Renderizado Web

Tipos de renderizado:

- **Server Side Rendering (SSR):** Es una estrategia donde el contendido HTML de la página web es renderizado en el servidor y posteriormente enviado al cliente (Navegador) en tiempo de ejecución. Esto a su vez permite generar contenido dinámico desde el servidor, un ejemplo puede ser nuevas reseñas de un producto o precios en un e-commerce.

El cliente pide una página (enviando una solicitud al servidor)

El servidor recibe la solicitud y determina qué página debe procesar

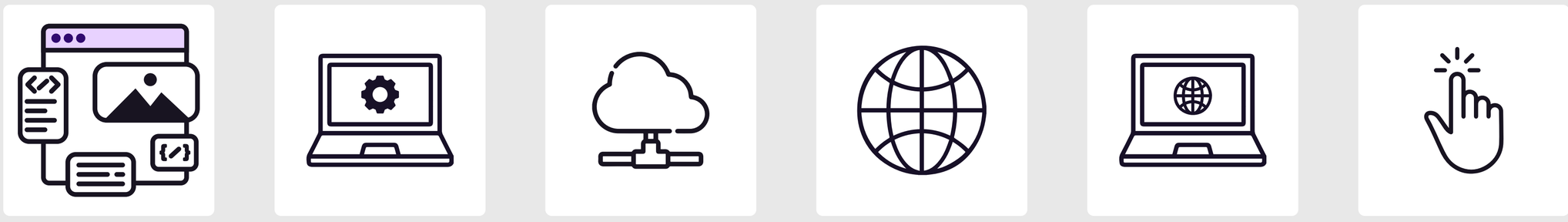
El navegador recibe el HTML pero no es interactivo

El navegador descarga el JavaScript

El navegador executa el JavaScript

El sitio web se vuelve interactivo (si se hace hidratación o similares)

- **Static Site Generation (SSG):** La renderización también ocurre en el servidor con la particularidad de que es en tiempo de construcción de la aplicación. De esta manera obtenemos archivo .html para determinadas URLs antes de que sean solicitadas, por lo cual son estáticos (solo se vuelven a generar al construir la aplicación).



Preparación de contenido (datos y plantillas) en el momento de la construcción

Generación de las páginas estáticas (HTML) usando los datos y plantillas

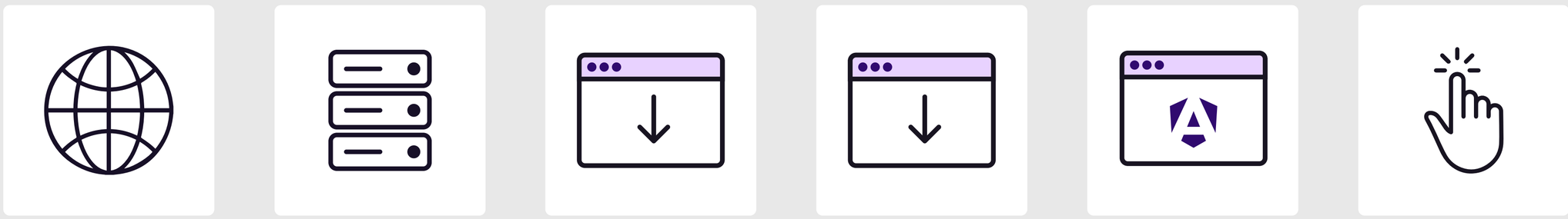
Almacenamiento de las páginas generadas en el servidor o en un CDN

El cliente solicita una página

El servidor o CDN envía la página estática al cliente

El navegador muestra la página con el HTML recibido

- **Client Site Rendering (CSR):** En esta estrategia todo el contenido HTML es renderizado en el cliente dinámicamente utilizando JavaScript. Frameworks y librerías como React, Angular y Vue.js son comúnmente utilizados para crear SPAs



El cliente pide una página (enviando una solicitud al servidor)

El servidor envía el HTML básico (junto con los archivos JavaScript)

El navegador descarga el HTML

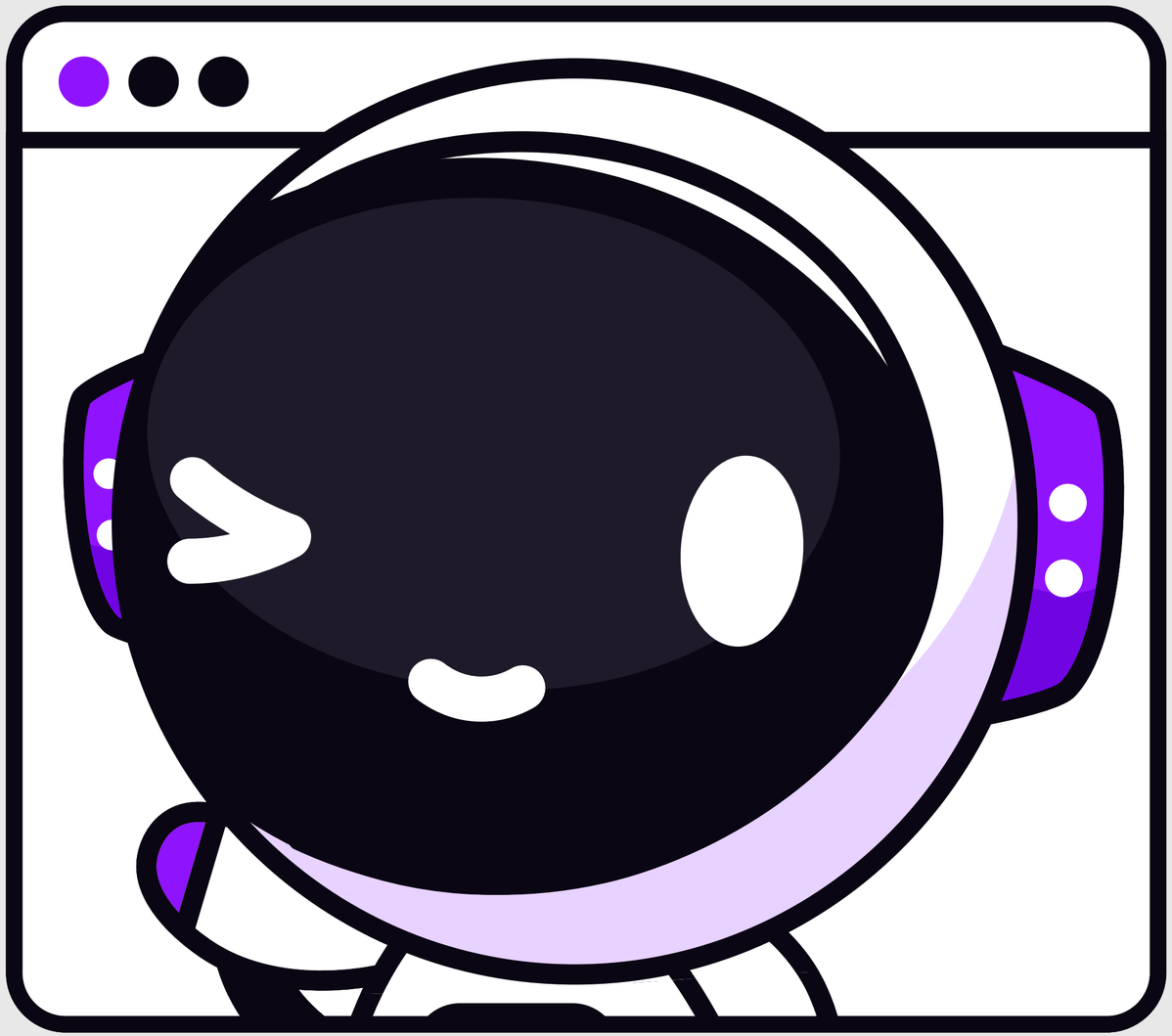
El navegador descarga el CSS y JavaScript

El navegador executa el framework o libreria

El navegador muestra la página con el contenido generado

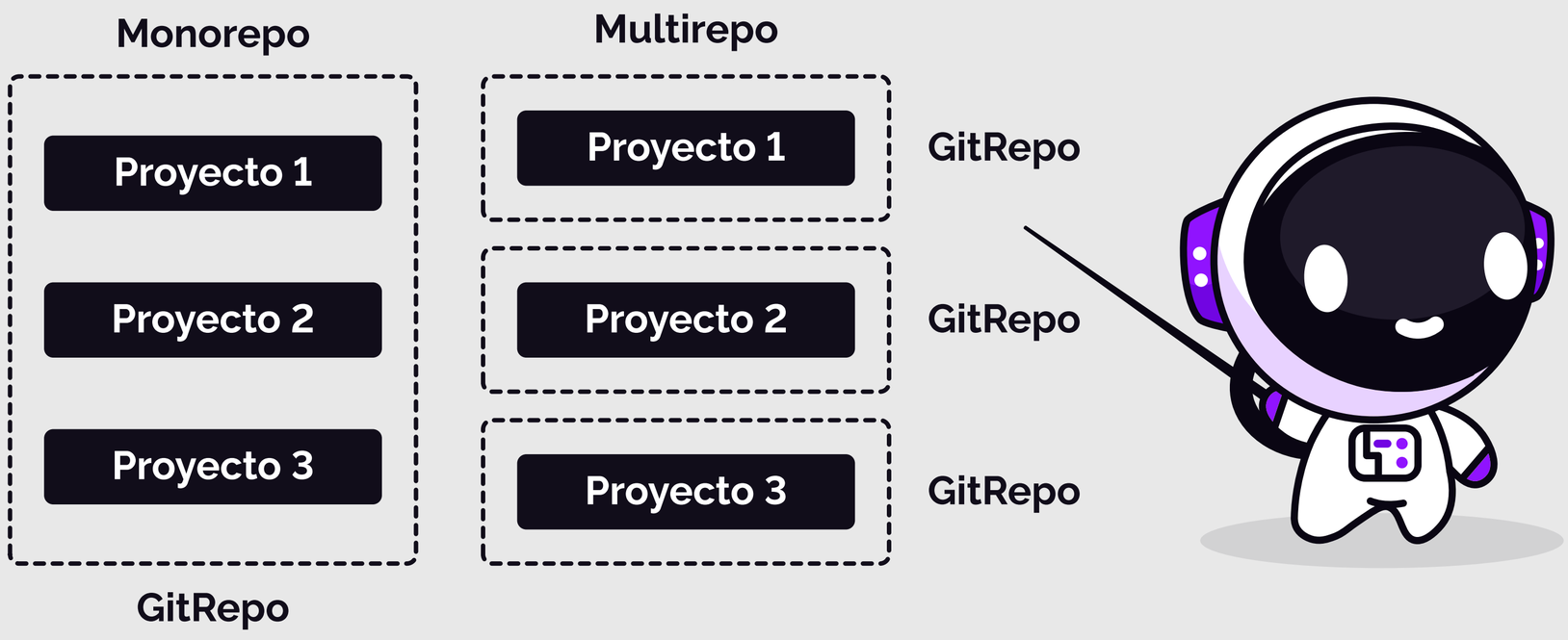
Links de referencia 1 Cursos relacionados Más recursos

Renderizado Web



Descarga el PDF adjunto

Monorepo VS Multirepos



Monorepo: es un enfoque en el desarrollo de software en el que todo el código fuente de varios proyectos se almacena en un solo repositorio de código con relaciones bien definidas. Sin embargo, esto no implica que un monorepo sea un monolito.

Ventajas:
- Compartir código entre proyectos de forma fácil.
- Facilita la colaboración y la revisión de código entre equipos y proyectos.
- Mejora la consistencia y coherencia del código al tener una única base de código.
- Facilita las actualizaciones e implementación de cambios en todos los proyectos.

Desventajas:
- Puede ser voluminoso, lo que dificulta la gestión y puede requerir herramientas especializadas.
- Puede haber problemas de rendimiento al escalar a un gran número de proyectos y colaboradores.
- A medida que crece la cantidad de proyectos, puede volverse más complicado de mantener y escalar el monorepo.

Herramientas:

Nx TURBO Bazel

Multirepos: en este enfoque cada proyecto tiene su propio repositorio de código independiente. Además, suele ser común que cada repositorio construye un solo artefacto.

Ventajas:
- Cada proyecto tiene su propio control de versiones, lo que puede simplificar la gestión y el control de cambios.
- Permite una mayor flexibilidad en la configuración de permisos de acceso a cada repositorio.
- Pueden ser más fáciles de administrar individualmente.

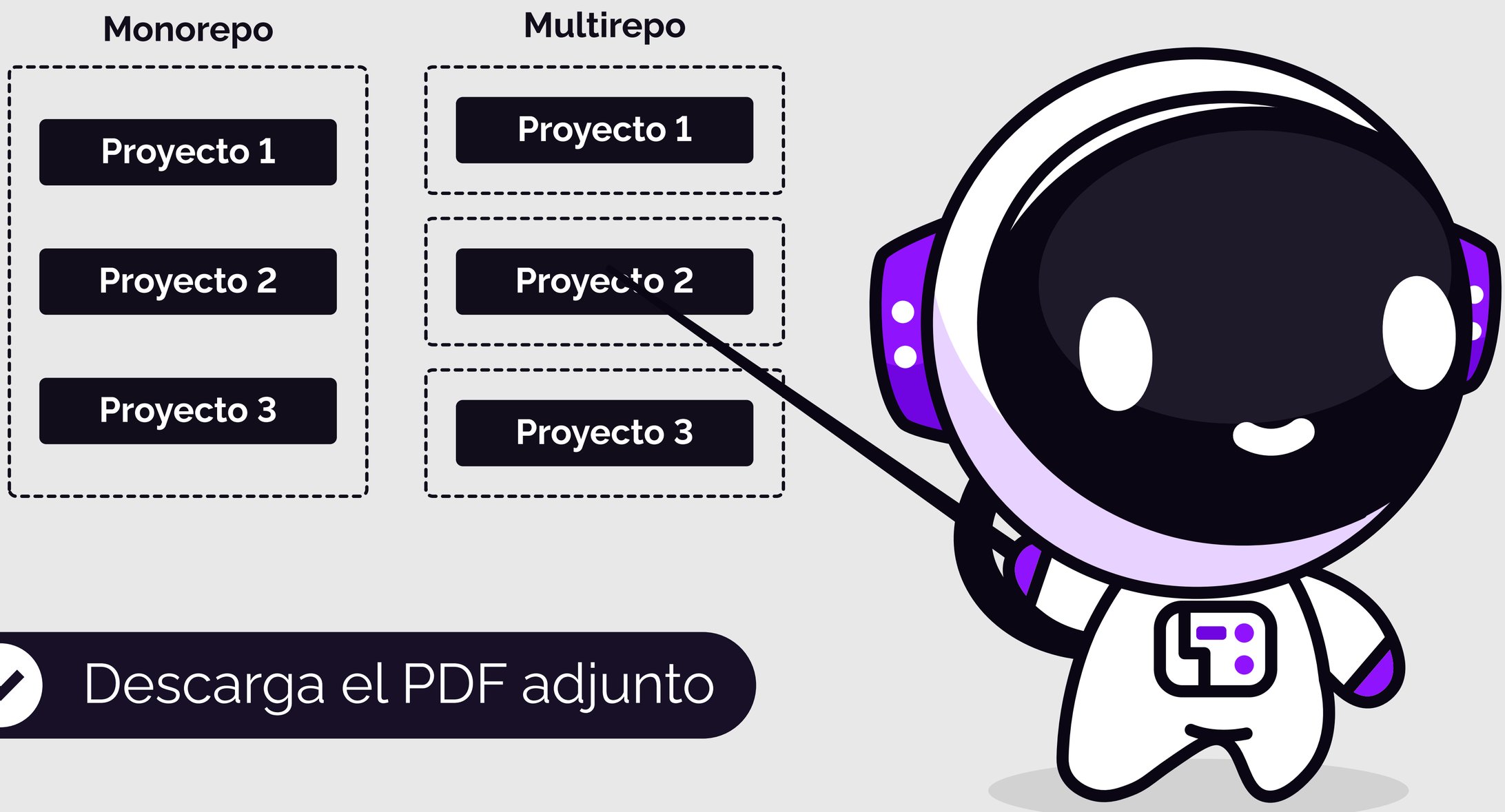
Desventajas:
- Duplicación de código entre repositorios, lo que puede llevar a inconsistencias.
- Puede ser más difícil compartir código y recursos entre proyectos.
- Coordinar cambios que afectan a varios repositorios puede ser más complicado.

Herramientas:

GitHub GitLab Bitbucket

Links de referencia 1 Cursos relacionados Más recursos

Monorepo vs Multirepos



Descarga el PDF adjunto

Lenguajes de alto nivel y de bajo nivel

Lenguajes de bajo nivel: estos lenguajes están más cerca del lenguaje máquina y son específicos de la arquitectura del hardware. Permiten un control detallado sobre el hardware y los recursos del sistema.

Ejemplos: Lenguaje de máquina, Ensamblador

Ventajas:
- Mayor control sobre el hardware, lo que les hace más eficientes en términos de uso de recursos.
- Permiten una optimización a nivel de código para mejorar el rendimiento de las aplicaciones.
- Son útiles para programación de microcontroladores y sistemas embebidos donde los recursos son limitados.

Desventajas:
- Son menos portables, ya que están directamente ligados al hardware específico de la computadora.
- Requieren un mayor esfuerzo y tiempo de desarrollo debido a su complejidad.
- Requieren un conocimiento profundo de la arquitectura del hardware específico.

Lenguajes de alto nivel: su sintaxis suele ser más cercana al lenguaje humano, además de ser independientes de la arquitectura del hardware, esto permite que se puedan abstraerse de los detalles específicos del hardware. Estos lenguajes suelen estar orientados a objetos, a eventos o a funciones, pudiendo estos combinarse.

Ejemplos: Java, PHP, Python, JavaScript, C#, entre otros.

Ventajas:
- Facilitan la programación al proporcionar abstracciones de mayor nivel y una sintaxis más clara.
- Promueven la productividad al permitir desarrollar aplicaciones más rápidamente.
- Mayor portabilidad, ya que el mismo código puede ejecutarse en diferentes plataformas.

Desventajas:
- Menor control sobre los recursos del sistema, lo que puede afectar la eficiencia en algunos casos.
- Algunos requieren que la máquina cliente posea una determinada plataforma.
- Algunos lenguajes de alto nivel pueden requerir la interpretación o traducción a un lenguaje de bajo nivel, lo que puede afectar el rendimiento.
- Coordinar cambios que afectan a varios repositorios puede ser más complicado.

También hay lenguajes considerados de nivel medio (tiene un mayor nivel de abstracción): Como lo pueden ser C/C++, Rust, Go, entre otros.

Links de referencia 1 2 Cursos relacionados Más recursos

Lenguajes de alto nivel y de bajo nivel

Descarga el PDF adjunto

Programación Orientada a Objetos (POO)

La programación orientada a objetos es un paradigma de programación que basa en los “objetos” para representar elementos en el programa a desarrollar. Dichos objetos son entidades que pueden almacenar información y realizar acciones.

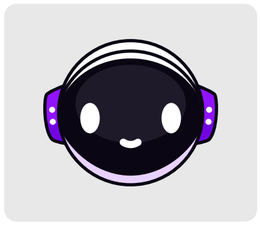
Conceptos fundamentales:

CLASE

Ej.: mascota corporativa.

Una clase es una plantilla que define las propiedades y métodos de los objetos que se crean a partir de ella.

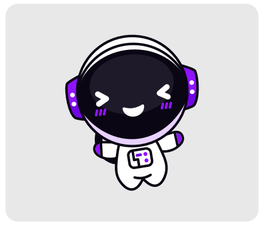
OBJETO



Ej.: mascota de DevTalles.

Es la instancia de la clase, este tendrá un conjunto de propiedades y comportamiento (métodos).

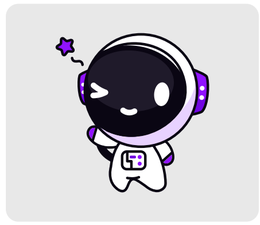
PROPIEDADES



Ej.: Devi, amigable, divertido.

Son contenedores de datos asociados a un objeto que pueden ser visibles externamente y modificados por métodos.

MÉTODOS



Ej.: comunicar, entretener.

Son funciones definidas dentro de una clase.

CONSTRUCTOR

Ej.: se encargará de inicializar los atributos de una nueva mascota.

Es un método especial que se invoca al crear un objeto y permite inicializar valores en la instancia de la clase, pudiendo incluir parámetros si es necesario.

Conceptos fundamentales:

- **Abstracción:** Permite definir las características de cada objeto por medio de las clases, para que de esta manera los objetos puedan interactuar sin que sea necesario conocer los detalles de su funcionamiento interno.

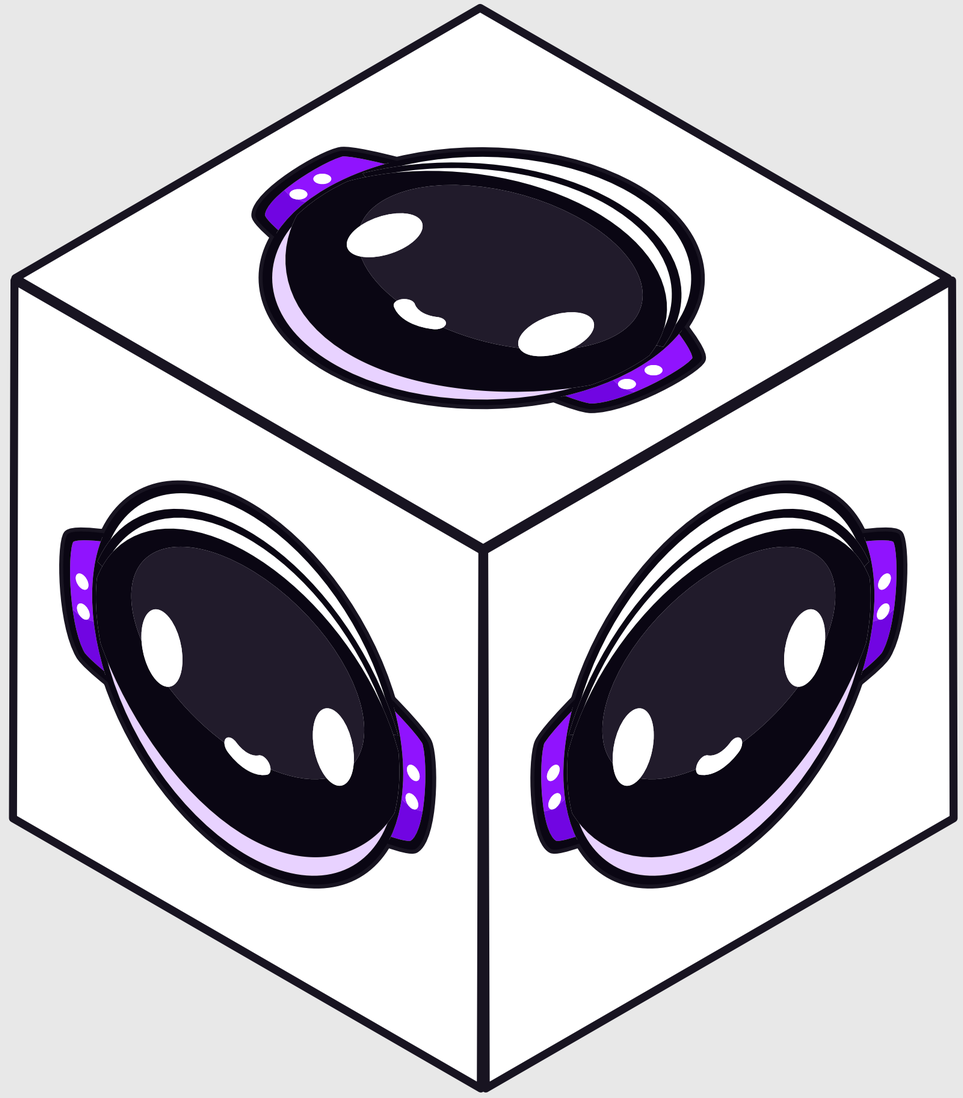
- **Encapsulamiento:** Nos permite proteger y ocultar información así como el comportamiento de los objetos. Esto permite evitar que se altere el

- **Herencia:** Permite que la funcionalidad de otra clase existente sea reutilizada, de tal manera que una clase hija extiende la funcionalidad de una clase padre.

- **Polimorfismo:** Es la capacidad de que un objeto de tener un determinado comportamiento dependiendo de su contexto. Por ejemplo, una clase hija puede implementar o sobrescribir un método definido en una clase padre.

Links de referencia 1 Cursos relacionados 1 2 3 Más recursos

Programación Orientada a Objetos (POO)



Descarga el PDF adjunto