Tipo de dato:

- **valores + operaciones** (conjunto de **valores** y **operaciones** asociadas a dichos valores)
- Podemos decir que tenemos 2 visiones:[1]
  - **Interna:** esta mas asociada a los valores
  - **Externa:** esta mas asociada a las operaciones (*esta visión es a la que apuntamos mucho en la materia*)

Paradigma	Valores[2]			Operaciones
	Simples	Compuestos[3]	Orden Superior	
Funcional	Numero (Int,Float…) Caracter Booleano	Lista Tupla	Función	Aplicar Funciones
Lógico	Numero Átomo	Lista Functor	Consulta[4]	Relacionar individuos
Procedural	Numero Char	Vector Registro (Struct)	-	Ejecutar procedimientos
Objetos	Objeto			Enviar mensajes

Con todo esto podemos decir que **programar** es **definir** los **valores** que voy a usar **y** las **operaciones** que voy a necesitar.

Pensemos un ratito en valores…

Todo lo que vamos a decir acá aplica a todos los paradigmas, empezamos con un ejemplo en funcional porque es lo q tienen mas fresquito

Si yo tengo:

Ej	Valor	Tipo de dato	Operaciones que puedo aplicar
A	[1,2,3]	[Int]	• sum • maximun • *length* • *head* • *tail*
B	[(“uno”,(1+)),(“dos”,(2*))]	[(String,(Int -> Int))]	• map fst • filter (isUpperCase.fst) • *length* • *head* • *tail*

En el ejemplo tanto A como B son de tipos diferentes (**[Int] ≠ [(String,(Int -> Int))]**) pero si miramos bien hay funciones que le puedo aplicar tanto a A como a B.
Esto de aplicar indistintamente huele a polimorfismo (quizás porque lo sea), recordemos la definición de polimorfismo que vimos en objetos:
Hay polimorfismo cuando un tercero usa indistintamente a dos o más objetos.
Si analizamos la definición podríamos decir que acá nos importan mas los valores (que objeto usa polimórficamente a cuales), pero hay otra forma de ver el polimorfismo mas relacionada con operaciones:
Hay polimorfismo cuando una operación puede recibir valores de distinto tipo
En nuestros ejemplos, las operaciones polimórficas son

- *length* –tipo-> [a]->Int

---

[1] Que le demos mas importancia a los valores no significa que me olvido de las operaciones y biceversa
[2] Son los parámetros de las operaciones
[3] Un valor compuesto se compone por uno o mas valores (de cualquiera de las 3 “categorías” que nombramos en el cuadrito)
[4] Que usa un predicado

- *head* –tipo-> [a]->a
- *tail* –tipo-> [a]->a

Otros ejemplos de operaciones polimórficas

- *fst* –tipo-> (a,b)->a
- *snd* –tipo-> (a,b)->b
- …

Entonces podemos resumir que hay varias maneras de ver el polimorfismo

- Mas enfocado en **operaciones** (por lo gral la gente que trabaja en el paradigma funcional lo piensa por acá)
- Mas enfocado en **valores** (por lo gral la gente que trabaja en el paradigma oo lo piensan por acá)
- A nosotros nos gusta decir que:
  - ***dos o mas valores son polimorfitos si comparten al menos un tipo***

Con nuestra definición (la q recuadramos) y volviendo al ejemplo podemos decir que el tipo que comparten tanto A como B es **Lista** (lo cual implica entender length, head, tail ….) Pero a su vez si miramos el ejemplo A puedo decir que es una **Lista de Números** (lo cual implica que le puedo decir sum, maximun,…)
Pero… momento A no era del tipo Lista… y ahora decís que es del tipo Lista de Números. SI, ***un valor puede tener más de un tipo***

Podemos comparar tipos, en nuestro ejemplo (el A) podemos decir que es del tipo lista y lista de números, decimos que lista de números es *mas particular* que lista porque me da una idea mas completa sobre las operaciones puedo hacer, mientras que lista es *mas general* que lista de números.
Pero también nos puede pasar que *no sean comparables*, por ejemplo el tipo lista de duplas donde el primer elemento (de la dupla) es un numero ([(Num,a)]) y el tipo lista de duplas donde el segundo elemento (de la dupla) es un numero ([(a,Num)]).

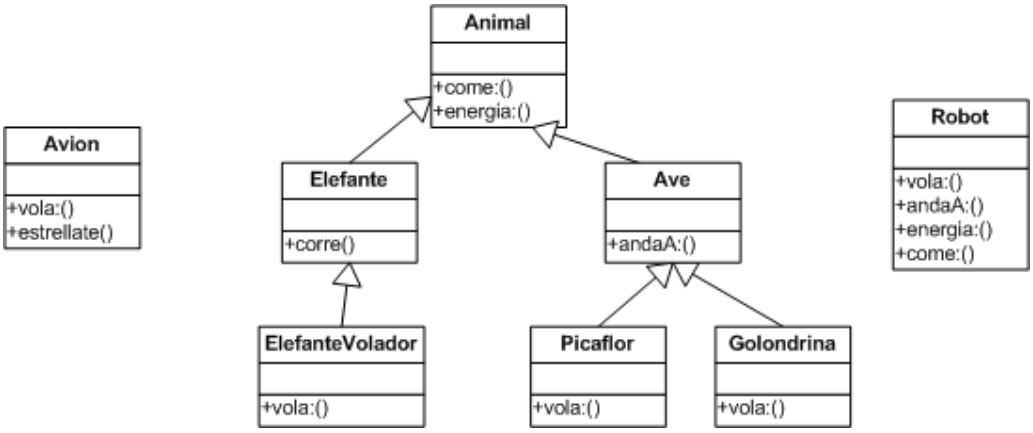
[a]	Mas general que	[Num]
[Num]	Mas particular que	[a]
[(a,Num)]	No es comparable con	[(Num,a)]

Si volvemos al cuadrito puedo decir:

Valor		**Operaciones**	**Tipo “donde están definidas”**
[1,2,3]	[Int]	sum maximun	Lisa de números
		length head tail	Lista
[(“uno”,(1+)),(“dos”,(2*))]	[(String,(Int -> Int))]	map fst	Lista de duplas
		filter (isUpperCase.fst)	Lista de duplas donde el primer elemento es un String
		length head tail	Lista

Veamos un ejemplo pensando en objetos (porque como dijimos antes todo esto es **independiente del paradigma**)

Imaginemos este dominio



Tenemos el siguiente workspace:

```
pepita := Golondrina new.
pp := Picaflor new.
estampi := Elefante new.
dumbo := ElefanteVolador new.
boing747 := Avion new.
r2d2 := Robot new.
```

Todo esto va a andar

```
pepita vola: 40.
pp vola: 40.
boing747 vola: 40.
estampi corre.
dumbo vola: 40.
r2d2 vola: 40.
robot come: 100.
boing747 estrellate.
pp isNil.
r2d2 = dumbo.
```

Entonces de que tipo es[5] r2d2

- Object (entiende isNil… etc)
- Ave (entiende andaA:)
- Animal(entiende come: y energia:)
- Robot

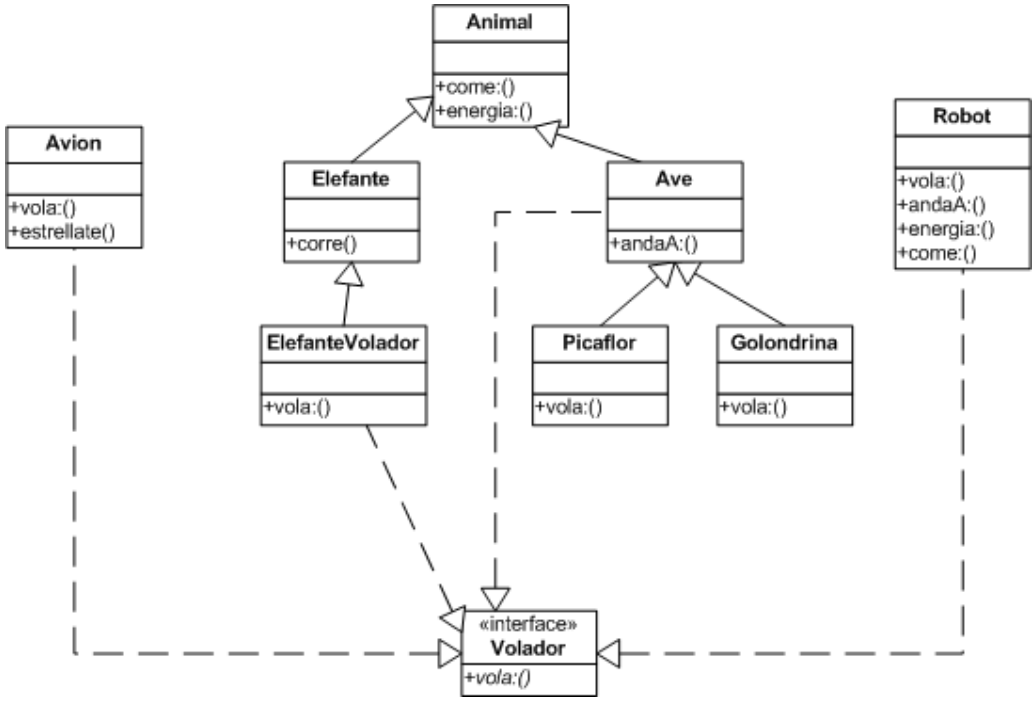
Ahora esto también anda:

```
deTodo = Bag with: pepita with: r2d2 with: boing747 with: dumbo.
deTodo do:[:e|e vola:40].
```

¿Por que anda? Porque todos (pepita, r2d2, boing747 y dumbo) entienden el mensaje vola: entonces podemos decir que **comparten un tipo**, el tipo **Volador**. Que si miramos el diagrama no esta explicitado.

Por comunicación se puede explicitar, pero primero demos una definición mas:
Interfaz: es la definición explicita de un tipo. (**nota: una interfaz NO tiene código solo define**)
Entonces si en el diagramas incorporamos la interfaz Volador nos queda:

[5] Para mirar de que tipo es un objeto miro en las operaciones



Existe algo llamado **duck typing** (si algo habla como pato, camina como pato y huele a pato, entonces ES un Pato)
También podemos agregar que hay lenguajes que me dejan no definir explícitamente el tipo como por ejemplo Smalltalk, que con lo que teníamos antes ya anda y esta todo ok, pero hay otro lenguajes que no (un ejemplo es Java, C++….. de esto hablamos mas abajo)

Aca tambien podemos comparar tipos, por ejemplo con `dumbo`:

Object	Mas general que	Elefante
ElefanteVolador	Mas particular que	Elefante
Elefante	No es comparable con	Volador

El ejemplo de lógico se los dejamos a ustedes ☺

Chequeo de tipos

Acá vamos a hablar a nivel **lenguaje**, noten que al principio veníamos hablando a nivel **paradigma**.
Podríamos decir que hay dos tipos de lenguajes, los que piden que explicitemos el tipo y los que no, pensemos en los lenguajes que conocemos y armemos un lindo cuadro resumen:

	Lenguaje	Definición explicita de tipos	Chequeo de tipos	Aclaraciones
Objetos	Smalltalk	No	Mientras ejecuta	
	Java C++ C#	Si	Antes de ejecutar	Tienen casteo (ejecutan chequeo de tipos mientras ejecutan)
Funcional	Hugs GHC	No[6] / Si	Antes de ejecutar	
	Lisp	No	Nunca	
Logico	Swi-Prolog	No	Nunca	Chequea en ejecución los parámetros del is
	Turbo-Prolog	Si	Antes de ejecutar	
Procedural	C Pascal Cobol	Si	Antes de ejecutar	C tiene casteo (nunca ejecuta el chequeo en este caso)
	Assembler	No	Nunca	

ERROR DE TIPO: Usar un valor en una Operación que no va para ese valor
Ejemplos:

	Operacion	**"Compila"/ Me deja guardarlo**	**Tiene error de tipos?**	**Que me dice?**
Smalltalk	$a+1		Si	Error: $a no entiende el mensaje +
	1 even		No	-
	pepita even		Si	Error: pepita no entiende el mensaje even
	pepita collect:[….]		Si	Error: pepita no entiende el mensaje collect:
Funcional	> 'a' + 1	-[7]	Si	Error de tipo
	f x = min x ('a'+1)		Si	Directamente no compila y me dice que tengo un error de tipos
Logico	?- hermano(1,'bart')		Si	Me responde que No[8]
	proximaEdad(E1,E2):- E2 is E1+1. ?- proximaEdad(2,X).		No	X=3
	proximaEdad(E1,E2):- E2 is E1+1. ?- proximaEdad('bart',1).		Si	Error!!

Ass	Mov AX, [0110011[9]]
	Mov BX, [1011110[10]]
	Add AX, BX
			Si	No pasa nada…

[6] **Motor de inferencia**
[7] Estoy en una consulta no aplica
[8] Que no se rompa el programa no significa que no exista el error, de hecho hermano tiene que aplicarse solo a personas y para mi el numero 1 NO es una persona.
[9] Representa el string "hola"
[10] Representa numero pi