Entorno de desarrollo de astro:

.Primero entramos en astro.build, donde vamos a encontrar el comando para crear proyectos. (necesitamos tener nodejs)
.Abrimos la terminal de windows, escribimos “cd Desktop”. (si no tenemos instalado astro, nos va a pedir hacerlo)
.Nos va a pedir que le pongamos nombre al proyecto.
.Luego nos va a dejar elegir usar plantillas ya creadas o un proyecto vacio.
.Nos va a pedir instalar las dependencias y ponemos que si.
.Luego nos pregunta si queremos crear un repositorio de git junto con el proyecto.
.colocamos en la terminal “cd astro-curso” o el nombre que le hayamos colocado al proyecto y dentro de este ponemos “npm run dev” y nos va a salir el puerto para ejecutar el programa.
.Ahora lo abrimos en el editor de codigo.
.Nos va a ayudar la extencion astro.

Estructura del proyecto:

. La primer carpeta que vamos a encontrar es vscode, donde se almacenan las configuraciones del editor.
. node_modules es donde estan todas las dependencias que necesita el proyecto para funcionar.
. En la carpeta public van los archivos que el navegardor va a poder acceder (imagenes, pdf, etc).
. En src van los archivos con codigo que vayamos a crear.
. Los archivos finales son de configuracion o funcionamiento del proyecto.

Pagina en ASTRO:

Para crear mas paginas, es dentro de la carpeta “page” y el nombre del archivo, seguido de un punto y “astro”. (servicios.astro)

Extension para ASTRO:

En los archivos astro, podemos escribir cualquier archivo HTML. Al igual que etiquetas <style> para agregar css. Y las etiquetas <script> para escribir JS.

```
<h1 class="title">HOLA MUNDO</h1>

<style>
    .title {
        color: skyblue;
    }
</style>

<script>
    console.log("HOLA  MUNDO");
</script>
```

La idea de astro es escribir codigo para un sitio web en varios archivos y que al final todos se junten en uno solo.
Si colocamos codigo dentro de 2 pares de guiones, este se va a mostrar en la consola del editor, nunca en el frontEnd por que significa que esta en desarrollo. No pasa a fase final.

Sintaxis JSX en Astro.build:

Cuando colocamos codigo entre llaves “{}”, interpreta sintaxis de JS.
Se pueden usar variables o constantes que interactuen con la interfaz.

```
<h1 class="danger">{3+3}</h1>
```

```
---
const color = "danger";
const hobbies = ["programar", "leer", "escribir", "ver peliculas"];
---
<h1 class={color}>{hobbies[0]}</h1>
```

```
<style>
    .title {
        color: skyblue;
    }
    .danger {
        color: red;
    }
</style>
```

```
<script>

</script>
```

Tambien se puede utilizar para traer datos de un servidor externo.

```
const color = "danger";
const hobbies = ["programar", "leer", "escribir", "ver peliculas"];
---
<ul>
    {hobbies.map( hobby => <li>{hobby}</li>)}
</ul>
```

```
<style>
    .title {
        color: skyblue;
    }
    .danger {
        color: red;
    }
</style>
```

De esta manera recorremos un array creando otros elementos con JS.
Operadores ternarios = (?) = if / (!)= else

```
const color = "danger";
const hobbies = ["programar", "leer", "escribir", "ver peliculas"];
const isActive = false;
---

<ul>
  {
    hobbies.map((hobby) => (
      <li>
        <h3>{hobby}</h3>
        <button>Like</button>
      </li>
    ))
  }
</ul>
<h3 class="danger">
  {isActive ? "El usuario esta activo" : "El usuario esta desactivado"}
</h3>

<style>
  .title {
    color: skyblue;
  }
  .danger {
    color: red;
  }
</style>

<script></script>
```

FETCH (JS API): Obtener datos de servidor. Usamos JSON PLACE HOLDER para practicar a traer datos desde un servidor y mostrarlos en pantalla.

```
---
const response = await fetch ("https://jsonplaceholder.typicode.com/posts")
console.log(response)
const posts = await response.json()
---
<h1>Mis publicaciones</h1>

{
    posts.map (post => (
        <div>
            <h5>{post.userId}</h5>
            <h3 class="title">{post.title}</h3>
            <p class="body">{post.body}</p>
        </div>
    ))
}
<style>
```

```
    .body {
        color: blue;
    }
    .title {
        color: purple;
    }
</style>
```

ASTRO COMPONENTS: En astro tambien se usan componentes. Creo un componete card con su contenido.

```
<div>
    <h1>Mi tarjeta</h1>
    <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Perspiciatis, maxime!</p>
</div>
```

Y lo importamos con la siguiente linea de codigo a la pagina o archivo que queramos.

```
---
import Card from "../components/cards.astro"
---
```

Los props son un tipo de datos que para acceder a ellos tenemos que usar una funcion de astro dentro del componente donde vayamos a utilizar estos datos.

```
---
const {title} = Astro.props
---
<div>
    <h1>{title}</h1>
    <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Perspiciatis, maxime!</p>
</div>
```

y los datos que tengan los props, los tenemos que poner en el archivo donde vayamos a usar los componentes.

```
<Card title="Programacion"/>
<Card title="Dibujo"/>
<Card title="Fotografia"/>
<Card title="Interfaz"/>
```

LAYAUTS: Sirve para reutilar codigo y no tener que crear cada vez que hacemos una pagina nueva, toda la estructura base. Creamos en la carpeta “componentes” un archivo llamado “Layout.astro” dentro, escribimos el codigo base de un archivo HTML y dentro de “body” ponemos la estiqueta “slot”, esta etiqueta sirve para decir que ahi va a aparecer el codigo que queremos ir cambiando.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Curso Astro</title>
</head>
<body>
```

```
    <slot/>

</body>
</html>
```

Luego en “index.astro” ponemos lo que queremos que aparezca en el archivo base y lo importamos al archivo base. Y el contenido dentro de unas etquetas con el nombre del archivo base. En este caso “Layout”.

```
---
import Layout from "../components/Layout.astro";
---
<Layout>
  <h1>Pagina inicial</h1>
  <p>Bienvenido a mi blog</p>
</Layout>
```

Esto lo podemos hacer en las paginas que querramos, de manera que nos ahorramos codigo. Y se pueden usar props de igual manera. En este caso lo usamos para ir cambiando el titulo.

```
---
const {PageTitle} = Astro.props
---

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> Astro/{PageTitle}</title>
</head>
<body>
    <h1>Esto esta en Layout</h1>
    <slot/>

    <style>
        body {
            background: #202020;
            color: white;
        }
    </style>

</body>
</html>
```

Y ahora lo que hacemos es poner el props en el archivo del que saquemos la informacion.

```
---
```

```
import Layout from "../components/Layout.astro";
---
<Layout PageTitle = "Pagina Inicio">
  <h1>Pagina inicial</h1>
  <p>Bienvenido a mi blog</p>
</Layout>
```

MARKDOW: Es un formato similar a HTML, solo que mucho mas sencillo de escribir. Para que nos lea este formato, tenemos que colocarle como terminacion ".md" al archivo. Y se escribe asi.

```
# Mi primer archivo con markdown

Este es mi primer md
```

Si queremos importar estilos es tan facil como en el archivo markdow entre los guiones escribimos layout seguido de la ruta del archivo que contenga los estilos.

```
---
layout: ../components/Layout.astro
---
```

La funcion "glob" de astro sirve para decirle que archivos queremos cargar.
Por lo tanto podemos hacer esto con glob, cargar todos los archivos con terminacion ".md" y de esta manera al ir creando mas archivo, se van a ir añadiendo a la pagina "blog" solos.

```
---
import Layout from "../../components/Layout.astro";

const posts = await Astro.glob("./post/*.md")

console.log()
---
<Layout PageTitle="Blog de desarrollo"
>
<h1>Blog</h1>

{
    posts.map( post => {
        return <a href={post.url} class="link">{post.frontmatter.title}</a>
        <br>
    })
}
<style>
    .link {
        background-color: burlywood;
        color: chocolate;
    }
</style>


</Layout>
```

Nos va a crear un objeto por cada archivo ".md" con todos los datos de este, su url, titulo, etc. y accedemos con la funcion "map" y usamos estos datos.

IMAGENES CON ASTRO: Las imagenes se insertan exactamente igualq ue con html, con la etiqueta <img> y en el src colocamos la url. O subimos la imagen a la carpeta puclic y en src colocamos la ubicacion. Tambien se puede crear una constante e importar la imagen y luego en la etiqueta img colocamos la variable con la imagen.

```
<img src="https://tutorias.co/wp-content/uploads/2020/03/expresionRegularCodigoEmpleadoC-1-672x372.jpg" alt="">
```

INTEGRATIONS: Astro permite usar componentes de otros frameworks. Y se pueden llamar desde astro y añadirlos a nuestro sitio. Usando los comandos predeterminados de la documentacion de Astro, se puede añadir react a nuestro proyecto con astro. En este caso para añadir react es “npx astro add react”. Luego nos va a ir diciendo que se van a hacer cambios en ciertos archivos y tenemos que poner que “si”.
Creamos un componente en react de un boton, y le añadimos la funcionalidad de que cambie lo que dice segun si esta en “false” o “true”.

```
import { useState } from "react"

function Buton() {
    const [subscribe, setSubscribe] = useState(false)
  return (
    <button onClick={() => setSubscribe(!subscribe)}>
        {
            subscribe ? "Ya estas suscripto" : "Suscribete"
        }
    </button>
  )

}
 export default Buton
```

Luego lo añadimos en el archivo donde queremos que aparezca.

```
---
import Layout from "../components/Layout.astro";
import Buton from "../components/buttonSus.jsx"
---
<Layout PageTitle = "Pagina Inicio">
  <h1>Pagina inicial</h1>
  <p>Bienvenido a mi blog</p>
<Buton client:load/>

</Layout>
```

El “client:load” sirve por que Astro requiere que le digamos cuando cargar el archivo js, de esta manera lo carga una vez cargada la pagina. Y hay otros modos de carga para “client” (investigar).
De esta manera se cargan interaciones de REACT.
Se pueden importar bibliotecas con pocos comandos, en este caso vamos a usar “Toastify” que sirve para agregar notificaciones o alertas ya creadas. De esta manera las importamos en el archivo que las vayamos a usar.

```
import { useState } from "react";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
```

Las agregamos al boton, y usamos las funciones que nos indica la documentacion, y luego añadimos el boton a donde lo vayamos a usar, antes importado.

```
function Button() {
  const [subscribe, setSubscribe] = useState(false);
  return (
    <>
      <button
        onClick={() => {
          toast.success("Felicidades, ya estas suscrito");
          setSubscribe(!subscribe);
        }}
      >
        {subscribe ? "Ya estas suscripto" : "Suscribete"}
      </button>
      <ToastContainer/>
    </>
  );
}
export default Button;
```