



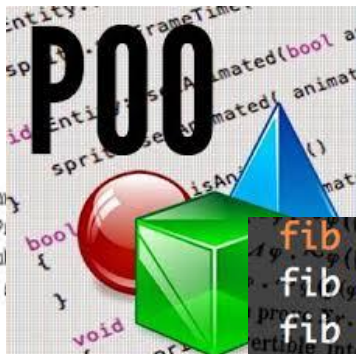
# Paradigmas de Programación

## CYPLP

# PARADIGMAS

Un paradigma de programación es un **estilo de desarrollo de programas**, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez, a partir del tipo de órdenes que permiten implementar, tiene una **relación directa con su sintaxis**.

```
1 procedure DandC(pbm, sol)
2 local var aux;
3 begin
4   if easy(pbm) then
5     solve(pbm)
6   else
7     begin
8       divide(pbm, subpbm, aux);
9       parallel 1 in 1..numP
10         DandC(subpbm[i], sol);
11       combine(subsol, aux, sol);
12     end
13 end;
```



```
?- p_exam(noest,notar).
false.
```

```
?- p_exam(est,tar).
true .
```

```
?- p_exam(noest,tar).
true .
```

```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

# PRINCIPALES PARADIGMAS

- **Imperativo:** sentencias + secuencias de comandos

**Declarativo.** Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.

**Lógico.** Aserciones lógicas: hechos + reglas, es declarativo

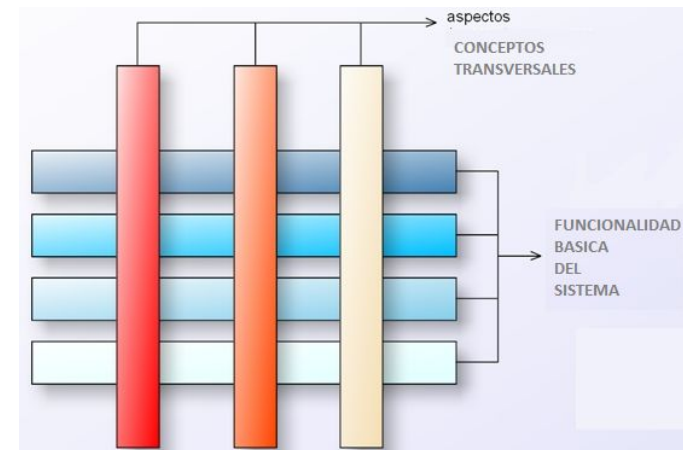
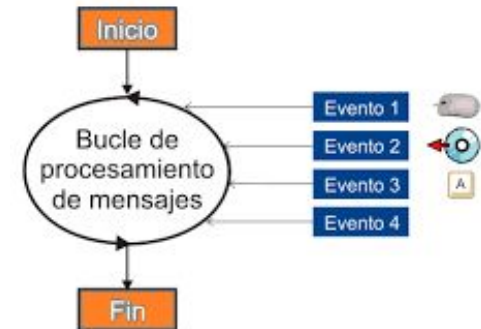
**Funcional.** Los programas se componen de funciones

**Orientado a Objetos :** Métodos + mensajes.

# PRINCIPALES PARADIGMAS

Otra forma de clasificación mas reciente:

- **Dirigido por eventos.** El flujo del programa está determinado por sucesos externos (por ejemplo, una acción del usuario).
- **Orientado a aspectos.**  
Apunta a dividir el programa en módulos independientes, cada uno con un comportamiento y responsabilidad bien definido. [Video explicativo.](#)



# PROGRAMACIÓN LÓGICA

- La programación lógica es un tipo de paradigmas de programación dentro del paradigma de programación declarativa
- Es un paradigma en el cual los programas **son una serie de aserciones lógicas.**
- El conocimiento se representa a través de **reglas y hechos**
- Los objetos son representados por **términos**, los cuales contienen constantes y variables
- PROLOG es el lenguaje lógico más utilizado.

# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

- La sintáxis básica es el “**término**”
- **Variables:**
- Se refieren a elementos indeterminados que pueden sustituirse por cualquier otro.
  - “*humano(X)*”, la X puede ser sustituida por constantes como: juan, pepe, etc.
  - Los nombres de las variables comienzan con mayúsculas y pueden incluir números.
- **Constantes:**
- A diferencia de las variables son elementos determinados.

“*humano(juan)*”

La constantes son string de letras en minúsculas  
(representan objetos atómicos) o string de dígitos  
(representan números).

# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

Término compuesto:

- Consisten en un "functor" seguido de un número fijo de argumentos encerrados entre paréntesis, los cuales son a su vez términos.
- Se denomina "aridad" al número de argumentos.
- Se denomina "estructura" (ground term) a un término compuesto ***cuyos argumentos no son variables.***

Ejemplos:

<b>padre</b>		<b>constante</b>
<b>Longitud</b>	→	<b>variable</b>
<b>tamaño(4,5)</b>	→	<b>estructura</b>
	→	

# ELEMENTOS DE LA PROGRAMACIÓN LÓGICA

Listas:

- La constante `[]` representa una lista vacía
- El functor “.” construye una lista de un elemento y una lista. Ejemplo: `.(alpha,[])`, representa una lista que contiene un único elemento que es `alpha`.
- Otra manera de representar la lista es usando `[]` en lugar de `.()`. Ejemplo anterior la lista quedaría: `[alpha,[]]`
- Y también se representa utilizando el símbolo `|`  
`[alpha | []]`

La notación general para denotar lista es : `[X | Y]`

X es el elemento cabeza de la lista e

Y es una lista, que representa la cola de la lista que se está modelando



# CLÁUSULAS DE HORN

- Un programa escrito en un lenguaje lógico es una secuencia de “cláusulas”.
- Las cláusulas pueden ser: un “Hecho” o una “Regla”.

Hecho:

- Expresan relaciones entre objetos
- Expresan verdades
- Son expresiones del tipo  $p(t_1, t_2, \dots, t_n)$

Ejemplos:

- $\text{tiene}(\text{coche}, \text{ruedas})$   $\square$  representa el hecho que un coche tiene ruedas
- $\text{longitud}([], 0)$   $\square$  representa el hecho que una lista vacía tiene longitud cero
- $\text{moneda}(\text{peso})$   $\square$  representa el hecho que peso es una moneda.

# CLÁUSULAS DE HORN

Regla:



- Cláusula de Horn
- Tiene la forma: conclusión :- condición. (‘:-’ syntaxis prolog)

Dónde:

- :- indica “Si”
  - conclusión es un simple predicado y
  - condición es una conjunción de predicados, separados por comas. Representan un AND lógico
- 
- En un lenguaje procedural una regla la podríamos representar como: if condición else conclusión.

# PROGRAMAS Y QUERIES

## Ejemplo de programa:

**OR** { longitud ([] , 0).  **HECHO**  **AND**  
longitud ([X | Y], N) :- longitud(Y, M), N ≤ M + 1.

Programa: conjunto de cláusulas

**REGLA** 

?-longitud([rojo | [verde | [azul | [] ] ] ] , X).

**QUERY** 

Query: Representa lo que deseamos que sea contestado

# PROGRAMAS Y QUERIES

## Programa:

longitud ([],0).

longitud ([X | Y],N) :- longitud(Y, M), N=M + 1.

?-longitud([rojo | [verde | [azul | [] ] ] ],X).

longitud([],T) T=0

longitud([azul | [] ],Z) Z=T+1 => Z=1

longitud([verde | [azul | [] ] ],M) M=Z+1 => M=2

longitud([rojo | [verde | [azul | [] ] ] ],X) X=M+1 => X=3

# EJECUCIÓN DE PROGRAMAS

- Un programa es un conjunto de reglas y hechos que proveen una especificación declarativa de que es lo que se conoce y la pregunta es el objetivo que queremos alcanzar.
- La ejecución de dicho programa será el intento de obtener una respuesta.
- Desde un punto de vista lógico la respuesta a esa pregunta es “YES”, si la pregunta puede ser derivada aplicando “deducciones” del conjunto de reglas y hechos dados.

Ejemplo???

# EJECUCIÓN DE PROGRAMAS: EJEMPLO

Programa que describe una relación binaria (rel) y su cierre (clos):

**rel(a,b).**  
**rel(a,c).**  
**rel(b,f).**  
**rel(f,g).**  
**clos(X,Y) :- rel(X, Y).**  
**clos(X,Y) :- rel(X, Z), clos(Z, Y).**

**?-clos(a,f)**

PROGRAMA

QUERY

clos (a,f)

rel(a,f)

falla

clos (a,f)

**rel(a,Z1),** clos(Z1,f)

**rel(a,b),** clos(b,f)

**rel(b,f)**

**YES**

EJECUCIÓN

# PROGRAMACIÓN ORIENTADA A OBJETOS

“Un programa escrito con una lenguaje OO es un conjunto de OBJETOS que **INTERACTÚAN** mandándose **MENSAJES**”

Los elementos que intervienen en la programación OO son:

- Objetos
- Mensajes
- Métodos
- Clases

## **Objetos:**

- Son entidades que poseen estado interno y comportamiento
- Es el equivalente a un dato abstracto

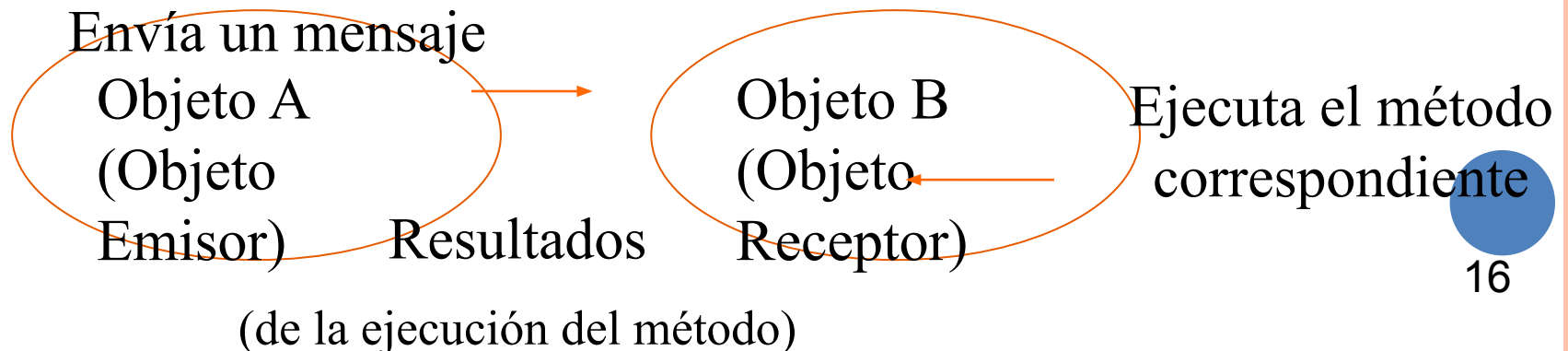
# PROGRAMACIÓN ORIENTADA A OBJETOS

## Mensajes:

- Es una petición de un objeto a otro para que este se comporte de una determinada manera, ejecutando uno de sus métodos
- TODO el procesamiento en este modelo es activado por mensajes entre objetos.

## Métodos:

- Es un programa que está asociado a un objeto determinado y cuya ejecución solo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes





# PROGRAMACIÓN ORIENTADA A OBJETOS

## Clases:

- Es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo
- Cada objeto pertenece a una clase y recibe de ella su funcionalidad
- Primer nivel de abstracción de datos: definimos estructura, comportamiento y tenemos ocultamiento.
- La información contenida en el objeto solo puede ser accedida por la ejecución de los métodos correspondientes

## Instancia de clase:

- Cada vez que se construye un objeto se está creando una INSTANCIA de esa clase
- Una instancia es un objeto individualizado por los valores que tomen sus atributos



## Otro aspecto de las abstracciones de datos

### GENERALIZACIÓN/ESPECIFICACIÓN



### HERENCIA

El segundo nivel de abstracción consiste en agrupar las clases en jerarquías de clases (definiendo SUB y SUPER clases), de forma tal que una clase A herede todas las propiedades de su superclase B (suponiendo que tiene una)

Ejemplo: Se tiene definido la siguiente clase

PERSONA

- Nombre
- Edad
- Sexo
- Documento
- Dirección
- Teléfono

**ver-nombre**  
**ver-edad**  
**ver-teléfono**  
**ver-documento**  
**ver-sexo**  
**cambiar-dirección**  
**sacar-documento, etc.**

EMPLEADO  
•Curriculum  
•cuil

## Otros conceptos adicionales

### Polimorfismo:

- Es la capacidad que tienen los objetos de distintas clases de responder a mensajes con el mismo nombre

### Ejemplo:

$3 + 5$	Se aplica suma entre números
"Buenos" + "días"	Se concatenan strings

### Binding dinámico:

Es la vinculación en el proceso de ejecución de los objetos con los mensajes

# PROGRAMACIÓN ORIENTADA A OBJETOS

## **C++ (Lenguaje híbrido)** **Algunas características**

- Lenguaje extendido del lenguaje C
- Incorporó características de POO

Los objetos en C++:

- Se agrupan en tipos denominados clases
- Contienen datos internos que definen su estado interno
- Soportan ocultamiento de datos
- Los métodos son los que definen su comportamiento
- Pueden heredar propiedades de otros objetos
- Pueden comunicarse con otros objetos enviándose mensajes

# Conceptos y Paradigmas de Lenguajes de Programación

Lenguajes Basados en Script



# Lenguajes Basados en Script – Análisis Lenguajes Convencionales

**Los lenguajes de programación tradicionales están destinados principalmente para la construcción de aplicaciones auto-contenidas:**

**Programas que aceptan una suerte entrada, la procesan de una manera bien entendida y finalmente generan una salida apropiada.**



# Lenguajes Basados en Script – Análisis

## Lenguajes Convencionales

Tienden a mejorar eficiencia, mantenibilidad, portabilidad y detección estática de errores. Los tipos se construyen alrededor de conceptos a nivel hardware como enteros de tamaño fijo, punto flotante, caracteres y arreglos. Esto requiere un diseño e implementación del lenguaje que sea robusto y eficiente.

# Lenguajes Basados en Script – Análisis Lenguajes Convencionales

**Sin embargo, muchos de los usos actuales en diferentes entornos, requieren la coordinación de múltiples programas, por lo que podría ser necesario “combinar” o “pegar” estos programas de alguna forma. Y en este caso buscaremos la forma más sencilla y directa de lograrlo.**



# Lenguajes Basados en Script – Definición

En un principio los lenguajes de scripting eran un conjunto de comandos escritos en un archivo para ser interpretado.

A este archivo se lo denomina “script”.

Los primeros LBS eran un conjunto de comandos que eran interpretados como llamadas al sistema como manejo o filtrado de archivos.

# Lenguajes Basados en Script – Definición

Luego se agregaron variables, flujo de control, etc. y fueron escalando hasta convertirse en lenguajes de programación completos como los conocidos actualmente.

(Perl, Python, PHP, javascript, etc.)

Los lenguajes script tienden a mejorar flexibilidad, desarrollo rápido y chequeo dinámico. Su sistema de tipos se construye sobre conceptos de mas alto nivel como tablas,

# Lenguajes Basados en Script – Definición

**“Los lenguajes script asumen la existencia de componentes útiles en otros lenguajes. Su intención no es escribir aplicaciones desde el comienzo sino por combinación de componentes”**

**John Ousterhout**

**Creador de TCL**

**“Tool Command Language”**

# Lenguajes Basados en Script – Características y Objetivos LBS

## Objetivos de los LBS:

- **Uso de scripts para “pegar” o combinar programas.**
- **Desarrollo y evolución rápida.**
- **Asociado a editores livianos.**
- **Interpretados – (modestos requerimientos de eficiencia)**
- **Alto nivel de funcionalidad en aplicaciones de áreas específicas.**

# Lenguajes Basados en Script – Características y Objetivos LBS

## Combinar Programas:

Los lenguajes script de propósito general (Perl, Python) suelen conocerse como glue-languages.

Se diseñaron para “pegar” programas existentes a fin de construir un sistema mas grande.

Se utilizan como lenguajes de extensión, ya que permiten al usuario adaptar o extender las funcionalidad de las herramientas script

# Lenguajes Basados en Script – Características y Objetivos LBS

## Desarrollo y evolución rápida:

Algunos “script” se escriben y ejecutan una única vez como una secuencia de comandos. En otros casos se utilizan más frecuentemente por lo que deben ser fácilmente adaptados a nuevos requerimientos. Esto implica que deben ser fáciles de escribir y con una sintaxis concisa.

## Asociado a editores livianos:

Pueden ser escritos en procesadores de texto

# Lenguajes Basados en Script – Características y Objetivos LBS

## Interpretados:

**La eficiencia no es un requisito esencial para los scripts. Sin embargo debe ser considerado al combinar programas.**

**La velocidad de ejecución de los script no es de importancia crítica. Los gastos generales de interpretación y de comprobación dinámica se puede tolerar.**

# Lenguajes Basados en Script – Aspectos Principales y Facilidades

Es difícil definirlos con precisión, aunque hay varias características que tienden a tener en común:

- Alto nivel de procesamiento de Strings y generación de Reportes. (Expresiones Regulares)
- Alto nivel para soporte de interfaces de usuario (GUI).
- Tipado dinámico.



# Lenguajes Basados en Script – Aspectos Principales y Facilidades

## Tipado Dinámico:

Cuando se utilizan como glue-languages puede necesitarse intercambiar datos de distinto tipo entre distintos subsistemas y estos pueden ser incompatibles. Por esto, si el LBS tiene un sistema de tipos simple podría ser demasiado inflexible y por otro lado uno muy completo atendería contra un rápido desarrollo y evolución del sistema.

Por lo general los tipos no necesitan ser

# Lenguajes Basados en Script – Aspectos Principales y Facilidades

En los LBS las declaraciones son escasas o nulas y proveen reglas simples que gobiernan el alcance de los identificadores.

Por ejemplo, en Perl, cada identificador es global por omisión (hay opciones para limitar el alcance).

En otros lenguajes (e.g., PHP y Tcl), cada cosa es local por omisión (un objeto global debe ser explícitamente importado).

Python adopta la regla: “a una variable que

# Lenguajes Basados en Script – Aspectos Principales y Facilidades

**Dado la falta de declaraciones, muchos LBS incorporan tipificación dinámica.**

**En algunos lenguajes el tipo de la variable es chequeada inmediatamente antes de su uso: e.g., PHP, Python, Ruby, y Scheme.**

**En otros, el tipo de una variable (por ende su valor) será interpretado de manera diferente según el contexto: e.g., Rexx, Perl, y Tcl.**

# Lenguajes Basados en Script – Aspectos Principales y Facilidades

## Ejemplo en Perl

```
$a = "4";
```

```
print $a . 3 . "\n"; # '.' es la concatenación
```

```
print $a + 3 . "\n"; # '+' es la suma
```

Dará la siguiente salida:

43

7

# Lenguajes Basados en Script – Aspectos Principales y Facilidades

- Los LBS son ricos en:
- Conjuntos
- Diccionarios
- Listas
- Tuplas, etc.

Por ejemplo:

En muchos LBS es común poder indizar arreglos a través de cadenas de caracteres, lo que implica una implementación de tablas de

# Lenguajes Basados en Script – Dominios de Aplicación

## Principales Ejemplos:

- Lenguaje de comandos (shell)
- Procesamiento de texto y Generación de Reportes
- Matemática y Estadística
- Lenguajes de "pegado"(GLUE) y de propósito general

## Extensión de Lenguajes

WWW como ejemplo especial

# Lenguajes Basados en Script – Caso de Estudio: Javascript y Python

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="JavaScript">
    <!-- function HolaMundo() { alert("¡Hola, mundo!"); }
    // --->
  </SCRIPT>
</HEAD>
<BODY>
<FORM>
  <INPUT TYPE="button" NAME="Boton" VALUE="Pulsame"
  onClick="HolaMundo()">
</FORM>
</BODY>
</HTML>
```

llamada al método alert (que pertenece al objeto window) que es la que se encarga de mostrar el mensaje en pantalla

Dentro de estos elementos se puede poner funciones en JavaScript

onClick es un *evento*. Cuando el usuario pulsa el botón, el evento onClick se dispara y ejecuta el código que tenga entre comillas, en este caso la llamada a la función HolaMundo(), que tendremos que haber definido con anterioridad.

# Lenguajes Basados en Script – Caso de Estudio: Javascript y Python

- Javascript fue desarrollado para el navegador Netscape en los años 90.
- El lenguaje estándar fue desarrollado a fines de los años 90 por European Computer Manufacturers Association (ECMA) como ECMA-262.
- Aunque el intérprete javascript puede ser embebido en cualquier aplicación por lo general se utiliza en los navegadores web, donde el código se asocia a documentos



# Lenguajes Basados en Script – Caso de Estudio: Javascript y Python

- La sintaxis es similar a Java pero por el contrario es dinámicamente tipado.
- Los strings y arreglos tienen longitud dinámica y sus índices no son chequeados.
- Soporta OO pero no Herencia. (Herencia por prototipos)
- Uno de los usos más importantes es la manipulación dinámica del documento HTML.

# Lenguajes Basados en Script – Caso de Estudio: Javascript y Python

## • Python:

- Software Libre.
- Interpretado (intérprete de comandos).
- Multiplataforma.
- Multiparadigma (Imperativo, OO, funcional).
- Fuertemente Tipado.
- Dinámicamente Tipado.
- Sintaxis simple y legible.
- Las variables NO se declaran. Nacen cuando se le

# PARADIGMA APLICATIVO O FUNCIONAL

Basado en el uso de funciones. Muy popular en la resolución de problemas de inteligencia artificial, matemática, lógica, procesamiento paralelo

Ventajas:

- Vista uniforme de programa y función
- Tratamiento de funciones como datos
- Liberación de efectos colaterales
- Manejo automático de memoria

Desventaja:

- Ineficiencia de ejecución

# PARADIGMA FUNCIONAL

## Características de los lenguajes funcionales

- Provee un conjunto de funciones primitivas
- Provee un conjunto de formas funcionales
- Semántica basada en valores
- Transparencia referencial
- Regla de mapeo basada en combinación o composición
- Las funciones de primer orden

# PROGRAMACIÓN FUNCIONAL

## Funciones

- El VALOR más importante en la programación funcional es el de una FUNCIÓN
- Matemáticamente una función es un correspondencia :  
 $f: A \rightarrow B$
- A cada elemento de A le corresponde un único elemento en B
- $f(x)$  denota el resultado de la aplicación de  $f$  a  $x$
- Las funciones son tratadas como valores, pueden ser pasadas como parámetros, retornar resultados, etc.

# PROGRAMACIÓN FUNCIONAL

## Definiendo Funciones:

- Se debe distinguir entre el VALOR y la DEFINICIÓN de una función.
- Existen muchas maneras de DEFINIR una misma función, pero siempre dará el mismo valor, ejemplo:

DOBLE X = X + X

DOBLE' X = 2 \* X

Denotan la misma función pero son dos formas distintas de definirlas

## Tipo de una función

- Puede estar definida explícitamente dentro del SCRIPT, por ejemplo:

cuadrado::num → num (define el tipo)

cuadrado x = x \* x (definición de la función)

- O puede **deducirse/inferirse** el tipo de una función

# PROGRAMACIÓN FUNCIONAL

## Expresiones y valores

- La expresión es la noción central de la programación Funcional
- Característica más importante:  
“Una expresión es su VALOR”
- El valor de una expresión depende ÚNICAMENTE de los valores de las sub expresiones que la componen.
- Las expresiones también pueden contener VARIABLES, (valores desconocidos)

# PROGRAMACIÓN FUNCIONAL

## Expresiones y valores

- La noción de Variable es la de “variable matemática”, no la de celda de memoria.
- Las expresiones cumplen con la propiedad de  
  
“TRANSPARENCIA REFERENCIAL”: Dos expresiones sintácticamente iguales darán el mismo valor.
- **No** existen EFECTOS LATERALES”



# PROGRAMACIÓN FUNCIONAL

- Ejemplos de expresiones para evaluar

Expresión	Valor	
47	47	
$(\textcircled{*} 4 \ 7)$	28	Se está utilizando una Función primitiva
$(\textcircled{+} 49 \ 5)$	54	

- Definiendo funciones....

$\text{cuadrado } x = x * x$

$\text{min } x \ y = x, \text{ if } x < y$   
 $y, \text{ if } x > y$

$\text{cube } (x) = x * x * x$

# PROGRAMACIÓN FUNCIONAL

Un script es una lista de definiciones y

- Pueden someterse a evaluación. Ejemplos:

?cuadrado (3 + 4 )

49

?min 3 4

3

- Pueden combinarse, Ejemplo:

?min(cuadrado ( 1 + 1 ) 3)

3

- Pueden modificarse, ejemplo: Al script anterior le agrego nuevas definiciones:

lado = 12

area = cuadrado lado

# PROGRAMACIÓN FUNCIONAL

- Algunas expresiones pueden **NO** llegar a reducirse del todo, ejemplo:  $1/0$
- A esas expresiones se las denominan **CANÓNICAS**, pero se les asigna **un VALOR INDEFINIDO** y corresponde al símbolo  $\text{bottom}(\wedge)$
- Por lo tanto toda **EXPRESIÓN** siempre denota un **VALOR**

# PROGRAMACIÓN FUNCIONAL

Evaluación de las expresiones:

- La forma de evaluar es a través de un mecanismo de REDUCCIÓN o SIMPLIFICACIÓN
- Ejemplo:

cuadrado (3 + 4)

=> cuadrado 7

se aplicó (+)

=> 7 \* 7 (cuadrado)

se aplicó cuadrado

=> 49

se aplicó cuadrado(\*)

Otra forma sería:

cuadrado (3 + 4)

=> (3 + 4) \* (3 + 4) (cuadrado)

=> 7 \* (3 + 4) (+)

=> 7 \* 7 (+)

=> 49 (\*)

**“No importa la forma de evaluarla, siempre el resultado final será el mismo”**

# PROGRAMACIÓN FUNCIONAL

Existen dos formas de reducción:

- Orden aplicativo
- Orden normal (lazy evaluation)
- Diferida (Haskell)

Aunque no lo necesite  
**SIEMPRE** evalúa los  
argumentos

No calcula más de lo necesario  
La expresión NO es evaluada hasta que  
su valor se necesite  
Una expresión compartida NO es  
evaluada más de una vez

# PROGRAMACIÓN FUNCIONAL

TIPOS {  
Básicos  
Derivados

**Básicos:** Son los primitivos, ejemplo:

**NUM (INT y FLOAT) (Números)**

**BOOL(Valores de verdad)**

**CHAR(Caracteres)**

**Derivados:** Se construyen de otros tipos, ejemplo:

**(num,char) Tipo de pares de valores**

**(num $\rightarrow$ □char) Tipo de una función**

**TODA FUNCIÓN TIENE ASOCIADO UN TIPO**

# PROGRAMACIÓN FUNCIONAL

- Expresiones de tipo polimórficas:

En algunas funciones no es tan fácil deducir su tipo.

Ejemplo:

**id x = x**

Esta función es la **función Identidad**

Su tipo puede ser de  $\text{char} \rightarrow \text{char}$ , de  $\text{num} \rightarrow \text{num}$ , etc.

Por lo tanto su tipo será de  $\beta \rightarrow \beta$  (denota un tipo polimórfico)

Se utilizan **letras griegas** para representar **tipos polimórficos**

Otro ejemplo: **letra x = “A”**

Su tipo será  $\beta \rightarrow \text{char}$

# PROGRAMACIÓN FUNCIONAL

## ● Currificación:

- Mecanismo que reemplaza argumentos estructurados por argumentos más simples.
- Ejemplo: sean dos definiciones de la Función “Suma”

1.  $\text{Suma}(\mathbf{x}, \mathbf{y}) = \mathbf{x} + \mathbf{y}$

2.  $\text{Suma}' \mathbf{x} \mathbf{y} = \mathbf{x} + \mathbf{y}$

→  $\text{Suma}' \mathbf{x} \mathbf{y} = \text{Suma}' \mathbf{x} (\mathbf{y}) = \mathbf{x} + \mathbf{y}$

Existen entre estas dos definiciones una diferencia sutil:

**“Diferencia de tipos de función”**

El tipo de Suma es :  $(\text{num}, \text{num}) \rightarrow \square \mathbf{num}$

El tipo de Suma' es :  $\text{num} \rightarrow (\mathbf{num} \rightarrow \square \mathbf{num})$

Por cada valor de  
x devuelve una  
función

**Aplicando la función:**

$\text{Suma}(1, 2) \rightarrow 3$

$\text{Suma}' 1 2 \square \text{Suma}' 1$  aplicado al valor  
 $2 \rightarrow 3$

Para todo los  
valores devuelve el  
siguiente



# PROGRAMACIÓN FUNCIONAL

## Cálculo Lambda

- El un modelo de computación para definir funciones
- Se utiliza para entender los elementos de la programación funcional y la semántica subyacente, independientemente de los detalles sintácticos de un lenguaje de programación en particular.
- Es un modelo de programación funcional que se independiza de la sintaxis del lenguaje de programación



# PROGRAMACIÓN FUNCIONAL

Las expresiones del Lambda cálculo pueden ser de 3 clases:

- Un simple identificador o una constante. Ej:  $x$ ,  $3$
- Una definición de una función. Ej:  $x.x+1$

$\lambda$

Una aplicación de una función. La forma es  $(e1\ e2)$ , dónde se lee  $e1$  se aplica a  $e2$ .

**Ej: en la funcion cube  $(x) = x * x * x$**

$\lambda x. x * x * x$

$\lambda (x. x * x * x) 2)$

Evaluamos la  
función con 2  
y resulta en 8

# PROGRAMACIÓN ORIENTADA A ASPECTOS

Divide responsabilidades en módulos conocidos como **aspectos**.

Estos aspectos suelen ser “transversales” a toda la aplicación.

Se puede combinar con otros paradigmas.

Su implementación está en un solo lugar en vez de repartirse en todo el código. (Mantenimiento y modificabilidad)

Usos:

- Autenticación
- Seguridad
- Trazabilidad
- Control de errores
- Otros.



# PROGRAMACIÓN ORIENTADA A ASPECTOS

## Conceptos

- Aspecto: Funcionalidad transversal de la aplicación (Ej. login, logs del sistema)
- Punto de enlace (joinpoint): Punto del programa donde el aspecto es conectado.
  - Llamada a un método
  - lanzamiento de excepción
  - Modificación de campo
  - Etc.
- Consejo (advice): Contiene el código que se ejecutará al implementar el Aspecto. Se inserta a través de los joinpoints.
- Punto de corte: define que advice (consejo) se usa en cada joinpoint.
- Introducción: Permite añadir métodos o atributos a una clase existente. Ej: en una auditoría podríamos establecer la fecha de modificación (como atributo) de un objeto.
- Target (objetivo): Clase sobre la que se aplica el aspecto.
- Proxy: Objeto resultante de aplicar un advice (consejo) a un objeto.
- Weaving (costura): integrar el advice (consejo) a los Target de nuestra aplicación.

# PROGRAMACIÓN ORIENTADA A ASPECTOS

## **Formas de definir cuando se ejecuta un aspecto.**

- Antes
- Antes de retornar luego de la ejecución de un método.
- Después de una excepción
- Otras

## **Desventajas**

- Puede ser engorroso de seguir el flujo del programa (tanto en debug como al depurar)
- Dificultad de identificación de problemas.

## **Ventajas**

- Mejora de modularidad
- Elimina dependencias
- Facilita la extensibilidad
- Reutilización de código
- Mejora el mantenimiento del código
- Código más limpio

# PROGRAMACIÓN ORIENTADA A ASPECTOS

## AOP



```
public void insertarCliente (Cliente elCliente, String IdUsuario){
```

```
// código para login
```

```
// código para comprobación usuario válido
```

```
Session miSesion=sessionFactory.getCurrentSession();
```

```
miSesion.save (elCliente);
```

```
}
```

Controlador

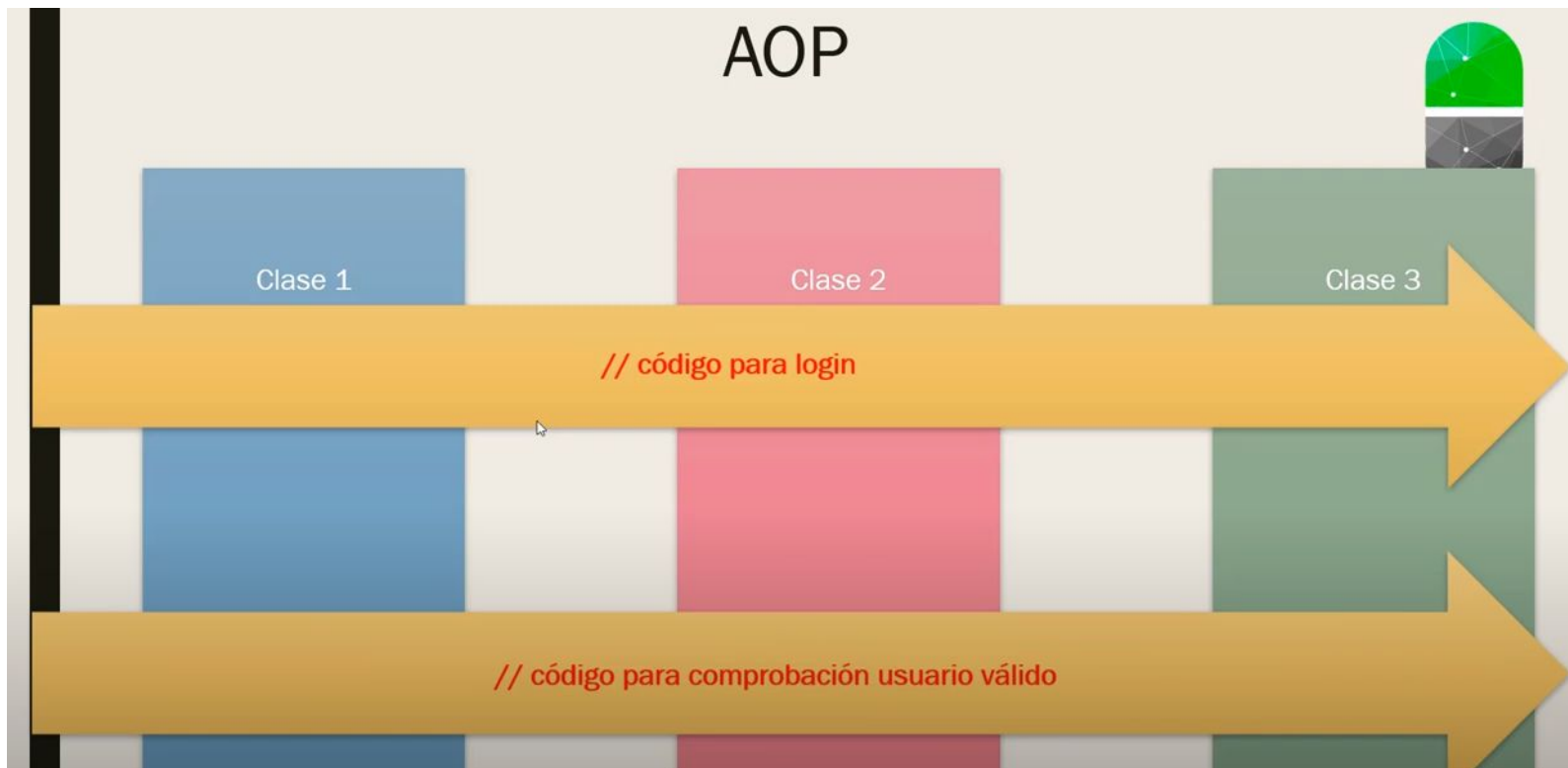
```
// código para login +  
// código para comprobación  
usuario válido
```

ClienteDAO

```
// código para login  
// código para comprobación  
usuario válido
```



# PROGRAMACIÓN ORIENTADA A ASPECTOS



# PROGRAMACIÓN ORIENTADA A ASPECTOS

```
public class Hello {  
  
    public static void main(String[] args) {  
        sayHello();  
    }  
  
    public static void sayHello() {  
        System.out.print("Hola ");  
    }  
}
```

```
public aspect AspectHello {  
  
    pointcut greeting() : execution (* Hello.sayHello(..));  
  
    before(): greeting() {  
        System.out.println("Bienvenidos a POA");  
    }  
  
    after() returning() : greeting() {  
        System.out.println("clase de lenguajes de programacion 2022-1");  
    }  
}
```

Bienvenidos a POA

Hola clase de lenguajes de programacion 2022-1

Process finished with exit code 0



# PROGRAMACIÓN ORIENTADA A ASPECTOS

Ejemplos de lenguajes con programación orientada a aspectos:

- Java: A través de frameworks como Spring AOP y AspectJ.
- C#: Con el framework PostSharp, entre otros.
- Python: Con librerías como Python AOP.
- Perl: Mediante el módulo Aspect.
- Ruby: Con librerías como Ruby AOP.

# PROGRAMACIÓN REACTIVA

La Programación Reactiva es un paradigma declarativo funcional relacionada con el tratamiento de flujo de datos ya que en principio busca **reaccionar** ante un **evento** de forma **asincrónica**. "Asíncrono" significa que algo no sucede en el mismo momento que otro evento, o que no está sincronizado con él. Tiene mucho que ver con el concepto de microservicios y se basa en un [Manifiesto](#).

- Responsivos: Capacidad de completar la tarea en un tiempo determinado.
- Resilientes: Soportar y recuperarse ante errores.
- Elásticos: Pueden aumentar o disminuir los recursos para adaptarse a cambios.
- Orientados a Mensajes: Basado en un sistema de intercambio de mensajes de forma asíncrona entre componentes.

# PROGRAMACIÓN REACTIVA

Está basado en el patron “observer”, que define una relación del tipo (uno a muchos) entre objetos. Cuando el estado de un objeto cambia, este notifica el cambio al resto de los objetos dependientes. La idea básica es que un objeto contiene atributos o métodos observables que otros objetos pueden suscribir pasando una referencia a sí mismos. El objeto observable mantiene una lista de quienes lo observan y le notifica los cambios.

## Componentes

- Observable: Interface que implementa todos los objetos observados.
- Observador: Interface que implementa los objetos que desean observar.
- Suscripción: Función que permite suscribir o desuscribir al observable.

# OPTATIVAS DE LA LIC. EN SISTEMAS

[Programación Lógica](#)

[Programación Funcional](#)

[Programación Funcional \(video\)](#)