Paradigmas

1. ¿Qué es un paradigma?

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 permitan implementar, lo cual tiene una relación directa con su sintaxis.

2. ¿Cuáles son los 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**: 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
  La sintáxis básica es el **término**. Éstos contienen:
  - **Constantes**: son elementos determinados. La constantes son string de letras en minúsculas(representan objetos atómicos) o string de dígitos(representan números). Ejemplo: *humano(juan)*, en donde *juan* representa la constante.
  - **Variables**: elementos indeterminados que pueden sustituirse por cualquier otro. Los nombres de las variables comienzan con mayúsculas y pueden incluir números. Ejemplo: *humano(X)*, la *X* representa la variable y puede ser sustituída por constantes como: juan, pepe, etc.

  Además de los términos, existen los **términos compuestos**. Éstos consisten en un functor seguido de un número fijo de argumentos encerrados entre paréntesis, los cuales son a su vez términos. Denominaremos *aridad* al número de argumentos y *estructura* a un término compuesto cuyos argumentos no son variables. Ejemplos: padre->constante; Longitud->variable; tamaño(4,5)->estructura. A su vez, la aridad de tamaño es 2.
  También existen las 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 *.()*. Volviendo al ejemplo anterior: [alpha,[]]. También puede utilizarse el símbolo | en lugar de ,. En nuestro ejemplo: [alpha|[]]. La notación general para denotar una lista es: [X|Y] donde *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:

  - **Hecho**: expresan relaciones entre objetos o verdades. Son expresiones del tipo p(t1,t2,….tn). Ejemplo: tiene(coche,ruedas)representa el hecho que un coche tiene ruedas.
  - **Regla**: podríamos representarla como *if(condición) else (conclusión);*. Concretamente, las que son de tipo cláusula de Horn tienen la forma de conclusión :- condición donde *:-* indica *si*, *conclusión* es un simple predicado y *condición* es una conjunción de predicados, separados por comas. Representan un AND lógico.

  Programas y queries

  - **Programa**: es un conjunto de reglas y hechos que proveen una especificación declarativa de qué es lo que se conoce y la pregunta es el objetivo que queremos alcanzar.
  - **Query**: representa lo que deseamos que sea contestado a esa pregunta.

Ejecución de programas La ejecución de un programa será el intento de obtener una respuesta a la pregunta ¿Cuál es el objetivo que queremos alcanzar? 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.

- **Funcional o aplicativo**: basado en el uso de funciones. Muy popular en la resolución de problemas de inteligencia artificial, matemática, lógica, procesamiento paralelo, etc.
  Características:
  - Define un conjunto de datos.
  - Provee un conjunto de funciones primitivas.
  - Provee un conjunto de formas funcionales.
  - Requiere de un operador de aplicación.
  - Semántica basada en valores.
  - Transparencia referencial.
  - Regla de mapeo basada en combinación o composición.
  - Las funciones de primer orden.

  Algunas ventajas son:

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

  La desventaja es la ineficiencia de ejecución.
  El **valor** más importante en la programación funcional es el de una **función**. Las funciones son tratadas como valores, pueden ser pasadas como parámetros, retornar resultados, etc.
  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* y *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 cuadrado x= x + x.
  - Puede **deducirse/inferirse** el tipo de una función.

  Expresiones y valores
  Una expresión es su **valor**. El valor de una expresión depende **únicamente** de los valores de las subexpresiones que la componen. Las expresiones también pueden contener **variables**(valores desconocidos). Las expresiones cumplen con la propiedad de "**transparecia referencial**": dos expresiones sintácticamente iguales darán el mismo valor, porque no existen **efectos laterales**.
  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 bottom(^).
  Por lo tanto toda **expresión** siempre denota un **valor**.

  IMPORTANTE: La noción de variable es la de “variable matemática”, no la de celda de memoria. Diferentes ocurrencias del mismo nombre hacen referencia al mismo valor desconocido.

  Evaluación de las expresiones
  La forma de evaluar es a través de un mecanismo de **reducción o simplificación**. Lo importante es que, de nuevo, no importa la forma de evaluarla, **el resultado final siempre será el mismo**. Dos formas de reducción:

  - **Orden aplicativo**: aunque no lo necesite, siempre evalúa los argumentos.

- **Orden normal**: 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.

Tipos de dato
Hay dos tipos:

- **Básicos**: son los primitivos. Ejemplo: NUM (incluye 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 -> char) Tipo de una función.

Toda función tiene asociado un tipo. Sin embargo, algunas veces no es tan simple deducirlo. Es el caso de las **expresiones de tipo polimórficas**.

Currificación Mecanismo que reemplaza argumentos estructurados por argumentos más simples.

Cálculo Lambda Es 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. Las expresiones del cálculo Lambda pueden ser de 3 clases:

- Un simple identificador o una constante. Ej: x, 3.
- Una definición de una función. Ej: λ x.x+1.
- Una aplicación de una función. La forma es (e1 e2), dónde se lee e1 se aplica a e2.

- **Orientado a Objetos**: es un conjunto de **objetos** que **interactúan** mandándose **mensajes**. Sus elementos son:
  - **Objetos**: son entidades que poseen estado interno y comportamiento. Es el equivalente a un dato abstracto.
  - **Mensajes**: es una petición de un objeto a otro para que este se comporte de una determinada manera.
  - **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. Cada objeto pertenece a una clase y recibe de ella su funcionalidad. Es un nivel muy abstracto en el que definimos estructura, comportamiento y tenemos ocultamiento(Primer nivel de abstracción de datos). La clase podría poseer instancias de ésta, es decir, un objeto individualizado por los valores que tomen sus atributos. Cada vez que se construye un objeto se está creando una **instancia** de esa clase.
  - **Clases**: es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo.

  Generalización/Especificación: herencia
  El segundo nivel de abstracción consiste en agrupar las clases en jerarquías de clases(definiendo subclases y superclases), de forma tal que una clase A herede todas las propiedades de su superclase B(suponiendo que tuviera una).

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

- **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.