Self-Study

1. Switch evolution in C# versions , Why C# has the upper hand?

Evolution of the Switch Statement in .NET/C#

C# 1.0–2.0 (.NET Framework 1.0–2.0, ~2002–2005):

Basic switch supported int, char, enum, and (from C# 2.0) string.

Compiler used **jump tables** for dense integer cases (O(1)) or **sequential comparisons** for sparse/string cases (O(n)).

String switches were slow, relying on linear equality checks.

C# 7.0 (.NET Core 2.0, ~2017):

Introduced **pattern matching**, allowing switches on types and when clauses for conditions.

```
switch (obj)
{
   case int i when i > 0: return "Positive";
   case string s: return "String";
}
```

Improved expressiveness, reducing if-else complexity.

C# 8.0 (.NET Core 3.0, ~2019):

Added **switch expressions** for concise, value-returning syntax.

```
var result = day switch

{

   DayOfWeek.Monday => "Workday",
```

```
    _ => "Other"
};
```

Introduced **property** and **tuple patterns**, enabling complex matching.

Optimized string switches with **hash-based lookups**, using dictionary-like structures for O(1) performance.

C# 9.0–11.0 (.NET 5–7, ~2020–2022):

Added **relational** (>, <) and **logical** (and, or) patterns.

```
var result = value switch
{
    < 0 => "Negative",
    >= 0 and <= 100 => "In range"
};
```

List patterns (C# 11.0) for matching arrays/lists.

```
int[] arr = { 1, 2, 3 };
var result = arr switch { [1, .., 3] => "Match" };
```

Further refined hash-based and jump table optimizations.

C# 12.0+ (.NET 8–9, ~2023–2025):

Continued improvements in pattern matching and JIT optimizations, with focus on scalability for large switch blocks.

Why C#'s Switch Has the Upper Hand

1. **Hash-Based String Switches**:
    - For string switches with many cases, C# compiles to a **hash table** or dictionary-like structure, mapping string hash codes to jump targets (O(1) lookup).
    - Collisions are resolved with equality checks, but performance remains near-constant time, unlike linear searches in languages like JavaScript or early Java.
2. **Jump Tables for Integers**:
    - Dense integer or enum cases use jump tables for O(1) performance, optimized by .NET's JIT (RyuJIT).
3. **Pattern Matching**:
    - C#'s pattern matching (types, properties, tuples, lists) is more expressive than traditional switches in C++, Java (pre-17), or JavaScript, reducing code complexity.
4. **Switch Expressions**:
    - Concise syntax for value-returning switches, surpassing verbosity of traditional switches in other languages.
5. **Runtime Optimizations**:
    - .NET's JIT inlines small switches, leverages branch prediction, and supports Native AOT for faster execution.
    - String interning and hash-based lookups make large string switches highly efficient.
6. **Comparison to Other Languages**:
    - **Java**: Lacked pattern matching until Java 17; string switches are less optimized.

- **C++**: Uses jump tables but lacks pattern matching or hash-based string switches.
- **Python**: match (3.10+) is expressive but less performance-optimized.
- **JavaScript**: Basic switches with no advanced optimizations or patterns.