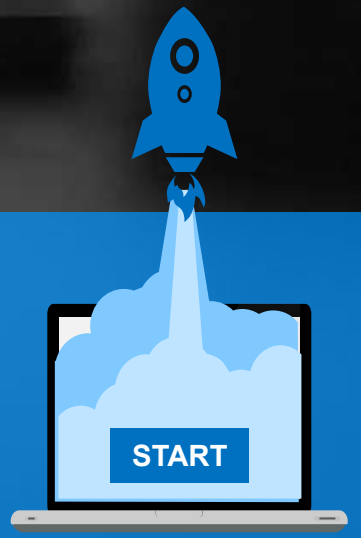
BIENVENIDOS

lsftn151 | AS | AYED II



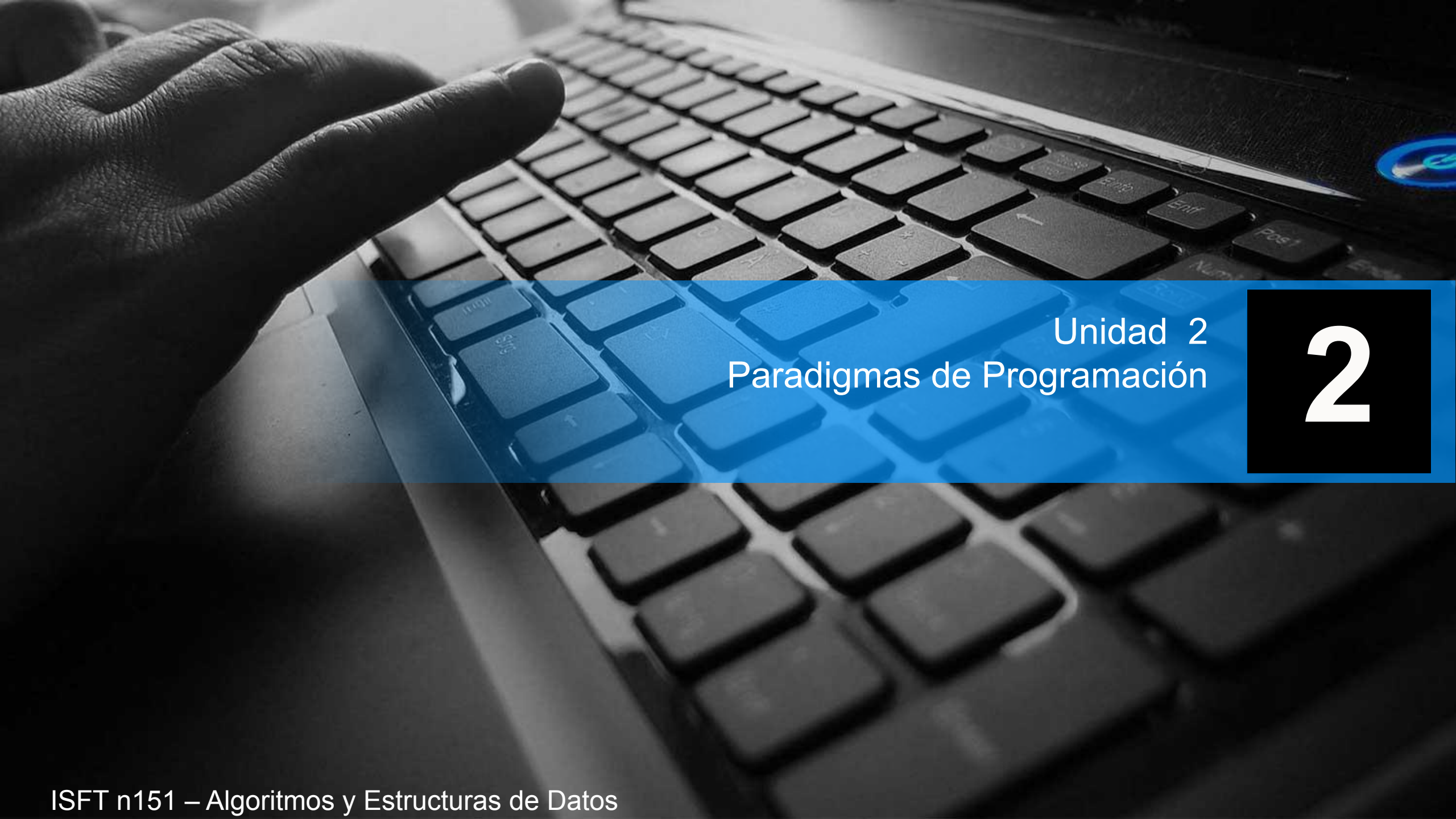
Un Momento por favor.

Hoy Comenzaremos este Tramo de formación juntos.
Sabemos que no será sencillo, pero material de sobra tenemos.

Bienvenidos

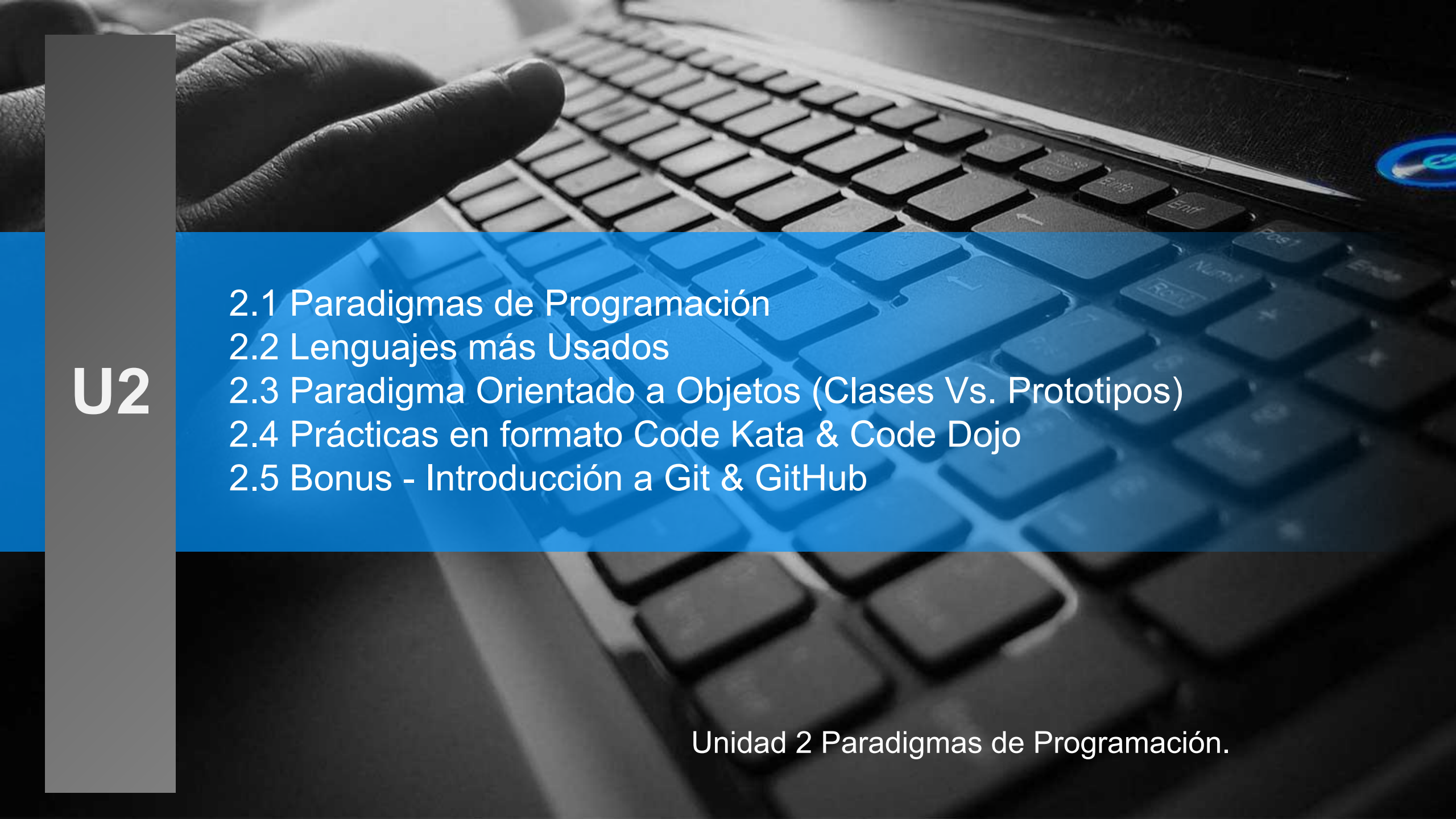
Vamos a poner a vuestra disposición una batería de conocimientos que sin dudarlo, les servirán para desempeñarse en el futuro laboral.

Acompañennos en este tramo poniendo lo mejor de Ustedes… para Ustedes.



Unidad 2
Paradigmas de Programación

2



U2

2.1 Paradigmas de Programación
2.2 Lenguajes más Usados
2.3 Paradigma Orientado a Objetos (Clases Vs. Prototipos)
2.4 Prácticas en formato Code Kata & Code Dojo
2.5 Bonus - Introducción a Git & GitHub

Unidad 2

Paradigmas de Programación

2.1

Algoritmos y Estructuras de Datos II

Paradigmas y Algo
Más…

Cualquier tonto puede escribir un código que una computadora pueda entender. Los buenos programadores escriben código que los humanos pueden entender.

- Martin Fowler

Martin Fowler es un ingeniero de software británico, autor y orador internacional sobre desarrollo de software, especializado en análisis y diseño orientado a objetos, UML, patrones de diseño, y metodologías de desarrollo ágil, incluyendo programación extrema.

“Algo…” sobre Calidad de software

Cuando se habla de la calidad del software profesional, se debe considerar que el software lo usan y cambian personas, además de sus desarrolladores. En consecuencia, la calidad no tiene que ver sólo con lo que hace el software. En cambio, debe incluir el comportamiento del software mientras se ejecuta, y la estructura y organización de los programas del sistema y la documentación asociada. Esto se refleja en los llamados atributos de calidad no funcionales del software. Ejemplos de dichos atributos son el tiempo de respuesta del software ante la duda de un usuario y la comprensibilidad del código del programa. El conjunto específico de atributos que se espera de un sistema de software depende evidentemente de su aplicación. Así, un sistema bancario debe ser seguro, un juego interactivo debe tener capacidad de respuesta, un sistema de conmutación telefónica debe ser confiable, etcétera.

“EL Problema de la Complejidad”

“EL Problema de la Complejidad”

Al programar, la complejidad es siempre el enemigo. Los programas de gran complejidad, con muchas partes móviles y componentes interdependientes, parecen inicialmente impresionantes. Sin embargo, la capacidad de traducir un problema del mundo real en una solución simple o elegante requiere una comprensión más profunda.

Mientras desarrollamos una aplicación o resolvemos un problema simple, a menudo decimos "Si tuviera más tiempo, habría escrito un programa más simple". La razón es que hicimos un programa con mayor complejidad.

“Cuanto menos complejidad tengamos, más fácil será depurar y comprender. Cuanto más complejo se vuelve un programa, más difícil es trabajar en él.”
La gestión de la complejidad es la principal preocupación de un programador .

Entonces, ¿cómo lidian los programadores con la complejidad?
Hay muchos enfoques generales que reducen la complejidad de un programa o lo hacen más manejable.
Uno de los enfoques principales es un paradigma de programación.

¡ Los Invito a Sumérjase en los paradigmas de programación!

¿Qué es exactamente un paradigma de programación?

Introducción a los paradigmas de programación

El término paradigma de programación se refiere a un estilo de programación .
No se refiere a un idioma específico, sino a la **forma** en que programa.
Hay muchos lenguajes de programación que son bien conocidos, pero todos necesitan seguir alguna estrategia cuando se implementan. Y esa estrategia es un paradigma.

Paradigmas de Programación

Los paradigmas de programación son los **principios** fundamentales de la programación de software. Lo más fácil es planteárselos como **estilos de programación fundamentalmente "diferenciados"** que, en consecuencia, generan códigos software que están estructurados de forma distinta.

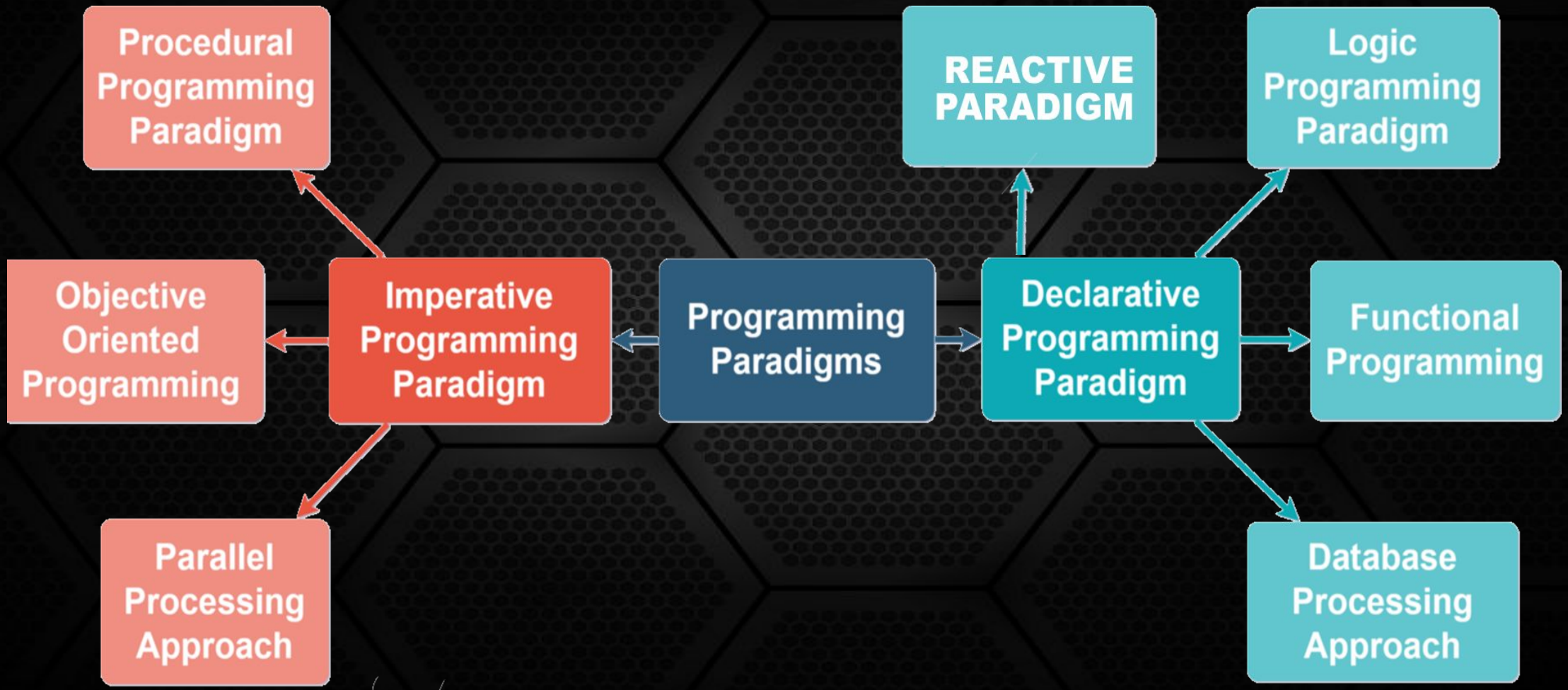
En la **evolución** de la programación han surgido diversas técnicas de programación que se han ido adaptando a las necesidades tecnológicas e informáticas del momento. Aunque la forma de **enfocar la elaboración** de los programas es diferente en cada una de ellas, el objetivo es el mismo:

facilitar la creación y el mantenimiento de programas informáticos.

Estas técnicas se han traducido en diferentes filosofías de creación de programas que son los denominados **paradigmas de programación**.

*Un **paradigma de programación** representa un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de **programas**. Es decir, un **paradigma** es una **filosofía** para la creación de programas.*

Paradigmas de Programación




1
Programación
Imperativa

1 Programación imperativa

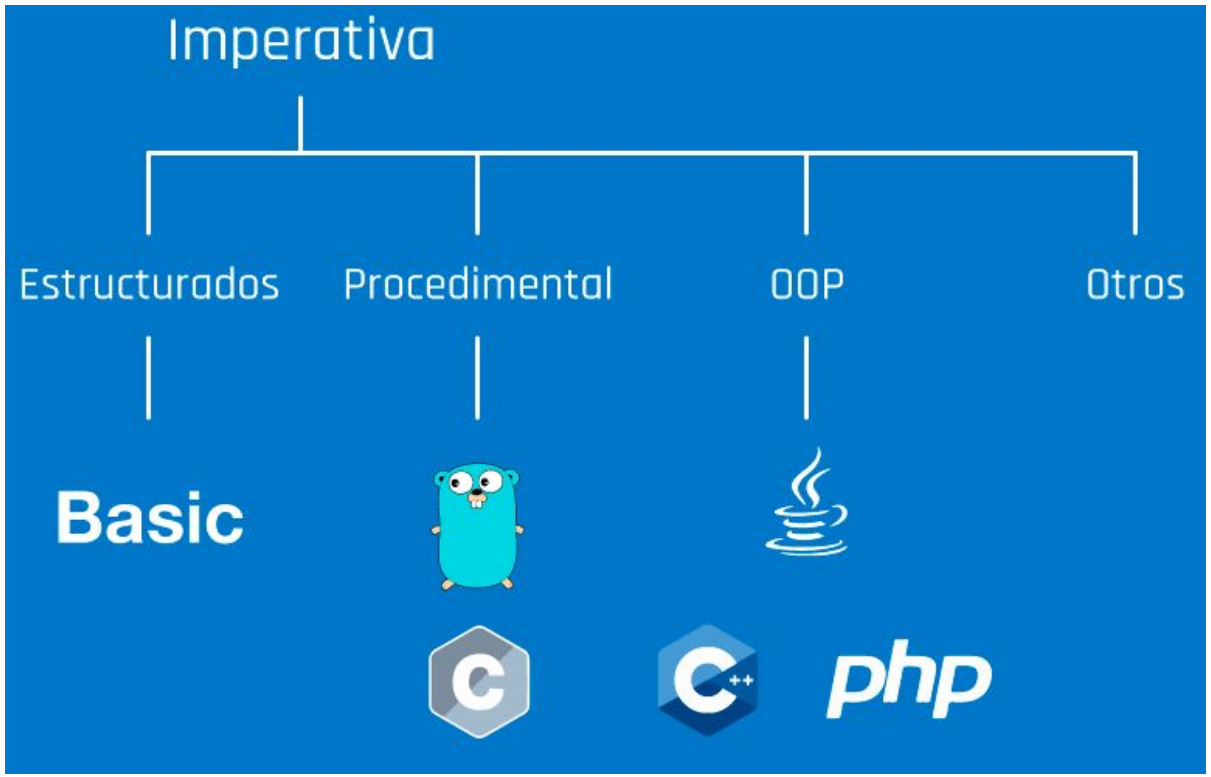
1 procesamiento imperativo

1.1 Programación procedimental

1.2 programación orientada a objetos

1.3 procesamiento paralelo

Veámoslo en más detalle…


Imperativa
Estructurados
Procedimental
OOP
Otros
Basic
C
C++
php

1 Paradigma de programación imperativo

La palabra "imperativo" proviene del latín "impero" que significa "yo mando".
Es la misma palabra de la que obtenemos "emperador", y eso es bastante apropiado. Eres el emperador. Le das a la computadora pequeñas órdenes para que las haga y las hace una a la vez y te informa.
El paradigma consta de varios enunciados, y luego de la ejecución de todos ellos, se almacena el resultado. Se trata de escribir una lista de instrucciones para decirle a la computadora qué hacer paso a paso.
En un paradigma de programación imperativa, el orden de los pasos es crucial, porque un paso dado tendrá diferentes consecuencias dependiendo de los valores actuales de las variables cuando se ejecuta el paso.
Para ilustrar, busquemos la suma de los diez primeros números naturales en el enfoque del paradigma imperativo.

```c
#include <stdio.h>

int main()
{
    int sum = 0;
    sum += 1;
    sum += 2;
    sum += 3;
    sum += 4;
    sum += 5;
    sum += 6;
    sum += 7;
    sum += 8;
    sum += 9;
    sum += 10;

    printf("The sum is: %d\n", sum); //prints-> The sum is 55

    return 0;
}
```

En el ejemplo anterior, le estamos ordenando a la computadora qué hacer línea por línea. Finalmente, almacenamos el valor y lo imprimimos

1 Programación imperativa

1.1 Programación procedimental

Veámoslo en más detalle…

1.1 Paradigma de programación procedimental

La programación procedimental (que también es imperativa) **permite dividir esas instrucciones en procedimientos** .

NOTA: Los procedimientos no son funciones. La diferencia entre ellos es que las funciones devuelven un valor y los procedimientos no.
Más específicamente, las funciones están diseñadas para tener efectos secundarios mínimos y siempre producen la misma salida cuando se les da la misma entrada. Los procedimientos, por otro lado, no tienen ningún valor de retorno. Su propósito principal es realizar una tarea determinada y provocar un efecto secundario deseado.

Un gran ejemplo de procedimientos sería el conocido bucle for. El propósito principal del bucle for es causar efectos secundarios y no devuelve un valor.
Para ilustrar, encontremos la suma de los primeros diez números naturales en el enfoque del paradigma procedimental.

```c
#include <stdio.h>

int main()
{
    int sum = 0;
    int i =0;
    for(i=1;i<11;i++){
        sum += i;
    }

    printf("The sum is: %d\n", sum); //prints-> The sum is 55

    return 0;
}
```

En el ejemplo anterior, usamos un bucle for simple para encontrar la suma de los primeros diez números naturales.
Los lenguajes que admiten el paradigma de programación procedimental son:
C
C ++
Java
Fusión fría
Pascal

1.1 La programación por procedimientos suele ser la mejor opción cuando:

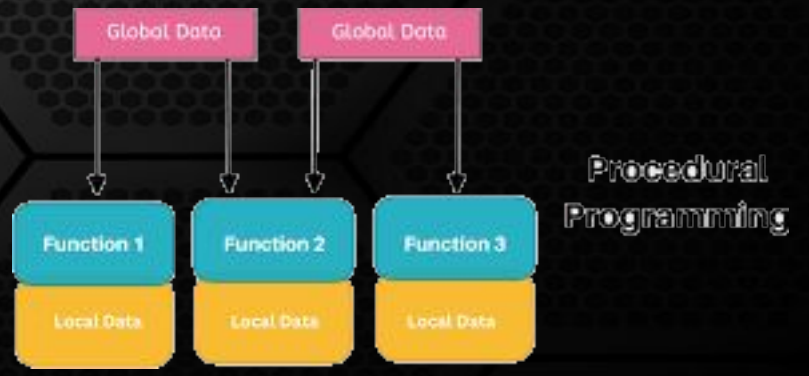
- Existe una operación compleja que incluye dependencias entre operaciones y cuando se necesita una visibilidad clara de los diferentes estados de la aplicación ('carga de SQL', 'carga de SQL', ‘leer en línea', 'sin hardware de audio', etc.). Esto suele ser apropiado para el inicio y el cierre de la aplicación .
- El programa es muy singular y se compartieron pocos elementos
- El programa es estático y no se espera que cambie mucho con el tiempo .
- Se espera que no se agreguen algunas características al proyecto a lo largo del tiempo .

¿Por qué debería considerar aprender el paradigma de la programación procedimental?

Es sencillo, una forma más sencilla de realizar un seguimiento del flujo del programa.
Tiene la capacidad de ser fuertemente modular o estructurado.
Necesita menos memoria: es eficiente y eficaz.

En el ejemplo anterior, usamos un bucle for simple para encontrar la suma de los primeros diez números naturales.
Los lenguajes que admiten el paradigma de programación procedimental son:

Pascal
Java
C ++
C



1 Programación imperativa

1.2 Programación Orientada a Objetos

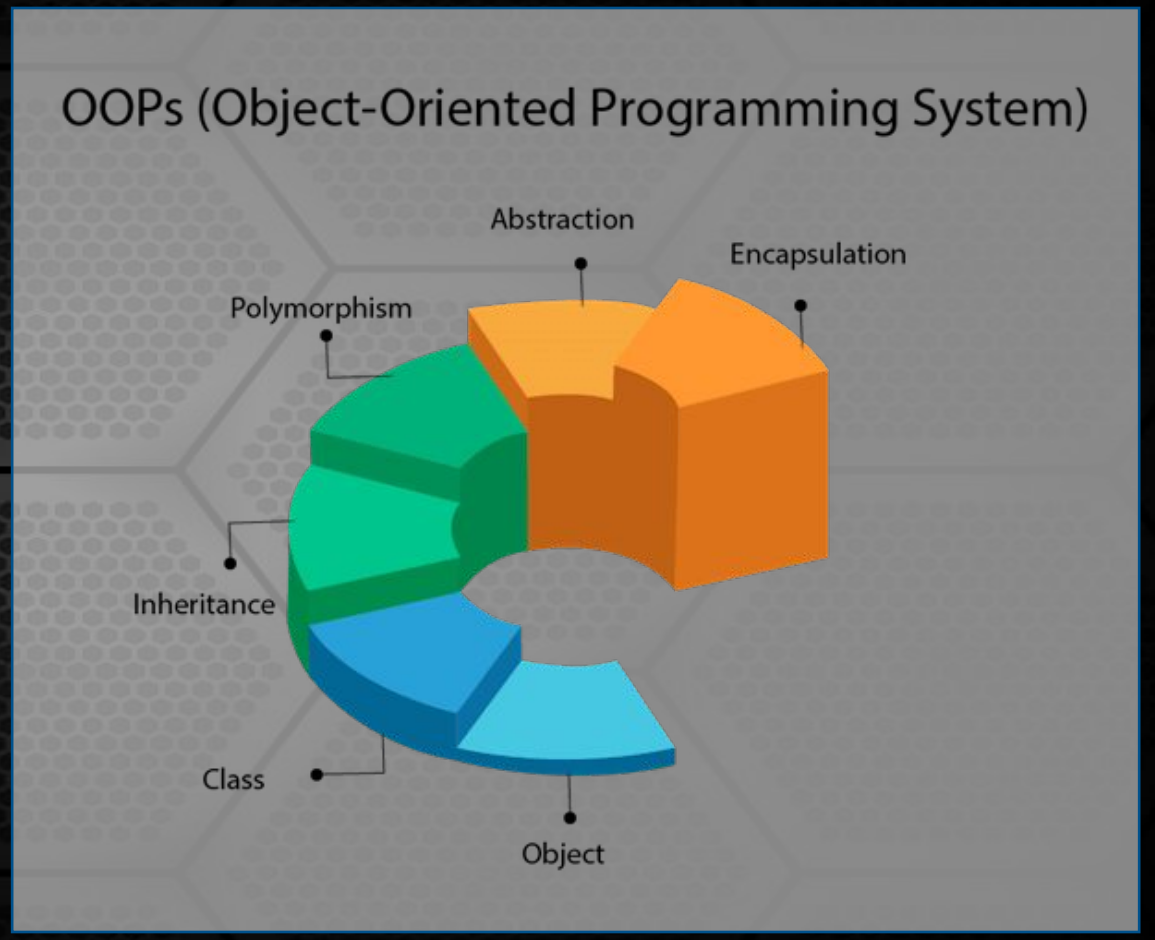
Veámoslo en más detalle…

1.2 Paradigma de programación orientada a objetos

OOP es el paradigma de programación más popular debido a sus ventajas únicas, como la modularidad del código y la capacidad de asociar directamente problemas comerciales del mundo real en términos de código.

La programación orientada a objetos ofrece una forma sostenible de escribir código espagueti. Le permite acumular programas como una serie de parches.
- Paul Graham

Las características clave de la programación orientada a objetos incluyen clase, abstracción, encapsulación, herencia y polimorfismo.
Una **clase** es una plantilla o plano a partir del cual se crean los objetos.



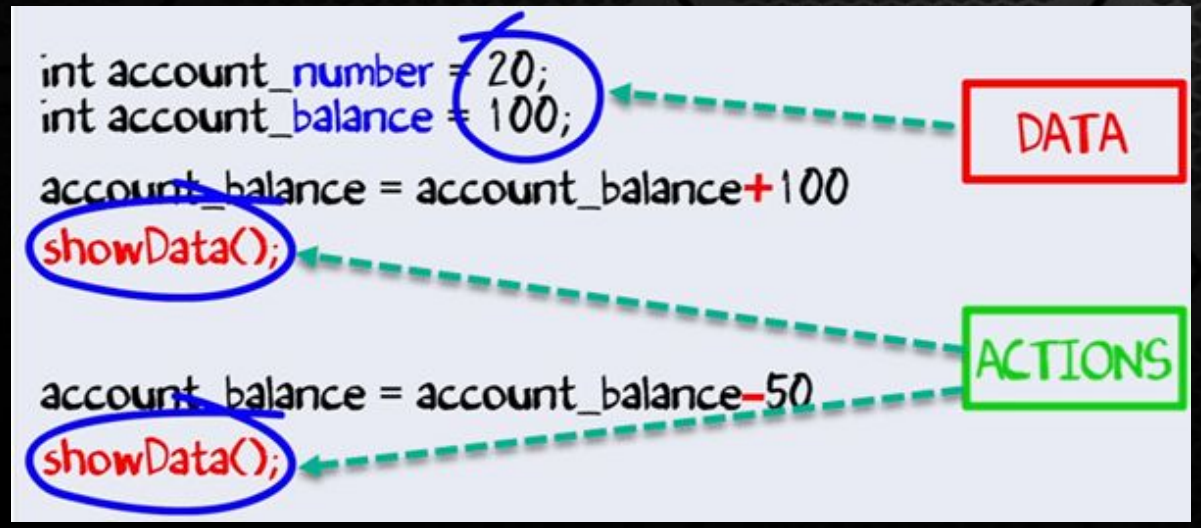
1.2 Paradigma de programación orientada a objetos

Los objetos son instancias de clases. Los objetos tienen atributos / estados y métodos / comportamientos. Los atributos son datos asociados con el objeto, mientras que los métodos son acciones / funciones que el objeto puede realizar.

La abstracción separa la interfaz de la implementación. La encapsulación es el proceso de ocultar la implementación interna de un objeto.
La herencia permite representar y perfeccionar las relaciones jerárquicas. El polimorfismo permite que objetos de diferentes tipos reciban el mismo mensaje y respondan de diferentes formas.

```java
public class Main
{
    public static void main(String[] args) {
        Addition obj = new Addition();
        obj.num = 10;
        int answer = obj.addValues();
        System.out.println("The sum is = "+answer); //prints-> The sum is 55
    }
}

class Addition {
    int sum =0;
    int num =0;
    int addValues(){
        for(int i=1; i<=num;i++){
            sum += i;
        }
        return sum;
    }
}
```



Tenemos una clase Addition que tiene dos estados sum y num que se inicializan a cero. También tenemos un método addValues() que devuelve la suma de numnúmeros.
En la Main clase, hemos creado un objeto, obj de la clase Addition. Luego, inicializamos el numen 10 y llamamos addValues()al método para obtener la suma.
Idiomas que admiten el paradigma orientado a objetos:

Phyton Rubí Java C ++ C# …

1.2 Paradigma de programación orientada a objetos

La programación orientada a objetos se utiliza mejor cuando:

Tiene varios programadores que no necesitan comprender cada componente (Holligan, 2016).
Hay mucho código que se puede compartir y reutilizar (Holligan, 2016).
Se prevé que el proyecto cambiará con frecuencia y se irá ampliando con el tiempo (Holligan, 2016).

¿Por qué debería considerar aprender el paradigma de la programación orientada a objetos?

- Reutilización de código por herencia.
- Flexibilidad a través del polimorfismo.
- Alta seguridad con el uso de ocultación de datos (encapsulación) y mecanismos de abstracción.
- Productividad mejorada del desarrollo de software: un programador orientado a objetos puede unir nuevos objetos de software para crear programas completamente nuevos (The Saylor Foundation, nd).
- Desarrollo más rápido: la reutilización permite un desarrollo más rápido (The Saylor Foundation, nd).
- Menor costo de desarrollo: la reutilización de software también reduce el costo de desarrollo. Por lo general, se pone más esfuerzo en el análisis y diseño orientado a objetos (OOAD), lo que reduce el costo general de desarrollo (The Saylor Foundation, nd).
- Software de mayor calidad: el desarrollo más rápido de software y el menor costo de desarrollo permiten utilizar más tiempo y recursos en la verificación del software. La programación orientada a objetos tiende a resultar en software de mayor calidad (The Saylor Foundation, sf).

Alena Holligan. (2016). When to use OOP over procedural coding

1 Programación imperativa

1.3 Programación Paralelo

Veámoslo en más detalle…

1.3 Enfoque de procesamiento paralelo

El procesamiento paralelo es el procesamiento de las instrucciones del programa dividiéndolas entre varios procesadores. Un sistema de procesamiento paralelo permite que muchos procesadores ejecuten un programa en menos tiempo dividiéndolos.

Idiomas que admiten el enfoque de procesamiento paralelo:

NESL (uno de los más antiguos)
C
C ++

El enfoque de procesamiento paralelo suele ser el mejor uso cuando:

- Tiene un sistema que tiene más de una CPU o procesadores multinúcleo que se encuentran comúnmente en las computadoras de hoy.
- Necesita resolver algunos problemas computacionales que tardan horas / días en resolverse incluso con el beneficio de un microprocesador más potente. Trabaja con datos del mundo real que necesitan simulación y modelado más dinámicos.

1.3 Enfoque de procesamiento paralelo

¿Por qué debería considerar aprender el enfoque de procesamiento paralelo?

Acelera el rendimiento.
Se utiliza a menudo en Inteligencia Artificial.
Facilita la resolución de problemas, ya que este enfoque parece ser un método de divide y vencerás.

A continuación, se muestran algunos recursos útiles para obtener más información sobre el procesamiento en paralelo:

Obtenga más información aquí: Inteligencia artificial y procesamiento paralelo por Seyed H. Roosta.

Programación paralela en C por Paul Gribble
Introducción a la programación paralela con MPI y OpenMP por Charles Augustine
INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA CON MPI Y OPENMP por Benedikt Steinbusch

2 Programación Declarativa

2 Programación Declarativa

programación lógica

programación funcional

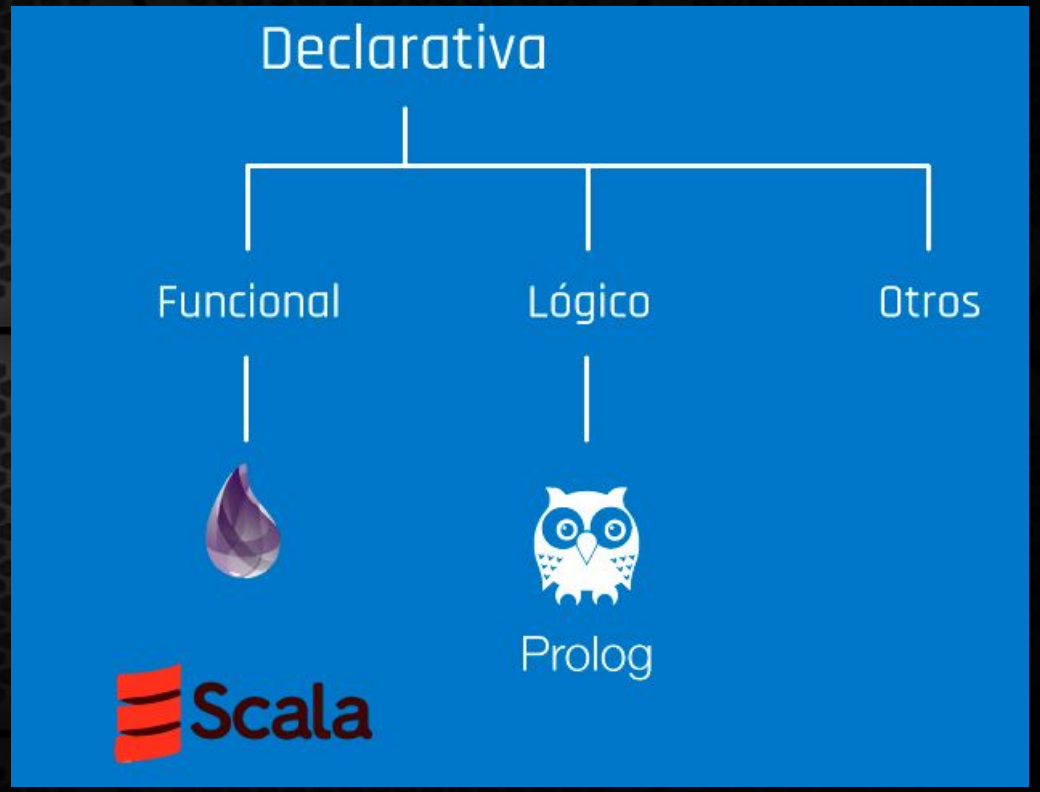
procesamiento de la base de datos

Programación Reactiva

Veámoslo en más detalle…

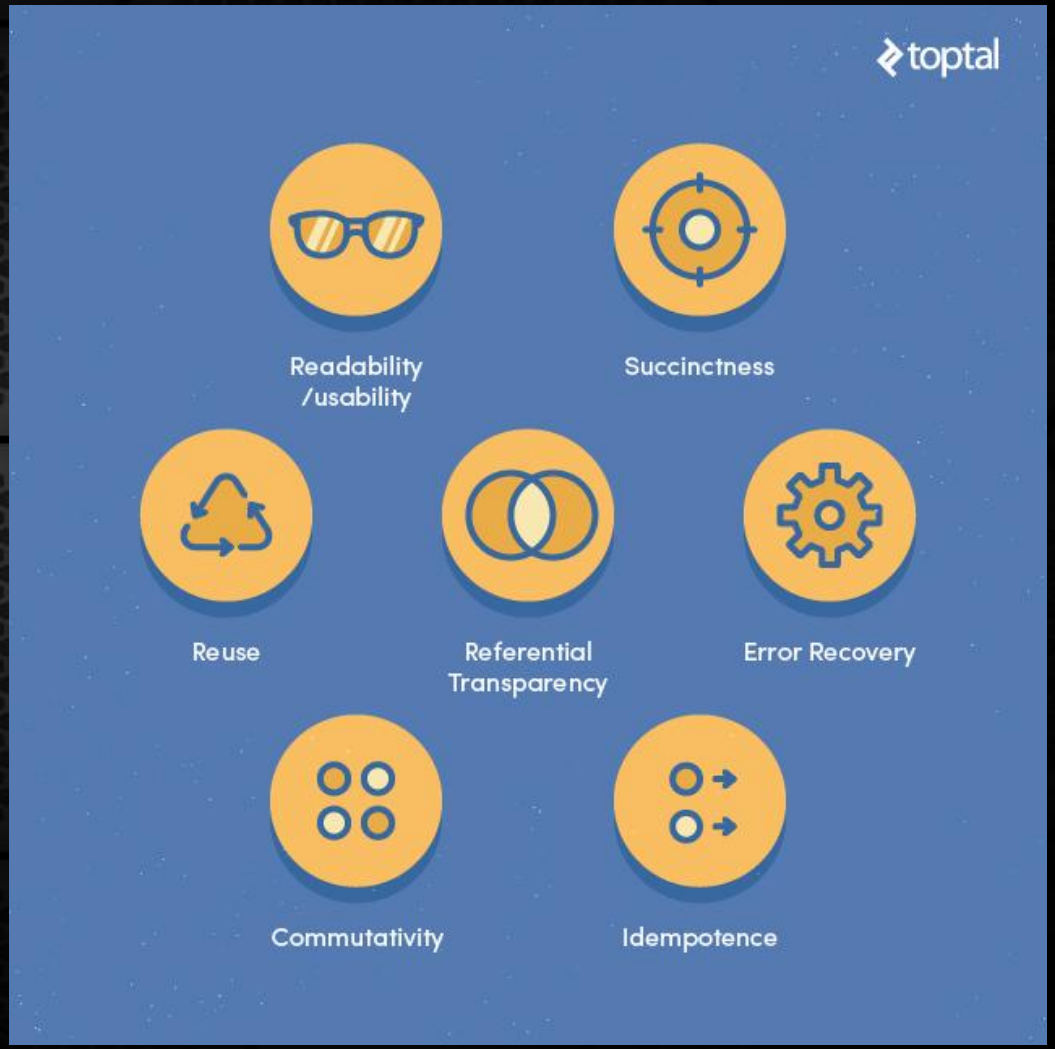
Programación declarativa: paradigmas de software del pasado más reciente

De forma paralela a la evolución continuada del hardware y el software, con el enfoque declarativo se desarrolló un paradigma alternativo para la programación de código. El principio fundamental de la programación declarativa radica en la **descripción del resultado final que se busca**. Por lo tanto, en primera línea se encuentra el **"qué"** del resultado y no **el "cómo" de los pasos que llevan a la solución**, como es el caso en la programación imperativa. Esto provoca que el código de la programación declarativa sea más difícil de comprender debido al **alto grado de abstracción**, aunque resulta muy corto y preciso. Dentro de los subtipos del paradigma de programación declarativa, existen más diferencias que dentro del estilo imperativo. Asimismo, su definición y clasificación no es siempre exacta. Los dos enfoques más importantes del paradigma de programación declarativa son la **programación funcional** y la **lógica**.



2. Paradigma de programación declarativa

La programación declarativa es un estilo de creación de programas que ***expresa la lógica de un cálculo sin hablar de su flujo de control.***
La programación declarativa es un paradigma de programación en el que el programador define lo **que** debe lograr el programa sin definir **cómo** debe implementarse. En otras palabras, el enfoque se centra en lo que se necesita lograr en lugar de instruir cómo lograrlo.
Imagínense al presidente durante el estado de la unión declarando sus intenciones de lo que quieren que suceda. Por otro lado, la programación imperativa sería como un administrador de una franquicia de McDonald's. Son muy imperativos y, como resultado, esto hace que todo sea importante. Por lo tanto, les dicen a todos cómo hacer todo, hasta las acciones más simples.
Entonces, las principales diferencias son que el imperativo te dice **cómo hacer algo** y el declarativo te dice **qué hacer** .



2. Paradigma de programación declarativa

La programación declarativa es, actualmente, el paradigma dominante de un conjunto extenso y diverso de dominios como bases de datos, plantillas y gestión de la configuración.
En pocas palabras, la programación declarativa consiste en instruir a un programa sobre *lo que hay que* hacer, en lugar de decirle *cómo* hacerlo. En la práctica, este enfoque implica proporcionar un lenguaje específico de dominio (DSL) para expresar *lo que* el usuario quiere y protegerlo de las construcciones de bajo nivel (bucles, condicionales, asignaciones) que materializan el estado final deseado.

Podría decirse que la herramienta de programación declarativa más exitosa es la base de datos relacional (RDB). Incluso podría ser la primera herramienta declarativa. En cualquier caso, los RDB exhiben las dos propiedades que considero arquetípicas de la programación declarativa:

- **Un lenguaje específico de dominio (DSL):** la interfaz universal para bases de datos relacionales es un DSL llamado Structured Query Language, más comúnmente conocido como SQL.

- **El DSL oculta la capa de nivel inferior al usuario**: desde el artículo original de Edgar F. Codd sobre los RDB, está claro que el poder de este modelo es disociar las consultas deseadas de los bucles subyacentes, índices y rutas de acceso que los implementan.

Antes de los RDB, se accedía a la mayoría de los sistemas de bases de datos a través de código imperativo, que depende en gran medida de detalles de bajo nivel, como el orden de los registros, los índices y las rutas físicas a los datos en sí. Debido a que estos elementos cambian con el tiempo, el código a menudo deja de funcionar debido a algún cambio subyacente en la estructura de los datos. El código resultante es difícil de escribir, difícil de depurar, difícil de leer y difícil de mantener. Voy a salir de un limbo y decir que la mayor parte de este código estaba, con toda probabilidad, largo, lleno de proverbiales nidos de ratas de condicionales, repetición y errores sutiles dependientes del estado.
Frente a esto, los RDB proporcionaron un tremendo salto de productividad para los desarrolladores de sistemas. Ahora, en lugar de miles de líneas de código imperativo, tenía un esquema de datos claramente definido, además de cientos (o incluso solo decenas) de consultas. Como resultado, las aplicaciones solo tenían que lidiar con una representación abstracta, significativa y duradera de los datos, e interactuar con ellos a través de un lenguaje de consulta potente pero simple. El RDB probablemente elevó la productividad de los programadores, y las empresas que los empleaban, en un orden de magnitud.

2. Paradigma de programación declarativa

1. **Legibilidad/usabilidad:** un DSL usualmente se acerca más a un lenguaje natural (como el Inglés o Español) que a un pseudocódigo y por ello es más fácil de leer y aprender por personas que no son programadores.

2. **Concisión:** la mayor parte de la tablatura es extraída por el DSL, dejando así menos líneas para hacer el mismo trabajo.

3. **Reutilización:** es más fácil crear un código que puede ser usado para diferentes propósitos; algo que todos saben es extremadamente difícil cuando se usan construcciones imperativas.

4. **Idempotencia:** puedes trabajar con *estados finales* y dejar que el programa haga el resto. Por ejemplo, en una operación de *upsert* puedes insertar una fila si no está ahí o puedes modificarla si ya se encuentra ahí, esto en lugar de escribir un código que se encargue de ambos casos.

5. **Error de Recuperación:** es fácil identificar una construcción que se detendrá ante el primer error, en lugar de tener que añadir listados de errores para cada posible error. (Si alguna vez has escrito 3 retrollamadas o *callbacks* anidadas en node.js entonces sabes de lo que habló.)

6. **Transparencia Referencial:** aunque esta ventaja se asocia más a la programación funcional, la verdad es que es válida para cualquier enfoque que minimice la manipulación manual del estado y se base en efectos secundarios.

7. **Conmutatividad:** la posibilidad de expresar un estado final sin que se especifique el orden real en el que se implementará.

2 Programación Declarativa

2.1 Paradigma de programación lógica

Veámoslo en más detalle…

2.1 (Declativo) Paradigma de programación lógica

El paradigma de la programación lógica adopta un enfoque declarativo para la resolución de problemas. Está basado en una lógica formal. El paradigma de la programación lógica no se compone de instrucciones, sino de hechos y cláusulas. Utiliza todo lo que sabe e intenta llegar a un mundo en el que todos esos hechos y cláusulas sean ciertos. Por ejemplo, Sócrates es un hombre, todos los hombres son mortales y, por tanto, Sócrates es mortal. El siguiente es un programa Prolog simple que explica la instancia anterior:

```
man(Socrates).
mortal(X) :- man(X).
```

La primera línea se puede leer, "Sócrates es un hombre". Es una cláusula básica , que representa un hecho simple.
La segunda línea se puede leer, "X es mortal si X es un hombre; '' en otras palabras," Todos los hombres son mortales. '' Esta es una cláusula , o la regla, para determinar el momento de su entrada X es "mortal '. '(El símbolo ": -", a veces llamado torniquete , se pronuncia "si").
Podemos probar el programa haciendo la pregunta:

```
/*We're defining family tree
facts*/
father(John, Bill).
father(John, Lisa).
mother(Mary, Bill).
mother(Mary, Lisa).
/*We'll ask questions to Prolog*/
 ?- mother(X, Bill). X = Mary
```

?- mortal(Socrates).

es decir, "¿Es Sócrates mortal?" (El " ?-" es la indicación de la computadora para una pregunta). Prolog responderá " yes". Otra pregunta que podemos hacer es:

?- mortal(X).

Es decir, "¿Quién (X) es mortal?" Prolog responderá " X = Socrates". Para darte una idea, John es el padre de Bill y Lisa. Mary es la madre de Bill y Lisa. Ahora, si alguien hace una pregunta como "¿quién es el padre de Bill y Lisa?" o "¿quién es la madre de Bill y Lisa?" podemos enseñarle a la computadora a responder estas preguntas usando programación lógica.
`X = Mary`
Ejemplo explicado:
`father(John, Bill).`
El código anterior define que John es el padre de Bill. Le preguntamos a Prolog qué valor de X hace que esta afirmación sea verdadera. X debería ser María para que la afirmación sea verdadera. Responderá`X = Mary`
`?- mother(X, Bill). X = Mary`

2.1 (Declativo) Paradigma de programación lógica

Lenguajes que soportan el paradigma de programación lógica:

- Prolog
- Absys
- ALF (lenguaje de programación funcional de lógica algebraica)
- Alicia
- Ciao

El paradigma de programación lógica es a menudo el mejor uso cuando:

Si planea trabajar en proyectos como demostración de teoremas, sistemas expertos, reescritura de términos, sistemas de tipos y planificación automatizada.

¿Por qué debería considerar aprender el paradigma de la programación lógica?

- Fácil de implementar el código.
- La depuración es sencilla.
- Dado que está estructurado usando declaraciones de verdadero / falso, podemos desarrollar los programas rápidamente usando programación lógica.
- Como se basa en el pensamiento, la expresión y la implementación, también se puede aplicar en programas no computacionales.
- Admite formas especiales de conocimiento, como el conocimiento de nivel meta o de orden superior, ya que puede modificarse.

2 Programación Declarativa

2.2 Paradigma de programación funcional

Veámoslo en más detalle…

2.2 (Declativo) Paradigma de programación funcional

El paradigma de programación funcional ha estado en el centro de atención durante un tiempo debido a **JavaScript**, un lenguaje de programación funcional que ha ganado más popularidad recientemente.
El paradigma de la programación funcional tiene sus raíces en las matemáticas y es independiente del lenguaje. El principio clave de este paradigma es la ejecución de una serie de funciones matemáticas.

Tú compones tu programa de funciones cortas. Todo el código está dentro de una función. Todas las variables tienen como ámbito la función.

En el paradigma de programación funcional, las funciones no modifican ningún valor fuera del alcance de esa función y las funciones en sí mismas no se ven afectadas por ningún valor fuera de su alcance.
Para ilustrar, identifiquemos si el número dado es primo o no en el paradigma de programación funcional.

```
function isPrime(number){
 for(let i=2; i<=Math.floor(Math.sqrt(number)); i++){
  if(number % i == 0 ){
   return false;
  }
 }
 return true;
}
isPrime(15); //returns false
```

2.2 (Declativo) Paradigma de programación funcional

En el ejemplo anterior, hemos utilizado funciones matemáticas `Math.floor()` y `Math.sqrt()` para resolver nuestro problema de manera eficiente. Podemos resolver este problema sin usar funciones matemáticas integradas de JavaScript, pero para ejecutar el código de manera eficiente se recomienda utilizar funciones JS integradas.
`Number` tiene el alcance de la función `isPrime()` y no se verá afectado por ningún valor fuera de su alcance. `isPrime()` La función siempre produce la misma salida cuando se le da la misma entrada.

NOTA: no hay bucles for y while en la programación funcional. En cambio, los lenguajes de programación funcionales se basan en la **recursividad para la iteración** (Bhadwal, 2019).
Lenguajes que soportan el paradigma de programación funcional:

- Haskell
- OCaml
- Scala
- Clojure
- Raqueta
- **JavaScript**

El paradigma de programación funcional a menudo se usa mejor cuando:

- Trabajar con cálculos matemáticos.
- Trabajar con aplicaciones orientadas a la concurrencia o paralelismo.
- **¿Por qué debería considerar aprender el paradigma de programación funcional?**
- Las funciones se pueden codificar rápida y fácilmente.
- Las funciones de propósito general se pueden reutilizar, lo que conduce a un rápido desarrollo de software.
- Las pruebas unitarias son más fáciles.
- La depuración es más sencilla.
- La aplicación general es menos compleja ya que las funciones son bastante sencillas.

2 Programación Declarativa

2.3 Paradigma de Bases de Datos

Veámoslo en más detalle…

2.3 Enfoque de procesamiento de la base de datos

Esta metodología de programación se basa en datos y su movimiento. Las declaraciones de programa se definen por datos en lugar de codificar una serie de pasos.
Una base de datos es una colección organizada de información estructurada, o datos, que normalmente se almacenan electrónicamente en un sistema informático. Una base de datos suele estar controlada por un sistema de gestión de bases de datos (DBMS) ("Qué es una base de datos", Oracle, 2019).
Para procesar los datos y consultarlos, las bases de datos utilizan **tablas** . A continuación, se puede acceder a los datos, gestionarlos, modificarlos, actualizarlos, controlarlos y organizarlos fácilmente.
Un buen enfoque de procesamiento de bases de datos es crucial para cualquier empresa u organización. Esto se debe a que la base de datos almacena todos los detalles pertinentes sobre la empresa, como registros de empleados, registros de transacciones y detalles salariales.
La mayoría de las bases de datos utilizan el lenguaje de consulta estructurado (SQL) para escribir y consultar datos.
Aquí hay un ejemplo en el enfoque de procesamiento de bases de datos (SQL):

La `PersonID`columna es de tipo int y contendrá un número entero. Los `LastName`, `FirstName`, `Address`, y `City`las columnas son de tipo VARCHAR y sostendrán caracteres, y la longitud máxima para estos campos es de 255 caracteres.

```
CREATE DATABASE personalDetails;
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
```

2.3 Enfoque de procesamiento de la base de datos

El enfoque de procesamiento de bases de datos a menudo se usa mejor cuando:

- Trabajar con bases de datos para estructurarlas.
- Acceder, modificar, actualizar datos en la base de datos.
- Comunicarse con servidores.

¿Por qué son importantes las bases de datos y por qué debería considerar aprender el enfoque de procesamiento de bases de datos?

- La base de datos maneja una gran cantidad de datos: a diferencia de las hojas de cálculo u otras herramientas, las bases de datos se utilizan para almacenar una gran cantidad de datos a diario.
- Preciso: con la ayuda de funcionalidades integradas en una base de datos, podemos validar fácilmente.
- Fácil actualización de datos: los lenguajes de manipulación de datos (DML) como SQL se utilizan para actualizar datos en una base de datos fácilmente.
- Integridad de los datos: con la ayuda de las comprobaciones de validez integradas, podemos garantizar la coherencia de los datos.

2 Programación Declarativa

2.4 Programación Reactiva

Veámoslo en más detalle…

2.4 La Programación Reactiva

Una introducción

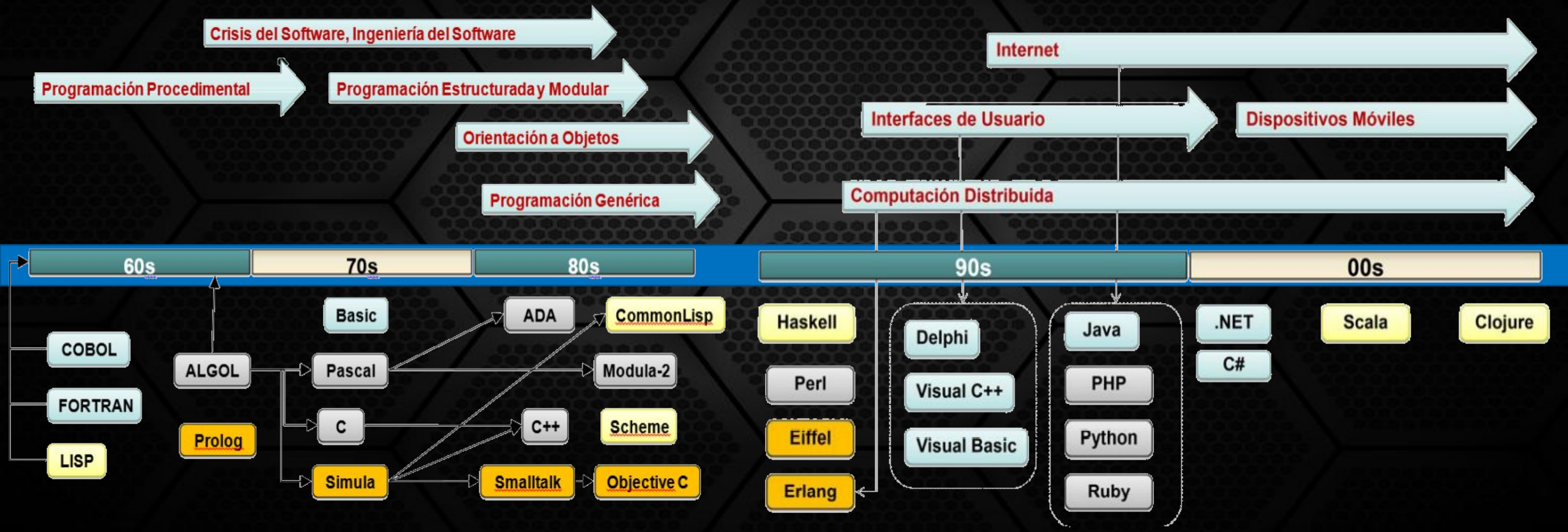
La **programación reactiva** es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos de manera asíncrona. Su concepción y evolución ha ido ligada a la publicación del Reactive Manifesto, que establecía las bases de los sistemas reactivos, los cuales deben ser:

Responsivos: aseguran la calidad del servicio cumpliendo unos tiempos de respuesta establecidos.
Resilientes: se mantienen responsivos incluso cuando se enfrentan a situaciones de error.
Elásticos: se mantienen responsivos incluso ante aumentos en la carga de trabajo.
Orientados a mensajes: minimizan el acoplamiento entre componentes al establecer interacciones basadas en el intercambio de mensajes de manera asíncrona.

La motivación detrás de este nuevo paradigma procede de la **necesidad de responder a las limitaciones de escalado presentes en los modelos de desarrollo actuales**, que se caracterizan por su desaprovechamiento del uso de la CPU debido al I/O, el sobreuso de memoria (enormes thread pools) y la ineficiencia de las interacciones bloqueantes.

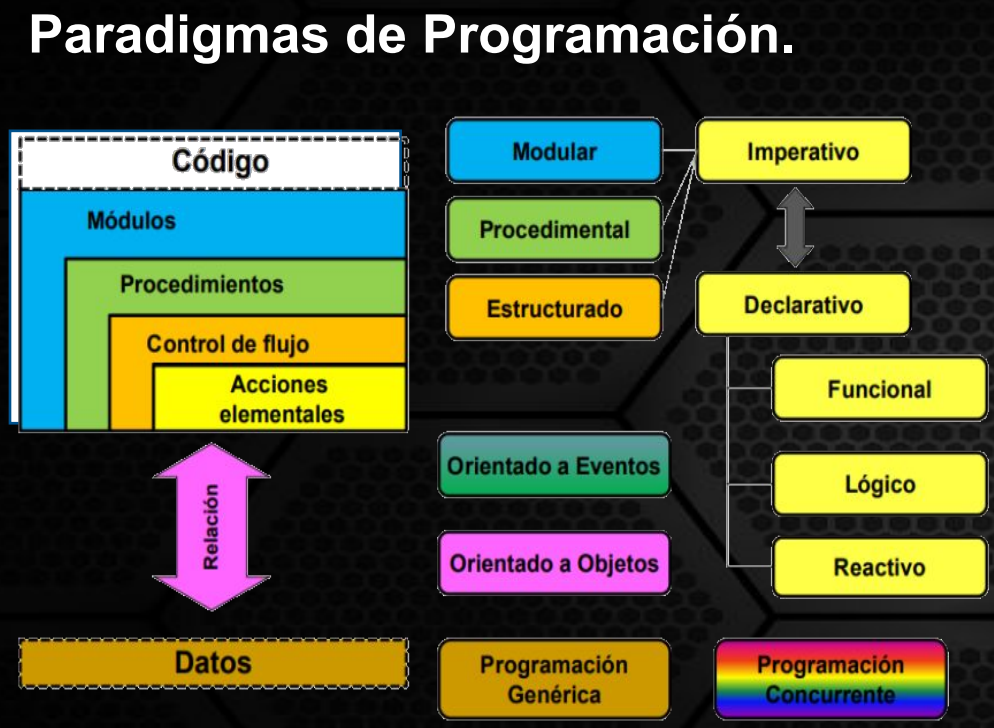
Lenguajes de Programación

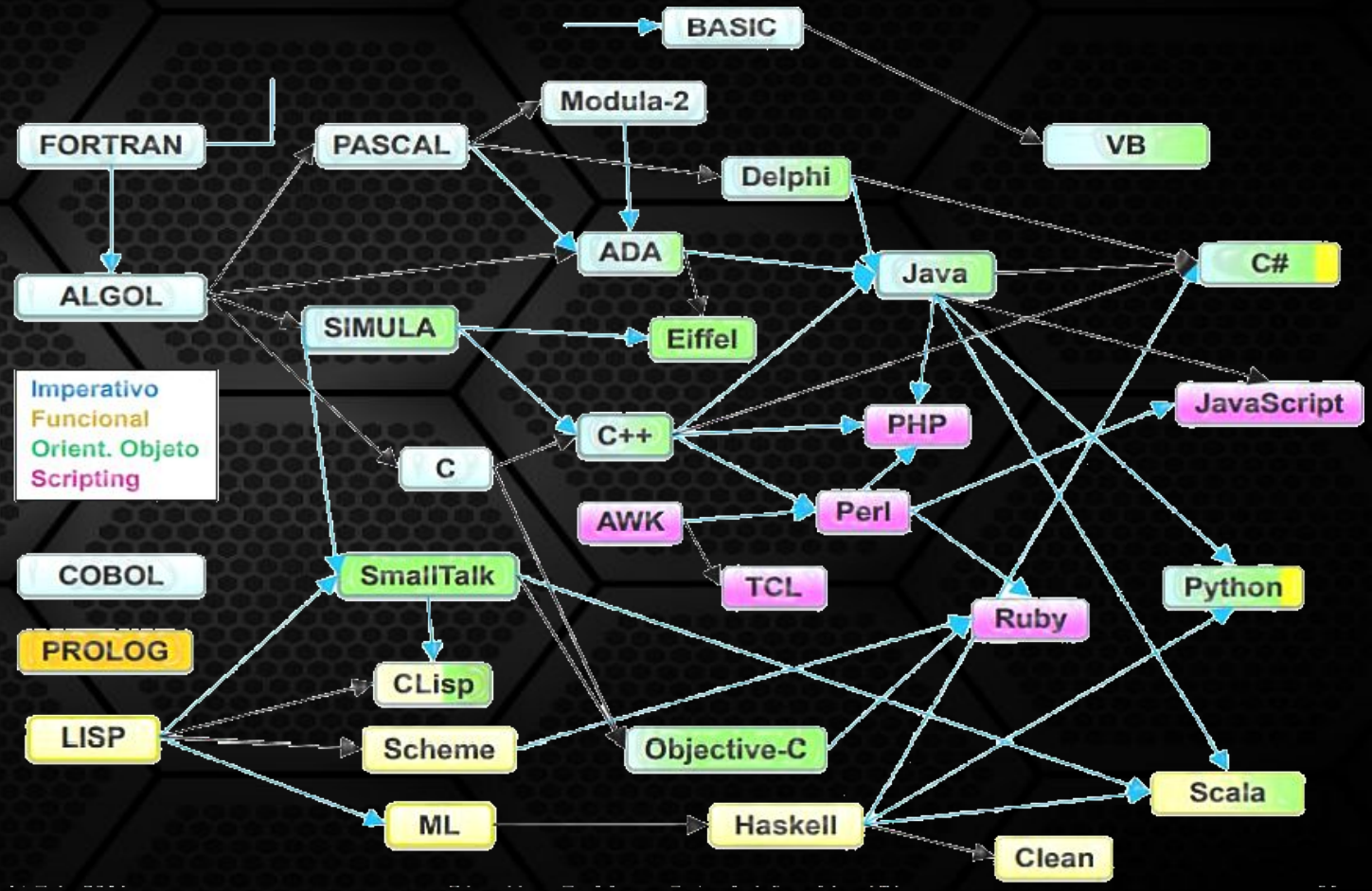
Evolución de los lenguajes de programación



Los lenguajes de programación de alto nivel y los Paradigmas.

Paradigmas de Programación.






MUCHAS GRACIAS
{Hasta la Próxima}

Muchas Gracias

Nos veremos en la Proxima.