TypeScript

- Patterns

Patterns

Exhaustive patterns

> In this case, exhaustive means "It covers all cases", e.g. for an enum, so when a new entry is added, it causes an error until this case is considered everywhere.

Example with `never`:

```
enum Status {
  Open,
  InProgress,
  Done,
}

function mapStatus(status: Status): string {
  switch (status) {
      case Status.Open:
        return 'open';
      case Status.InProgress:
        return 'in progress';
      case Status.Done:
        return 'done';
      default:
        const _exhaustiveCheck: never = status;
        return _exhaustiveCheck;
  }
}

// Adding an item to Status causes the error:
// TS2322: Type Status is not assignable to type never
// on _exhaustiveChecks in line 16
```

Example with record and return type

```
enum Status {
  Open,
  InProgress,
  Done,
}

const mapping: Record<Status,string> = {
  [Status.Open]: 'open',
  [Status.InProgress]: 'in progress',
  [Status.Done]: 'done'
}

function mapStatus(status: Status): string{
  return mapping[status];
}

// Adding another item in Status causes:
// TS2741: Property [Status.Another] is missing
// on "mapping" in line 7
```