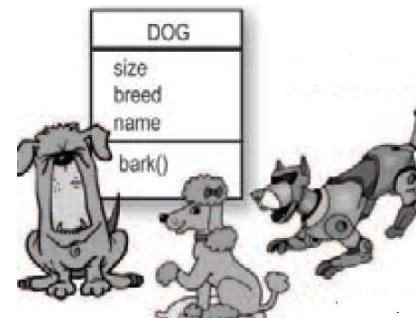
## ○ Nueva estructura → Tipo

- Arreglos, registros, listas

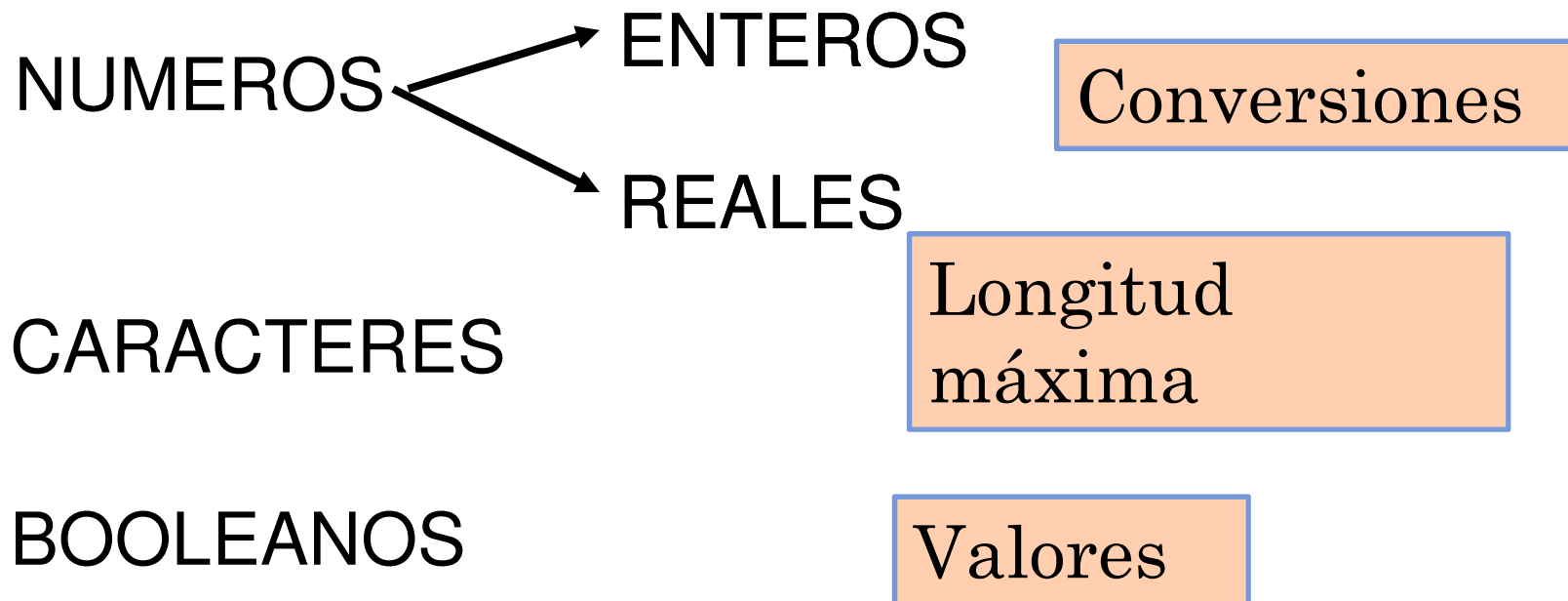
```
a = array([6, 1, 3, 9, 8])
```

## ○ Nuevo comportamiento → TAD

- Clases, paquetes, unidades



# TIPOS PREDEFINIDOS ELEMENTALES



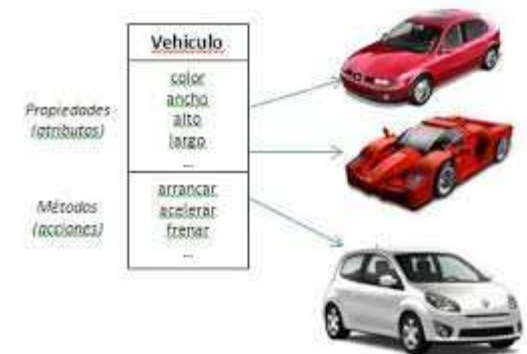
- Ventajas de los tipos predefinidos:
  - Invisibilidad de la representación
  - Verificación estática
  - Desambiguar operadores
  - Control de precisión



## TIPOS DEFINIDOS POR EL USUARIO

Separan la **especificación** de la **implementación**. Se definen los tipos que el problema necesita.

- Definir nuevos tipos e instanciarlos
- Chequeo de **consistencia**



## TIPOS DEFINIDOS POR EL USUARIO

- **Legibilidad** : elección apropiada de nuevos nombres

*dias [0..31]*  
*vec [dias]*

*vec[0..31]*

- Estructura **jerárquica** de las definiciones de tipos: proceso de refinamiento

```
class Persona {  
    String nombre,apellido;  
    int edad;  
    .....  
    Persona vecino= new Persona(...);  
}  
vecinos = array[1..10] of Persona;
```



# TIPOS DEFINIDOS POR EL USUARIO

- Modificabilidad : Solo se cambia en la definición
- Factorización: Se usa la cantidad de veces necesarias

La instanciación de los objetos en un tipo dado implica una descripción abstracta de sus valores. Los detalles de la implementación solo quedan en la definición del tipo



# TIPOS DEFINIDOS POR EL USUARIO - ENUMERATIVOS

**DOMINIO:** lista de constantes simbólicas

**OPERACIONES:** comparación, asignación y posición en la lista

*type MES is (ENERO, FEBRERO, ....., DICIEMBRE)*

**potencian** la expresividad del lenguaje.

Se define el nuevo tipo enumerado

Se instancia el tipo

*X: MES*

**noción de orden** (predecesor y sucesor)

relaciones de  $>$ ,  $<$ ,  $\leq$ ,  $\geq$ ,  $=$ .

**Ambigüedad:.**

*X= ENERO*

*MES' ORD(ENERO)*

*VERANO' ORD(ENERO)*

*type VERANO is  
(ENERO, FEBRERO, MARZO)  
Y: VERANO*

## ENUMERADOS - SUBRANGOS

**Dominio:** subconjunto de un tipo entero o de un enumerado

**Operaciones:** hereda las operaciones del tipo original.

```
type verano = ENERO .. MARZO
```

La implementación es la que determina si pueden mezclarse variables de tipo verano y mes

### Chequeo dinámico

```
type subind [1..10]  
a,b,c: subind  
a:=b+c
```

El resultado sera  
del tipo subind??



# TIPOS DEFINIDOS POR EL USUARIO

## TIPOS COMPUESTOS

Nuevos tipos definidos por el usuario  
usando los constructores

**Constructores:** mecanismo para agrupar datos denominados compuestos

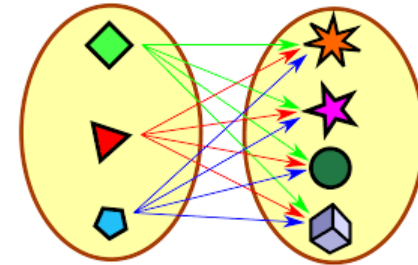
- **Dato compuesto:** posee nombre único.  
Accesible a través de un mecanismo de selección.  
Posibilidad de manipular conjunto completo.
- **Rutinas:** constructores que permiten combinar instrucciones elementales para formar un nuevo operador.

COMPARACION



# TIPOS DEFINIDOS POR EL USUARIO COMPUESTOS

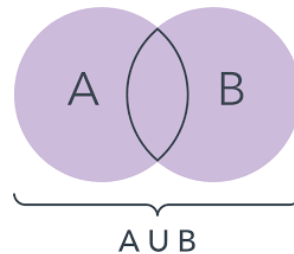
- Producto Cartesiano



- Correspondencia Finita



- Union



- Recursión



# PRODUCTO CARTESIANO

## ○ C: estructuras

```
typedef struct {  
    int nro_lados;  
    float tamaño_lado;  
} reg_poligon
```

```
reg_poligon pol = {3, 3.45}
```

```
pol.nro_lados = 4
```

definición

instancia con valor  
compuesto inicial

operaciones  
de campo

acceso

operaciones  
de tipo

# CORRESPONDENCIA FINITA

correspondencia finita en general

$$f: DT \longrightarrow RT$$

Si DT es un subrango de enteros

$$f: [li..ls] \longrightarrow RT$$

**conjunto de valores accesibles via un subinidice**

## COMPARACION

## ARREGLOS

•**Rutina:** su definición es la regla de asociación de valores del tipo DT en valores del tipo RT.

**definición intencional:** que especifica una regla (la intención) en lugar de una asociación individual

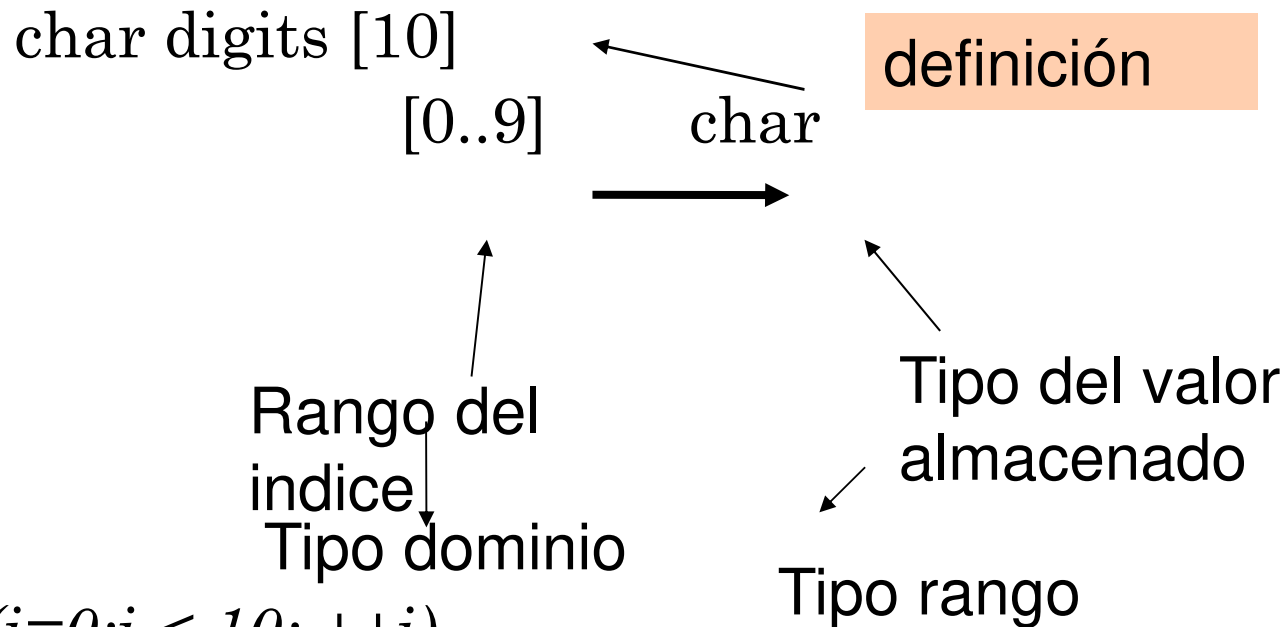
$$F(x) = 2x$$

•**Arreglos:** definición **extensional**, los valores de la función son explícitamente enumerados



# CORRESPONDENCIA FINITA

- C



- *for* (*i=0; i < 10; ++i*)

*digits[i] = ";*

acceso

*aplicacion de la correspondencia para i*

**Phyton:** lis = ["una lista", [1, 2]]

mi\_var = lis[1][0] # mi\_var vale 1



# LÍMITES DE CADA ÍNDICE

Como ligar el dominio un subconjunto específico de un tipo dado?

- **Ligadura en compilación:** `vec[0..9] of integer`  
el subconjunto se fija cuando el programa se escribe  
Pascal - C

ESTATICA

- **Ligadura en la creación del objeto:**  
el subconjunto se fija en ejecución,  
cuando se crea la instancia del objeto.(arreglos  
semidinámicos)

`real A(N..M)`

ADA

DINAMICAS

- **Ligadura en la manipulación del objeto**  
es la mas flexible y costosa en términos de tiempo de ejecución.

Arreglos flexibles: los que el tamaño del subconjunto puede variar durante la vida del objeto.

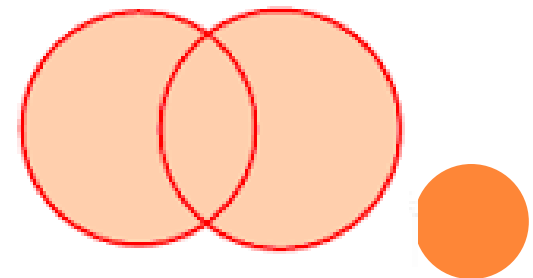
APL - Snobol4 - C++ - phyton

heap

# UNION Y UNION DISCRIMINADA

La unión / union discriminada de dos o mas tipos define un tipo como la disjunción de los tipos dados

- Permite manipular diferentes tipos en distinto momento de la ejecución
- Chequeo dinámico



# UNION Y UNION DISCRIMINADA

```
struct tipoRevista {  
    int codigo;  
    char nombre[32];  
    int mes;  
    int anio;  
}
```

```
struct tipoLibro {  
    int codigo;  
    char autor[80];  
    char titulo[80];  
    char editorial[32];  
    int anno;  
};
```

UNION  
En C

```
union tipoEjemplar {  
    tipoLibro l;  
    tipoRevista r;  
};  
  
tipoEjemplar tabla[100];
```

Mututamente  
excluyentes

¿cómo sabemos si  
tipoEjemplar  
contiene un libro o  
una revista ?

UNION  
DISCRIMINADA



# UNION DISCRIMINADA

**Unión discriminada** agrega un **discriminante** para indicar la opción elegida.

Si tenemos la unión discriminada de dos conjuntos S y T, y aplicamos el discriminante a un elemento e perteneciente a la unión discriminada devolverá S o T.

- El elemento e (tipoEjemplar) debe manipularse de acuerdo al valor del discriminante.
- En la unión y en la unión discriminada el chequeo de tipos debe hacerse en ejecución.
- La unión discriminada se puede manejar en forma segura consultando el discriminante antes de utilizar el valor del elemento



# UNION DISCRIMINADA

UNION  
Discriminada  
En C

```
enum eEjemplar { libro, revista};

struct tipoEjemplar {
    eEjemplar tipo;
    union {
        tipoLibro l;
        tipoRevista r;
    };
};
```

En manos del  
Programador

INSEGURO



# UNION DISCRIMINADA

## Problemas:

- El discriminante y las variantes pueden manejarse independientemente uno de otros.
- La implementación del lenguaje ignora los chequeos
- Puede omitirse el discriminante, con lo cual aunque se quisiera no se puede chequear



# RECURSIÓN

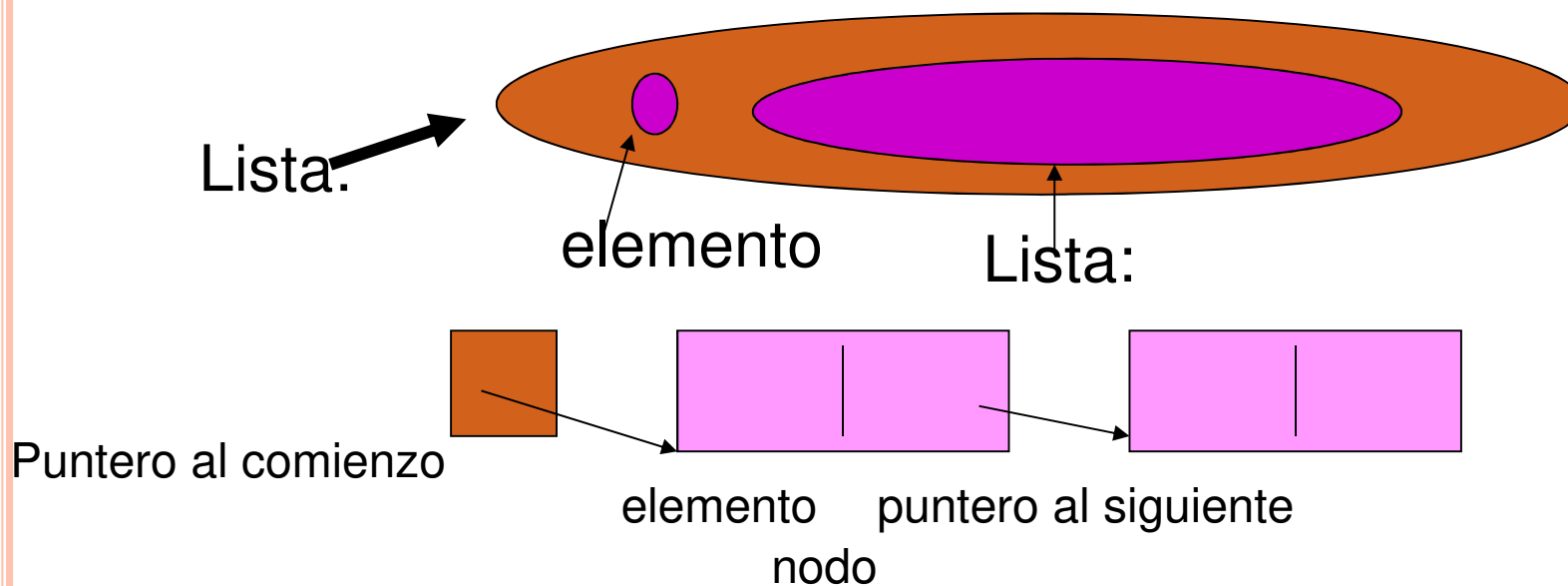
Un tipo de dato recursivo  $T$  se define como una estructura que puede contener componentes del tipo  $T$ .

- Define datos agrupados:
  - cuyo tamaño puede crecer arbitrariamente
  - cuya estructura puede ser arbitrariamente compleja.



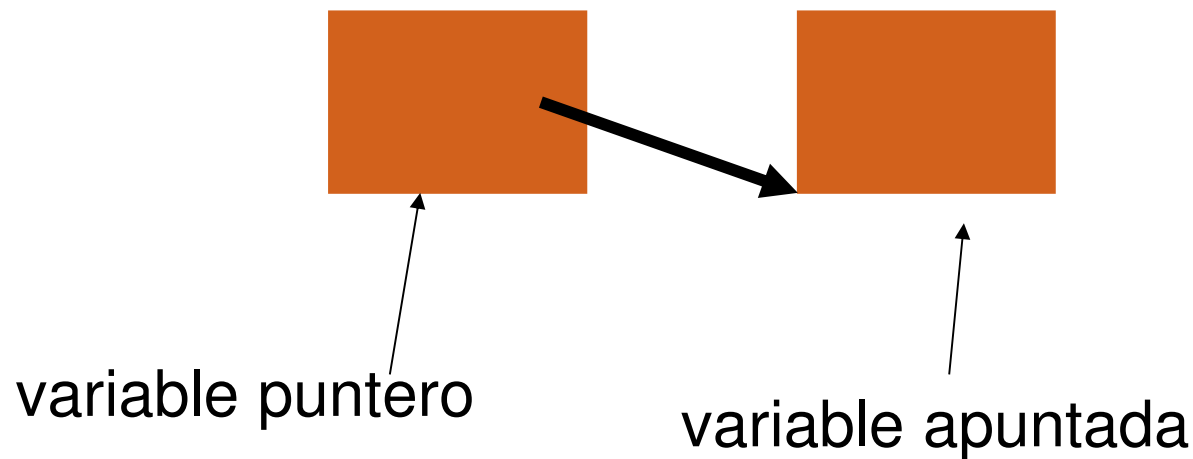
# RECURSIÓN - IMPLEMENTACIÓN

- Los lenguajes de programación convencionales soportan la implementación de los tipos de datos recursivos a través de los **punteros**.
- Los lenguajes funcionales proveen mecanismos mas abstractos que enmascaran a los punteros



# PUNTEROS

- Un **puntero** es una referencia a un objeto.
- Una **variable puntero** es una variable cuyo r-valor es una referencia a un objeto.



# PUNTEROS

- **VALORES:**

- direcciones de memoria
- valor nulo (no asignado) dirección no valida

- **OPERACIONES** (l-valor , r-valor de la variable apuntada)

- **asignación de valor:** generalmente asociado a la alocaación de la variable apuntada

- **referencias:**

- a su valor (dirección) operaciones entre punteros
- al valor de la variable apuntada: **dereferenciación** implícita

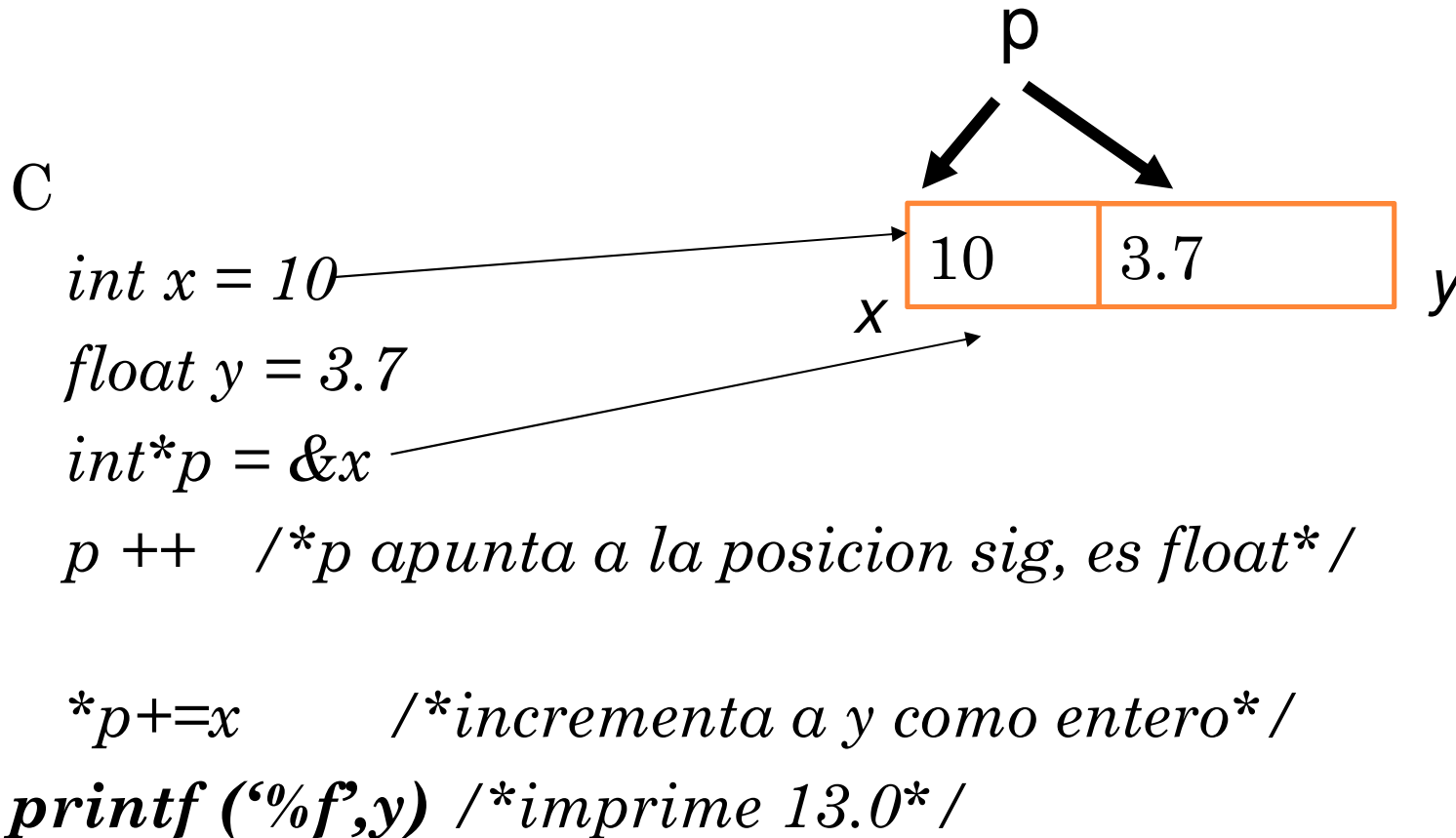


# INSEGURIDAD DE LOS PUNTEROS

1. Violación de tipos
2. Referencias sueltas - referencias dangling
3. Liberación de memoria: objetos perdidos
4. Punteros no inicializados
5. Alias



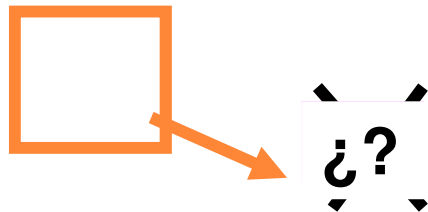
# 1 - VIOLACIÓN DE TIPOS





## 2- PUNTEROS SUELTOS

- Si este objeto no esta alocado se dice que el puntero es peligroso (dangling).
- Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalocada. Si luego se usa el puntero producirá error.



### 3. LIBERACIÓN DE MEMORIA: OBJETOS PERDIDOS

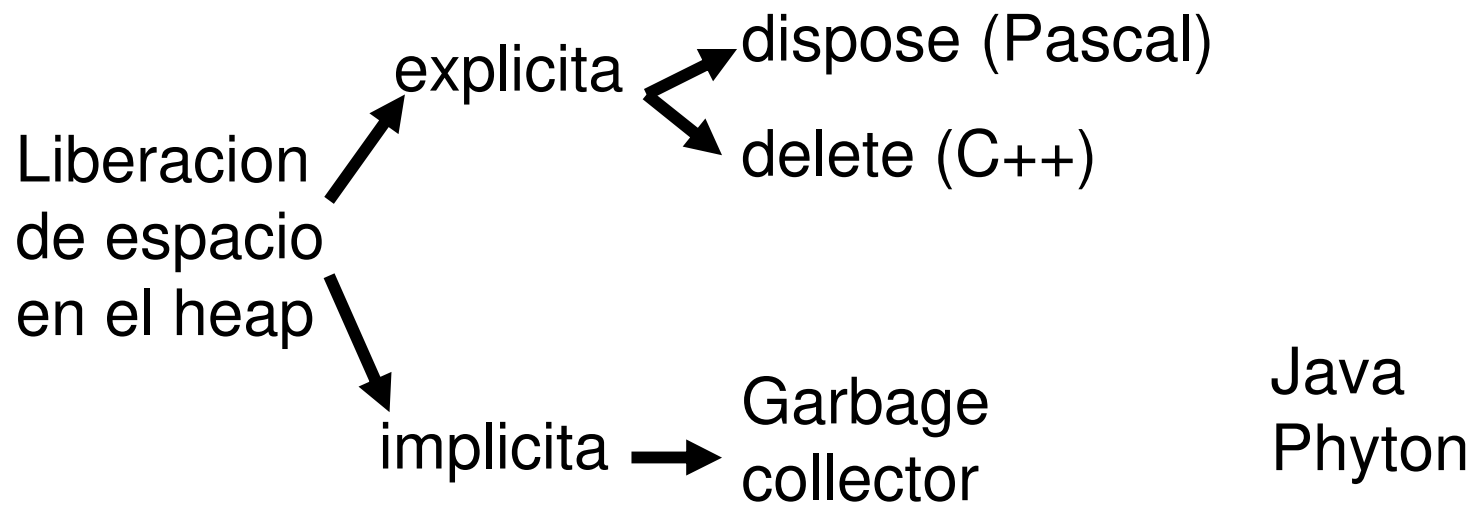
- los objetos (apuntados) que se alocan a través de la primitiva new son alocados en la heap
- La memoria disponible (heap) puede agotarse
- si los objetos en el heap dejan de ser accesibles (objeto perdido) esa memoria podría liberarse



### 3. LIBERACIÓN DE MEMORIA: OBJETOS PERDIDOS

reconocimiento de que  
porción de la memoria  
es basura

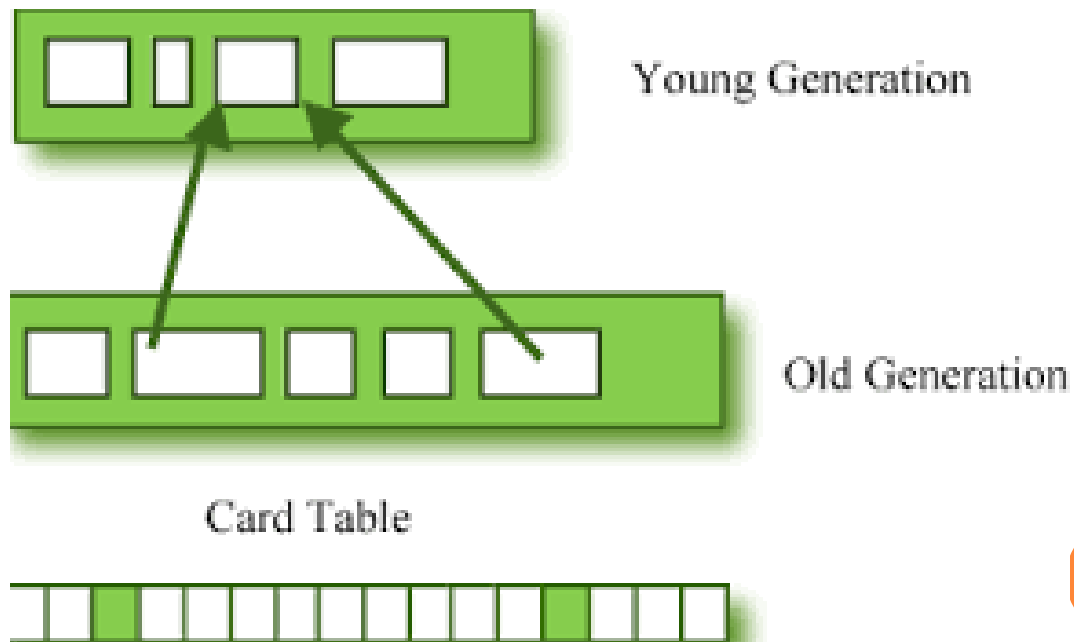
se requiere o no  
intervención del usuario



# IMPLICITA: GARBAGE COLLECTOR

- El sistema, dinámicamente, tomará la decisión de descubrir la basura por medio de una algoritmo de recolección de basura. **garbage collector**.

- LISP
- ADA
- Eiffel y Java
- C
- python
- ruby



## 4. PUNTEROS NO INICIALIZADOS

- Peligro de acceso descontrolado a posiciones de memoria
- Verificación dinámica de la inicialización
- Solución:

valor especial nulo:        nil en Pascal

void en C/C++

null en ADA, Phytton



## 5 - ALIAS

*int\* p1*

*int\* p2*

*int x*

*p1 = &x*

*p2 = &x*

p1 y p2 son punteros

p1 y x son alias

p2 y x también lo son

