Índice

[PROGRAMACIÓN 2](#_Toc134114317)

[¿Qué es un paradigma? 2](#_Toc134114318)

[METODOS DE TRABAJO RECOMENDADOS, ROLES & REGLAS 3](#_Toc134114319)

[PYTHON, PYCHARM 4](#_Toc134114320)

[HYPER TEXT MARKUP LANGUAGE, TEORÍA 5](#_Toc134114321)

[CASCADING STYLE SHEETS, TEORÍA 6](#_Toc134114322)

[JAVASCRIPT, TEORÍA 7](#_Toc134114323)

[JAVA, TEÓRICO 8](#_Toc134114324)

[FRAMEWORKS TEORÍA 9](#_Toc134114325)

[NODE.JS TEORÍA 10](#_Toc134114326)

[C#. TEORÍA 11](#_Toc134114327)

[DATA SCIENCE TEORÍA 12](#_Toc134114328)

[DATA BASE TEORÍA 13](#_Toc134114329)

[GIT 13](#_Toc134114330)

[GITHUB, SOURCETREE Y TEORÍA 14](#_Toc134114331)

[HTTP (HYPERTEXTTRANSFERPROTOCOL), TEORÍA 15](#_Toc134114332)

[RESTFUL, CRUD (CREATE READ UPDATE AND DELETE) FOR DATABASE 17](#_Toc134114333)

[ARQUITECTURA WEB, TEORÍA 18](#_Toc134114334)

[EXPRESS.JS, TEORÍA 19](#_Toc134114335)

[SQL (MYSQL), TEORÍA 20](#_Toc134114336)

[SSL (SECURE SOCKET LAYER) 21](#_Toc134114337)

[SISTEMAS DE AUTOMATIZACION 22](#_Toc134114338)

[C++ 23](#_Toc134114339)

[LIDERAZGO: 25](#_Toc134114340)

[TELEGRAFÍA 27](#_Toc134114341)

[ARDUINO 28](#_Toc134114342)

[Libro: The Clean Code 29](#_Toc134114343)

[Libro: The Clean Coder 30](#_Toc134114344)

[CODE COMPLETE 31](#_Toc134114345)

[SOLID (Single-responsibility, Open-closed, Liskov substitution, Interface segregation y Dependency Inversion principles) 32](#_Toc134114346)

[REFACTORING 33](#_Toc134114347)

[TODAS LAS BUENAS PRACTICAS 34](#_Toc134114348)

# PROGRAMACIÓN

**¿Qué es un paradigma?**USO SIN USO

Se le dice paradigma a un estilo de resolver un problema en programación, un modelo de estructura que tienen los lenguajes de programación.

Y hay de diferentes tipos [Estructurada – Orientada a Objetos – Reactiva – Funciona - Procedural].

**Estructurada:**  Estilo secuencial básico con el que se aprende a programar, un DIAGRAMA DE FLUJO.

**Orientada a Objetos:** Divide los componentes del programa en objetos que tienen datos y comportamiento y se comunican entre sí.

**Reactiva:** Observa flujos de datos asincrónicos y reacciona frente a sus cambios.

**Funcional:** Divide el programa en tareas pequeñas que son ejecutadas por funciones.

Procedural: Estilo que permite desglosar el código en varias porciones, para entender el mismoorientada

**Código Fuente:** Código de un programa que la computadora usa para generar una versión capaz de procesar

**Compilador:** Transformador de un lenguaje de programación a un lenguaje maquina o algún nivel cercano al código binario que procesa el procesador

Interpretador: Programas de la computadora que leen el código fuente de un lenguaje de programación interpretado, convierte una línea de binario a una de código ejecutable

WTF CODE: Cantidad de WTF por programa

# METODOS DE TRABAJO RECOMENDADOS, ROLES & REGLAS

***Métodos***

-**PairPrograming:** Técnica de desarrollo en donde dos personas usan una misma computadora para programar una tarea.

**-Sprint:** Ciclo de tiempo en el cual se llevan a cabo actividades (Programar).

**-Kanban:** Sistema simple y visual para organizar tareas y el flujo de trabajo, Tabla de Kanban.

**-Ágil:** Conjunto de métodos y principios con la idea de desarrollo iterativo, usa equipos autogestionados y multifuncionales

**-Scrum:** Framework para gestión y planificación de productos y proyectos

-Producto Mínimo Viable (**MVP**) [Parece para Empresas]

**Backlog=** Lista de trabajo ordenada por prioridades

***Roles***

Product Owner: El responsable de definir las prioridades del producto backlog y registrar todos los requisitos, reglas y prioridades de Entrega de un producto o proyecto.

YO

-Scrum Master: Responsable de asegurar que todos se adhieran al proceso de desarrollo que sigan sus ceremonias y que garanticen la organización y realización del proyecto.

Yo, Sebas, Mateo

-Developer Team: Equipo de desarrollo, personas que participan en el proyecto.

[10] Yo, Mateo T, Carlos, Joshua, -NA- Maxi, -NA-, -NA-, -NA-, -NA-

[5] Yo, Mateo T, Carlos, Joshua, Maxi [¿12?] Yo

[4] Yo, Mateo T, Carlos, Joshua

***Reglas***

Requisitos sobre cómo debe comportarse una funcionalidad o producto dado un determinado criterio, trabajo en conjunto y convivencia

***Extraído de “Conceptos Informáticos-Glosario”***

# PYTHON, PYCHARM

# HYPER TEXT MARKUP LANGUAGE, TEORÍA

# CASCADING STYLE SHEETS, TEORÍA

# JAVASCRIPT de soy Dalto

**Capítulo I**

1. ***Introducción Que es Js***

Lenguaje de programación interpretado y orientado a objetos, imperativo de tipado débil y lenguaje dinámico, basado en prototipos/classless/instancias, case sensitive

1. ***Editor de texto***

Sublime text3, Visual, Atom

1. ***Para que se usa***

Dinamismo en sitios web, servidor de nodejs, tecnologías frontend (frameworks), también se puede usar para: IA, Mobile apps, Desktop apps, Placas electrónicas.

1. ***Formas de enlazar Js***

En línea: Eventos dentro de etiquetas o del HTML, se usa mejor en externo. También esta <script>, pero se usa un archivo con formato js.

1. ***Variables***

Concepto (IMG), Ejemplo, Tipos de datos (easy), datos especiales (Undefined, Null, Nan), Scope, Hoisting, Tipos de variables, Múltiples variables (separlas con coma).

1. ***Operadores***

De Asignación: asigna un valor desde la izquierda, que se asigna a otro en la derecha

Aritméticos: son valores numéricos en forma de números o variables que devuelven un numérico.

1. ***Concatenación***
2. ***Backticks (Acento grave) y Escape***

No creo llegar a concatenar, pero en todo caso es dejar un espacio DX o usar ´ esto ´.

1. ***Operadores Intermedios***

Lógicos: dan true/false dependiendo, son operadores AND/OR/NOT (&&/ || /!)

De Comparación: se usan para comparar y dar resultados true/false. 5>6 = false

1. ***Camel Case***

conglomeraturaDeCamelloSeEscribeAsi, primera en minúscula, el resto en mayuscula

1. ***Condicionales***

If, elselif, else (condicion) {accion}

1. ***Historia de Cofla I***

**Capítulo II**

# JAVA, TEÓRICO

# FRAMEWORKS TEORÍA

# NODE.JS TEORÍA

# C, C#. TEORÍA

# DATA SCIENCE TEORÍA

# DATA BASE TEORÍA

# GIT

GIT es el sistema de control de versiones más usado del mundo, se va a ver una introducción a la terminal, y profundizar GIT, almacenar el código en la nube, la herramienta para tener un historial completo evitar o reconocer errores a través de una versión guardada y hacer rollback (Volver a una versión anterior).

Reconocer que GIT es un sistema descentralizado y cada developer tiene una copia de su trabajo

*Usos: Historial - Almacenar Código - Trabajar en Equipo - Identificar errores*

Usar GIT: SourceTree - GitKraken - GitHub Desktop - Terminal (BASH)

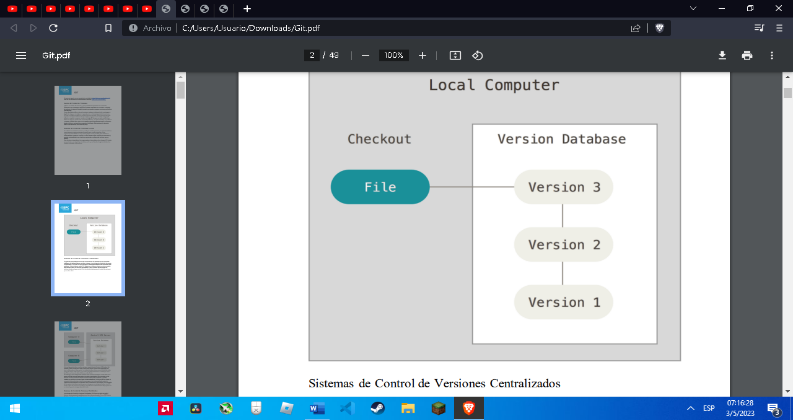
Aprender a usar BASH, usar git antes de llamar un comando

***¿Qué es el Sistema de Control de Versiones?***

Es un sistema que registra cambios en un archivo o conjunto de archivos a lo largo del tiempo, para volver a una versión anterior y evitar o reconocer errores.

**¿Qué es un Sistema de Control de Versiones locales?**

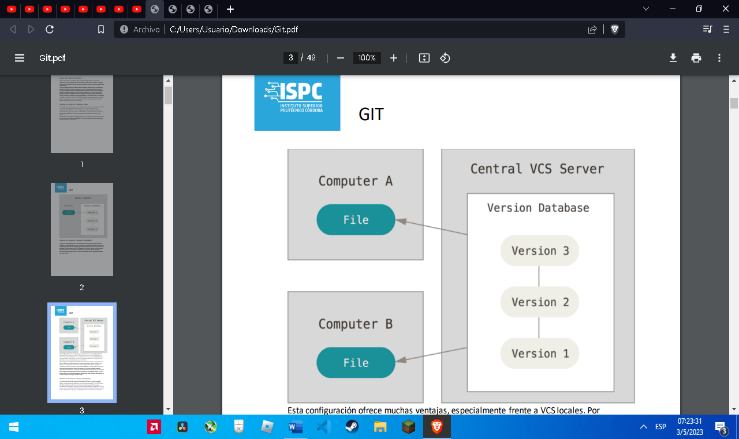
Uno de los métodos de un VCS (Version Control System) Local es copiando los archivos con fecha y hora, hacia otro directorio, es sencillo pero propenso a errores, para afrontar el problema se presentó una base de datos que registraba todos los cambios y cada versión.

Version data base (Archivo versión)

Checkout para comprobar el archivo actual o guardarlo)?

Version data base (Archivo version)

Version data base (Archivo version)

**Sistema de Control de Versiones Centralizados (CVCS)**

Fueron desarrollados para colaborar con programadores en otros sistemas, sistemas como VCS, Subversion y Perforce, tienen un único servidor con todos los archivos(versiones), y varios clientes van para descargar versiones.

Version para computadora A, B y diferentes versiones disponibles

Con cd nombredecarpeta se navega a través de las carpetas

Con ls se ven los distintos directorios

Con pwd se ve el directorio actual

# GITHUB, SOURCETREE Y TEORÍA

# HTTP (HYPERTEXTTRANSFERPROTOCOL), TEORÍA

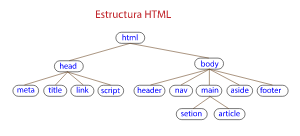
***HTTP es un tipo de protocolo que se utiliza para***

***Rest (Representational Estate Transfer), Teoría***

# RESTFUL, CRUD (CREATE READ UPDATE AND DELETE) FOR DATABASE

# ARQUITECTURA WEB, TEORÍA

Se puede decir que las páginas web son árboles, nodos relacionados entre sí para proporcionar cierta interacción o transmitir mejor la información.

Las imágenes a continuación son una muy buena guíaHay entonces:

**Head:** Va a contener {meta}, {title}, {link}, {script}

{meta}= Es como la referencia que va a tener el buscador para presentar la web

{title}= Va a ser el titulo de la pagina misma en la pestaña

{link}= Va a relacionar el HTML con el css3, para una web con más diseño

{script}= Va a relacionar el HTML con el js, para una web con más interacción

**Body:** Va a contener {header}, {nav}, {main, [section, article]}, {aside}, {footer}

{header}

{nav}= Área de navegación para el usuario en la pagina

{main [section, article]} = Contenido principal de la pagina

{aside}= Contenido al costado en una ventana de la pagina

{footer} = Links de referencia, etc. Pies de pagina

# EXPRESS.JS, TEORÍA

# SQL (MYSQL), TEORÍA

# HTTPS; SSL (SECURE SOCKET LAYER)/TLS (TRANSPORT LAYER SECURITY)

# SISTEMAS DE AUTOMATIZACION

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CURSO – C++\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

# C++

**PROGRAMACION ATS** Objetivos:

-Aprender a desarrollar algoritmos mediante C++ para solventar problemas y crear programas, programación orientada a objetos y sistemas

Temas:

[Básicos]

1. Introducción y Entrada-Salida

#include<iostream> // Comentarios de una sola línea

-using namespace std;

-int main () {

-cin>>*variable input*; cout<<” \n mensaje \n” <<endl o variable;

Lo que va a hacer la función /\*

return 0; Función principal(main) y comentario multilínea

} \*/

Tipos de datos:

**Int** **Float** **Char** **Double Bool**

1. Expresiones y Operadores
2. Condicionales
3. Bucles o Ciclos
4. Arreglos
5. Matrices o Tablas
6. Cadenas
7. Métodos de ordenamiento
8. Búsqueda
9. Estructuras
10. Funciones
11. Punteros

[Intermedios]

1. Pilas
2. Colas
3. Listas
4. Arboles
5. Archivos

[Avanzados]

1. POO: Clases y Objetos
2. POO: Clases derivadas – Herencia y Polimorfismo
3. Genericidad – Plantillas (Templates)
4. Análisis y Eficiencia de Algoritmos
5. Algoritmos Recursivos
6. Biblioteca Estándar de plantillas (STL)

***------------------------------------------CLASE I-----------------------------------------------***

# LIDERAZGO:

***Concepto:***

***- ¿Qué es liderazgo?*** *-*, capacidad de influenciar a las personas usando atributos personales y/o comportamientos para un objetivo en común. No busca un fin propio, busca sacar lo mejor de las personas para inspirarlas y motivarlas, para reservar talentos, energías y habilidad para que aporten en la organización de un proyecto.

***Técnicas:***

El líder tiene que actuar rápido y detenerse a pensar, ser sensato, tomar decisiones legitimas y argumentadas, debería saber aprovechar fortalezas y rodearse de personas que se complementen, sabe que cada persona es única y potencia las fortalezas de cada uno, liderazgo es potenciar la capacidad de las personas.

***Líder:*** Aquellas personas que pueden modificar o influenciar a otras personas para lograr objetivos específicos, resuelve problemas y soluciona conflictos

***- ¿Ser buen líder? -*** *Nacer teniendo habilidades de líder y desarrollando otras aprendiendo y practicando.*

***Ser jefe no quiere decir que es ser Líder, Así mismo un líder no siempre es un jefe, igualmente seguirá siendo líder.***

***----------------------------------------CLASE II------------------------------------------------***

***Objetivos:*** *-Analizar la reacción al cambio en un líder y ciertas personas.*

***Cambios:***

*No hay miedo a los cambios, sino al camino al que hay que recorrer “Marilyn Ferguson”*

***Reacción al cambio:***

Al cambiar ideas “*innovadoras*”, cambia la visión y se espera que los cambios sean aceptados y colaboradores se adapten a los cambios, sin peros. **ETAPAS DE REACCIÓN AL CAMBIO:**

*Final* – Negociación- Enojo – Miedo – Shock – Frustración - Confusión – Estrés (Decreciente)

Creatividad – Escepticismo – Aceptación – Impaciencia – Esperanza – Energía – Emoción - Inicio (Ascendente)

Inicio Fin

***Palabras Prohibidas:***

-No lo sé -No -Usted está equivocado -Por Que Necesita Saberlo

-No es mi Trabajo -Nunca lo hemos hecho de esa forma -Lo tiene que hacer

*Personas Negadas a un cambio, prefieren evitarlo o directamente conformarse con la misma visión e ideas. Siempre se busca una reacción positiva*

***Métodos para manejar los cambios:***

\*Pensar en ventajas y desventajas a cambio, \*analizar los retos, arriesgarse, \*no tener miedo, \*ver lo mejor siempre

***----------------------------------------CLASE III-----------------------------------------------***

***Objetivos:*** Analizar la importancia de la resiliencia en el liderazgo, por audiovisual

***Resiliencia:***

Una capacidad para arriesgarse y hacer que las cosas sucedan, buscar oportunidades de crecimiento/desarrollo, promover ideas que beneficien al resto, mantener una actitud de empoderamiento, ser proactivo y manejar el tiempo.

Las características de una persona resiliente:

1. Actitud de resiliencia, controlar lo inesperado.
2. Ven hacia el futuro, personas que se preocupan por lo que suceda o no suceda.
3. Son flexibles, esta habilidad de adaptarse al cambio, ven diferentes puntos de vista.
4. Imponen orden al caos, ven las dificultades/problemas desde diferentes puntos de vista para buscar una solución. Mantienen la calma y piensan con mente fría.
5. Oportunidad al cambio, ven la oportunidad al cambio, no se preocupan de las dificultades.

***----------------------------------------CLASE IV-----------------------------------------------***

***------------------------------------------CLASE I-----------------------------------------------***

# TELEGRAFÍA

***------------------------------------------CLASE I-----------------------------------------------***

# ARDUINO

------------------------------------The Clean Code Book---------------------------------------

# Libro: The Clean Code

Autor: Robert C. Martin

The Clean Code promete las mejores prácticas agiles para limpiar código y entregar valor al mismo, los ejemplos se encuentran mayormente para java. Escribir código limpio aumente muchas proporciones de eficiencia y mantenibilidad. Habla de:

Escribir bien variables

Mental mapping

funciones anidadas (Posible mala práctica)

el código tiene que explicarse solo

formatear código?

nombres para estructuras de datos,

cómo manejar errores

el try catch

prácticas de test unitarios

mover test?

Abstracción

mejora continua

factorización

code smell

código muerto

refactorización

------------------------------------The Clean Coder Book--------------------------------------

# Libro: The Clean Coder

Autor: Robert C. Martin

Libro orientado a la conducta, es la continuación de The Clean Code, ¿ser respetado?, ¿ser profesional?, ¿Cumple con eso?

Sin código, únicamente historias, responder, reconocer.

Malas prácticas, ser un juguete.

Startups son empresas pequeñas con pocos recursos, hay una carencia de profesionalismo en informática, como decir historias

Recuperarse en jornadas extensas, (si trabajo 8 horas descanso 8 horas).

Reconocer tiempos imposibles y lideres buenos o malos, o directamente jefes imposibles.

Conocer la zona en programación

**ESTIMACIONES**

Estimar el tiempo de entrega de un proyecto bien hecho

# CODE COMPLETE

# Hackathon 2023 [Evento]

Fue realizado el 20/05/2023 Por el CSIRT de Córdoba con temática de inteligencia artificial y ciberseguridad, centrada en modelos NLP, se pudo comprender el propósito y como funciona en parte una inteligencia o cuales son patrones relacionados para la tecnología de la información y comunicación.

Mi equipo (Equipo 30), se encargo de realizar un generador de contraseñas por medio de ChatGPT y JavaScript para hacer contraseñas con frases mnemotécnicas y fáciles de recordar.

Puede que no hayamos ganado, pero si vimos cuales eran las cosas en las que debemos mejorar, aprender y/o reforzar, además de lo brillante que pueden ser las personas especializadas en un área, por ejemplo, en mi grupo un programador era el que sobresalía por sus bastos conocimientos. Los universitarios eran fuente de apoyo y aporte de ideas, mis compañeros eran la base y formación de archivos y comprensión de actividad, yo fui un simple observador que intento forzarse a aprender algo que ya debería saber.

Pero por sobre todo queda la experiencia, queda la magia que pueden hacer las personas sobre el resto por una competencia o por pasión, y eso es lo que me gusta de la programación, esa característica de desafiarse y querer aprender muchísimo más, aunque no me alcanza el tiempo en la semana, si puedo realizar todo lo que tengo que realizar y terminar lo que tengo que terminar, puedo aprender más de lo que ya se.

Tengo un sueño, un sueño de ser el mejor.

# TODAS LAS BUENAS PRACTICAS

**DRY** (Don’t Repeat Yourself) = No repetir el código, que sea entendible, por partes y modificable, que no se repita hace un código más legible, una pieza del código debe ser única

**WET** (Repetir el código y sus consecuencias) = Hacer un código extenso, desperdicia el tiempo de todos para el código.

**KISS** (Keep It Simple, Stupid) = Un Sistema funciona mucho mejor si es simple, se usan las demás buenas practicas para llegar a hacer el **KISS**

**YAGNI** (You aren’t going to need it) = Evitar funcionalidades extra, código no necesario o código inflado, da más mantenibilidad al programa, si no te lo piden no hagas cosas extra.

**SOLID** (Principios) = Responsabilidad única, Código abierto cerrado, Sustitución de Liskov, Segregación de Interfaces, Dependencia

**ACOPLAMIENTO** = Visualizar cuales son los componentes que requieren de otros componentes, puede haber acoplamientos aferentes y eferentes, por tipos de datos, de mensajes (llama componentes), son los útiles.

Eferentes son los acoplamientos que tiene un componente que requiere de otros.

Aferentes son los acoplamientos que requieren de un componente para su función

**PRINCIPIO DE MENOR CONOCIMIENTO** = Algo así como usar funcionalidades conocidas en vez de desconocidas

INYECCION DE DEPENDENCIAS = Es la inversión de control, y lo que un objeto necesita (depende), usa métodos, interfaces o un control externo.

**REFACTORING** =

# SOLID (Single-responsibility, Open-closed, Liskov substitution, Interface segregation y Dependency Inversion principles)

**SOLID** son los principios de responsabilidad que existen dentro de la programación, para evitar malos diseños, refactorización y tener un código más eficiente y fácil de mantener  
**(BUENAS PRACTICAS)**

PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS:

Que es una clase: Una clase es donde se guardan atributos y métodos, se pueden sacar varios objetos con mismas características, ej. Clase bloques de MC: en esa clase se van a crear objetos con atributos parecidos, y los métodos son las acciones que hacen los objetos, la TNT tendría el método de explotar.

Encapsulamiento: (Buena Practica), en el encapsulamiento se modifican los atributos de ciertos objetos ingresando únicamente en su código, solo la clase principal puede modificar y/o llamar a los atributos. Se usan Getters y Setters -> JavaScript y C++ define atributos públicos o privados. Getter: Tener objetos Setter: introducir objetos.

Abstracción: Es abstraer los atributos importantes que tiene un objeto, y definir que es para el programa un objeto con atributos similares.

Herencia: Capacidad de heredar características de otras clases, reutilizar código, por ejemplo, hacer 2 personajes con el mismo código y añadirle atributos.

Polimorfismo: Capacidad de tener funciones con el mismo nombre y firma, pero con diferentes comportamientos, por ejemplo, cambiar el tipo de características que tiene una función.

***Con todo esto podemos decir que SOLID***

Tiene los objetivos de: Hacer el código más mantenible (fácil de cambiar y encontrar y resolver errores), facilitar la incorporación de nuevas funcionalidades, y tener un código más legible.

***Responsabilidad Única [Single Responsability]***

Indica que cada clase debería tener una sola parte del sistema, que haga una sola cosa bien hecha.

***Abierto – Cerrado [Open Closed]***

Indica que una entidad debe ser abierta para su extensión, pero cerrada para su modificación, escribir código que no se tenga que cambiar al agregar funcionalidades

***Sustitución de Liskov [Liskov Subtitution]***

Una clase hija debería de utilizarse también como clase padre, sin afectar el programa

***Segregación de la Interfaz [Interface Segregation]***

Tener una interfaz especializada sin muchos métodos que no les de uso, ser especifico y privar clases

***Inversión de la Dependencia [Dependency Inversion]***

Los módulos de alto nivel no deberían depender de los de bajo nivel, deberían depender de interfaces (Abstracciones), no depende de nada, solo una abstracción. Es más fácil trasladar código con esto.

# REFACTORING

1. Evolución del Desarrollo de Software
   1. Evolución de la Ingeniería
   2. Evolución de las metodologías de Desarrollo de Software
2. Contexto del Refactoring
3. Bucle del Refactoring
4. Precondición del Refactoring
   1. Código maloliente vs código limpio
   2. Pruebas Unitarias Automáticas
5. Invariante del Refactoring
6. Condición del Refactoring
7. Cuerpo del Refactoring
   1. Micro pasos del Refactoring
8. Casos Prácticos

# MongoDB

# Lenguaje no verbal