TypeScript + Web Development Study Guide

■ VARIABLES & BASIC TYPES

- string: let name: string = 'Marjorie'
- number: let age: number = 19
- boolean: let isStudent: boolean = true

■ ARRAYS

- Syntax: let fruits: string[] = ['apple', 'banana']
- Example: let scores: number[] = [90, 85, 78]

■ OBJECTS & TYPE ALIASES

- Define object shape:
  type User = { name: string; age: number }
  let user: User = { name: 'Bloom', age: 19 }

■ INTERFACES

- Interfaces are extendable object contracts:
  interface Product { id: number; name: string; price: number }
  let item: Product = { id: 1, name: 'Book', price: 9.99 }

■ FUNCTIONS

- With parameter + return type:
  function add(a: number, b: number): number { return a + b }
- Arrow functions:
  const greet = (name: string): string => `Hello, ${name}`

■ OPTIONAL & DEFAULT PARAMETERS

- Add ? to mark optional:
  function logMessage(msg: string, prefix?: string) { ... }

■ UNION TYPES

- Allow more than one type:
  let id: string | number

■ GENERICS

- Reusable flexible types:
  function getFirst<T>(arr: T[]): T { return arr[0] }

■ TYPE ASSERTIONS

- Tell TS a variable type:
  let val: unknown = 'hello'
  let len: number = (val as string).length

■ ENUMS

- Define fixed sets of values:
  enum Direction { Up, Down, Left, Right }

```
  let move: Direction = Direction.Up
```

■ REACT + NEXT.JS WEB DEV

■ COMPONENTS

- A component is just a function returning JSX:

```
  function Navbar() { return <nav>Bloom</nav> }
```

■ NEXT.JS LINK

- Use Link instead of <a> for client-side navigation:

```
  import Link from 'next/link'
  <Link href='/about'>About</Link>
```

■ PROPS

- Props = inputs to a component, typed with TS:

```
  type ButtonProps = { label: string; onClick: () => void }
  function Button({ label, onClick }: ButtonProps) {
    return <button onClick={onClick}>{label}</button>
  }
```

■ CHILDREN

- Special prop for nested JSX:

```
  type CardProps = { children: React.ReactNode }
  function Card({ children }: CardProps) {
    return <div className='card'>{children}</div>
  }
  <Card><h2>Title</h2><p>Body</p></Card>
```

■ STATE

- useState remembers values inside components:

```
  import { useState } from 'react'
  const [count, setCount] = useState<number>(0)
  <button onClick={() => setCount(count + 1)}>Count: {count}</button>
```

■ EVENTS

- Typed event handlers for safety and autocomplete:

```
  function SearchBar() {
    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
      console.log(e.target.value)
    }
    return <input onChange={handleChange} />
  }
```

■ RENDERING LISTS WITH map()

- map arrays to JSX, always give a key:

```
const items = ['apple','banana']
<ul>
  {items.map((fruit, i) => <li key={i}>{fruit}</li>)}
</ul>
```

■ ASYNC DATA FETCH

- Fetch in async components (Next.js 13+):

```
async function getArticles() {
  const res = await fetch('https://example.com/rss')
  return res.json()
}
export default async function NewsPage() {
  const articles = await getArticles()
  return <pre>{JSON.stringify(articles, null, 2)}</pre>
}
```

--

■ KEY TAKEAWAYS

- TypeScript ensures your props, state, and events are correct.
- Next.js Link enables fast navigation without full page reload.
- useState, props, and events form the building blocks of interactivity.
- map() lets you display lists of data cleanly in JSX.
- Always define types for clarity and to prevent bugs.