```text
TypeScript Notes
================

1. Basics
---------
- Superset of JavaScript: adds static typing.
- Files have `.ts` (or `.tsx` for React JSX).
- Compile with `tsc` (TypeScript Compiler) !’ JavaScript output.
- `tsconfig.json` configures the compiler (target, module, strict, etc.).

2. Types
--------
| Type | Description |
|---|---|
| `number` | All numeric values (int, float) |
| `string` | Textual data |
| `boolean` | `true` / `false` |
| `any` | Opt  out of type checking |
| `unknown` | Safer `any`; must be narrowed first |
| `void` | No value (usually for functions) |
| `null` / `undefined`| Absence of value |
| `never` | Function never returns (throws/error) |
| `array` | `type[]` or `Array<type>` |
| `tuple` | Fixed  length array with known types |
| `enum` | Enumerated constant values |
| `object` | Non  primitive values |
| `type` alias | Custom type definitions (`type Alias = … `) |
```

| `interface`  | Shape of objects, can be extended  |

3. Variable Declarations
---
- `let` – mutable block  scoped variable.
- `const` – immutable block  scoped variable.
- `var` – function  scoped (avoid unless required).

```ts
let count: number = 0;
const PI: number = 3.1415;
```

4. Functions
---
```ts
// Parameter types & return type
function add(a: number, b: number): number {
  return a + b;
}

// Optional & default parameters
function greet(name: string, greeting = "Hello"): string {
  return `${greeting}, ${name}!`;
}

// Rest parameters
function sum(...values: number[]): number {
  return values.reduce((a, b) => a + b, 0);
}
```

```ts
// Overloads
function combine(a: string, b: string): string;
function combine(a: number, b: number): number;
function combine(a: any, b: any): any {
  return a + b;
}
```

5. Interfaces
---
```ts
interface Person {
  name: string;
  age?: number;               // optional property
  readonly id: string;        // cannot be reassigned
  greet(): void;
}

// Extending
interface Employee extends Person {
  department: string;
}

// Implementing
class Worker implements Employee {
  readonly id = "E123";
  constructor(public name: string, public department:
string) {}
  greet() { console.log(`Hi, I'm ${this.name}`); }
}
```

6. Type Aliases & Unions
---
```ts
type ID = string | number;                // union type
type Callback = (err: Error | null, data?: any) => void;

type Point = { x: number; y: number };
type Circle = { center: Point; radius: number };
type Shape = Circle | Rectangle;          // discriminated union
```

7. Generics
---
```ts
// Generic function
function identity<T>(value: T): T {
  return value;
}

// Generic constraints
function getLength<T extends { length: number }>(obj: T): number {
  return obj.length;
}

// Generic interfaces / classes
interface Repository<T> {
  getById(id: string): T | null;
  save(item: T): void;
```

```
}

class InMemoryRepo<T> implements Repository<T> {
  private store = new Map<string, T>();
  getById(id: string) { return this.store.get(id) ?? null; }
  save(item: T) { /* ... */ }
}
```

8. Advanced Types
---
- **Intersection Types**: `type A = B & C;`
- **Mapped Types**: `type Readonly<T> = { readonly [K in keyof T]: T[K] };`
- **Conditional Types**: `type IsString<T> = T extends string ? true : false;`
- **Template Literal Types**: `type EventName = `on${Capitalize<string>}`;`
- **Utility Types** (built in): `Partial<T>`, `Required<T>`, `Pick<T, K>`, `Omit<T, K>`, `Record<K, T>`.

9. Modules & Namespaces
---
```ts
// Exporting
export interface User { id: number; name: string; }
export const VERSION = "1.0";

// Importing
import { User, VERSION } from "./models";
import * as utils from "./utils";
```

```

- Prefer ES6 module syntax (`import`/`export`).
- Namespaces (`namespace Foo {}`) are legacy; use modules instead.

10. Decorators (Experimental)
---
```ts
function Log(target: any, propertyKey: string, descriptor: PropertyDescriptor) {
  const original = descriptor.value;
  descriptor.value = function (...args: any[]) {
    console.log(`Calling ${propertyKey} with`, args);
    return original.apply(this, args);
  };
}

class Service {
  @Log
  fetch(id: number) { /* ... */ }
}
```
*Enable `"experimentalDecorators": true` in `tsconfig.json`.*

11. Configuration (`tsconfig.json`)
---
```json
{
  "compilerOptions": {

```
    "target": "ES2022",
    "module": "ESNext",
    "strict": true,
    "noImplicitAny": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "esModuleInterop": true,
    "sourceMap": true,
    "outDir": "./dist",
    "rootDir": "./src",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmitOnError": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

12. Common Patterns
---
- **DTOs** (Data Transfer Objects) via interfaces or `type`.
- **Union discriminators** (`type: 'circle' | 'square'`) for exhaustive `switch`.
- **`as const`** to infer literal types.
- **`unknown` + type guards** for safe runtime checks.

13. Tips & Best Practices
---
- Keep `"strict": true` – it enables the full suite of safety checks.

- Use `readonly` for immutable data structures.
- Prefer `interface` for public API shapes, `type` for unions/computed types.
- Write explicit return types on exported functions.
- Leverage utility types to avoid duplication.
- When interacting with plain JS, use `declare` or `// @ts-ignore` sparingly.
- Run `tsc --watch` during development or integrate with bundlers (Webpack, Vite, Rollup).

--- End of Notes ---
```
```