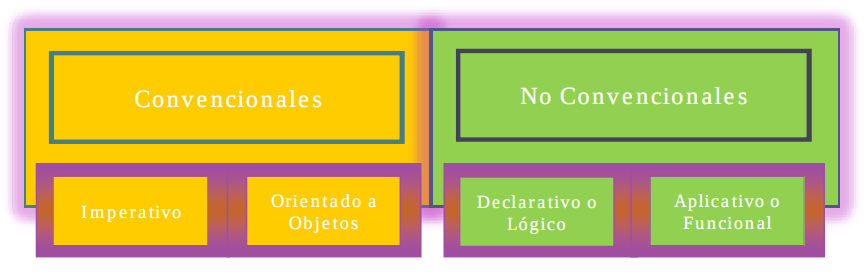
Definición de Paradigma de Programación

* Colección de patrones conceptuales que modelan la forma de razonar sobre problemas, de formular algoritmos y de estructurar programas.
* Métodos que orientan el proceso de la programación.
* Un lenguaje **soporta** un paradigma si el lenguaje provee mecanismos que facilitan su implementación eficiente.
* Un lenguaje **admite** un paradigma si es posible escribir un programa siguiendo los lineamientos del paradigma pero con un notable esfuerzo. El lenguaje no exige la aplicación del paradigma. No provee facilidades para implementarlo.

Clasificación general:



* Convencionales:
  + Modelo de ejecución basado en el hardware de la computadora tradicional .
  + Ejecución de instrucciones en forma secuencial.
  + Ajustados a la arquitectura de Von Neumann.
  + Centrados en responder como resolver determinados problemas.
  + Tipos:
    - Imperativo: consiste en construir los estados sucesivos de la maquina necesarios para llegar a la solución, mediante el uso de ordenes.
    - Orientado a objetos: se construyen objetos complejos de datos y se designa un conjunto limitado de funciones para que operen sobre esos datos.
* No convencionales:
  + La forma de especificar y ejecutar esta alejada de la arquitectura.
  + La lógica y la matemática brindan formas de expresión basada en reglas como el declarativo o lógico y basados en funciones como en el aplicativo.
  + Consisten en determinar que desea calcular y no como se desea hacerlo.
  + Tipos:
    - Funcional: se examina la función que el programa representa y no los cambios de estado de acuerdo a como se ejecuta el programa. Se observa el resultado deseado y no los datos disponibles.
    - Lógico: las declaraciones verifican la existencia de cierta condición habilitadora, cuando se satisface se ejecuta una acción apropiada. Ejecución no secuencial, las condiciones habilitadoras determinan el orden de ejecución.

Clasificación detallada de los paradigmas:

* Paradigma imperativo:
  + Debe su nombre al papel dominante que desempeñan las sentencias imperativas.
  + Calculo iterativo paso a paso de valores de nivel inferior y su asignación a posiciones de memoria.
  + Pueden implementarse muy eficientemente debido a su estrecha con la arquitectura de la computadora.
  + Concepto de celda de memoria para almacenar valores, principal componente de la arquitectura es la memoria, compuesta por gran numero de celdas, estas tienen nombre (variables).
  + Operaciones de asignación, estrechamente ligado a la arquitectura de la memoria, se tiene la idea de que cada valor calculado debe ser almacenado (asignado a una variable).
  + Nociones de celdas de memoria y asignación a bajo nivel se extienden a todos los lenguajes de programación y fuerzan a los programadores un estilo de pensamiento adaptado a la Arquitectura de Von Neumann.
  + Un programa imperativo normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales. La única forma de ejecutar algo complejo es ejecutar una secuencia de instrucciones.
  + Entorno de programación procedimental.
  + Secuencia de control, variable, operación de asignación son sus componentes principales.
  + Abstracción de datos = variables + sentencias de asignación.
  + En la programación imperativa se solía hacer un uso indiscriminado de las instrucciones de saltos condicionales e incondicionales, lo que aumentaba la complejidad del código y dificultaba su modificación.
  + Ej. De lenguajes Fortran, C, Pascal, Prolog, etc.
  + Programación estructurada (técnica Paradigma Imperativo):
    - Eliminar dificultades del paradigma imperativo.
    - Definida como la técnica sin saltos condicionales e incondicionales.
    - Programa estructurado puede ser leído de principio a fin sin interrupciones en la lectura.
    - Mayor claridad del programa.
    - Puesta a punto mas rápida, así como la confección de la documentación.
    - Fácil de leer y comprender.
    - Fácil de codificar en una gran cantidad de lenguajes y de sistemas.
    - Fácil de mantener.
    - Eficiente aprovechar al máximo los recursos de la computadora.
    - Modularizable.
  + Diseño Top Down o diseño descendente (técnica Paradigma Imperativo):
    - Serie de descomposiciones sucesivas del programa inicial que describen el refinamiento progresivo del repertorio de las instrucciones que van a formar parte del programa.
    - Objetivo: Simplificar el problema.
    - Los subprogramas resultantes de cada descomposición, pueden ser programados de manera independiente, incluso por diferentes personas.
    - Programa final queda estructurado en forma de módulos.
  + Limitaciones:
    - Dificultad para razonar una solución:
      * Alta dependencia de la arquitectura
      * Una variable es una abstracción de una celda de memoria.
    - No se cuenta con Transparencia referencial.
    - Efectos colaterales y aliasing.
    - Accesibilidad vulnerable.
    - Estructuras de control desdibujan la lógica del programa.
  + Falla fundamental de la programación imperativa eran las **variables globales**, ya que pueden accederse y actualizarse desde cualquier parte del programa.
* Paradigma Orientado a Objetos:
  + Extensión del paradigma imperativo, surge como una solución a los problemas que este paradigma presentaba.
  + Código reutilizable y ampliable.
  + Reconoce las entidades del problema, entidad es un objeto caracterizado por atributos y comportamiento, de acuerdo a su propósito y habilidades.
  + Programa en POO: conjunto de objetos que interactuan a través del paso de mensajes, por eso se describe con el termino Objeto-Mensaje, cada objeto es solicitado para realizar un determinado servicio mediante el envió a ese objeto de un mensaje apropiado. El que solicita el servicio no tiene idea de como esta implementado el objeto, importa el que se puede obtener y no como.
  + Se basa en la idea de cooperación entre objetos
  + Objeto:
    - Elemento autónomo de información.
    - Creado en tiempo de ejecución.
    - Entidades que combinan datos y procedimientos que operan sobre esos datos.
    - Poseen estado, comportamiento e identidad
    - Estructura abstracta que describe con la mayor fiabilidad posible un objeto del mundo real y como se relaciona e interactua con el resto del mundo que lo rodea a través de interfaces.
  + Propiedades:
    - Abstracción de datos: permite a una clase definir un patrón de comportamiento a partir de la cual crear varias instancias.
    - Encapsulamiento: permite asegurar que el contenido de la información de un objeto esta oculta al interior. Representación física de la clase y las implementaciones invisibles desde el exterior.
    - Modularidad: división de una aplicación en partes mas pequeñas.
    - Jerarquía: clasificar u u ordenar las abstracciones.
  + Conceptos claves:
    - Objetos.
    - Clases y subclases:
      * Son la descripción abstracta de un grupo de objetos con características comunes
      * Comparten estructura y comportamiento.
    - Herencia:
      * Por especialización(una subclase representa una especialización de la clase superior) o por generalización(una superclase representa un generalización del las subclases).
      * Permite a los objetos ser construidos a partir de otros objetos.
      * Clase base y jerarquía de clases que contienen las clases derivadas de la clase base.
      * Se heredan atributos y métodos.
      * Reduce la redundancia y fomenta la reusabilidad en un sistema.
    - Polimorfismo por inclusión: capacidad de que objetos de diferentes clases respondan al mismo mensaje. Mismo mensaje actúa de distinto modo cuando actua sobre objetos diferentes ligados por herencia.
  + El lenguaje puro que representa al paradigma es Smalltalk. Pero también hay otros como C++, Java, etc.
* Paradigma Funcional:
  + Basado en el modelo matematico de composicion funcional, donde el resultado de un calculo es la entrada del siguiente y asi sucesivamente hasta obtener el resultado deseado.
  + Fundamento de la programación funcional es el calculo lamda, enunciado por Church en 1941.
  + Para programar se construyen funciones complejas sobre funciones sencillas construidas previamente que se aplican a los datos de entrada, no existiendo la idea de variable o asignación de memoria.
  + No hay una noción implícita de estado, por eso no hay necesidad de instrucciones de asignación.
  + Un programa sera una función, que **transformara un input en un output,** dependiendo el output del input. Todo esto en base a evaluación de expresiones, no mediante la ejecución de sentencias.
  + Conceptos claves:
    - Expresiones.
    - Funciones.
    - Polimorfismo paramétrico.
    - Abstracción de datos.
    - Evaluación perezosa.
  + Componentes:
    - Conjunto de objetos, miembros del dominio, rango de las funciones y dependen de las características del lenguaje.
    - Funciones primitivas: son el conjunto de funciones que provee el lenguaje como predefinida y dependen del dominio de aplicación del lenguaje.
    - Formas funcionales:
      * Funciones de primer orden.
      * Funciones de orden superior: funciones cuyo argumento o resultado es una función y tienen un alto impacto en la expresividad del lenguaje, en conjunto con las funciones primitivas, constituyen la base para construir otras funciones.
    - Operación de aplicación: principal mecanismo de control, determina como las funciones se llaman entre si y como se aplican los argumentos a una función determinada.
  + Propiedades:
    - Semántica de valores: desaparece la noción de estado de la arquitectura subyacente. No existe el concepto de locación de memoria. Variables se asignan a valores, no a locaciones de memoria.
    - Transparencia referencial: una funcion computa diferentes resultados unicamente a partir de sus argumentos. Misma entrada, misma salida.
    - La semántica del sistema puede determinarse a partir de la semántica de sus partes sin importar computaciones previas y el orden de evaluación de las mismas
    - Funciones como valor de primera clase: funciones pueden ser pasadas como argumentos, retornadas como resultados.