C# Patterns Cheatsheet

Name	Description	Examples
Constant pattern	Matches a specific constant value.	`case 1:` `case "hello":`
Type pattern	Matches a specific type or checks if a value is of a specific type.	`case int i:` `case string s:`
Var pattern	Matches any value and assigns it to a new variable.	`case var x:` `case var (a, b):`
Disjunctive pattern	Matches any pattern in a set of patterns.	`case 0 or 1:` `case "foo" or "bar":`

Conjunctive pattern	Matches a pattern if all subpatterns match.	`case Point { X: 1, Y: 2 }`
		`and { X: 3, Y: 4 }:`
		`case var (x, y) and (x > 0,`
		`y > 0):`

Negated pattern	Matches a pattern if the given pattern does not match.	`case not null:` `case not 0:`

Recursive pattern	Matches a pattern against nested subpatterns.	`case List<int> { Capacity: 0`
		`}:`
		`case (1, (2, 3)):`

Relational pattern (with constant)	Matches values based on a relational condition with a constant value.	`case > 10:` `case <= 5:`
Relational pattern (with type)	Matches values based on a relational condition with a value of a specific type.	`case < (int)DateTime.Now:` `case >= (double)Math.PI:`

Size pattern	Matches an array or collection of a specific size.	`case int[] { Length: 0 }:`
		`case List<int> { Count: 10`
		`}:`

Property pattern	Matches an object based on its properties.	`case Point { X: 1, Y: 2 }:` `case { Length: 0 }:`

Tuple pattern	Matches a tuple or deconstructs a tuple into its individual elements.	`case (int x, int y):`
		`case (int x, int y) when x >`
		`y:`

Positional pattern (with constant)	Matches values based on a positional condition with a constant value.	`case 1, 2:` `case > 10, < 20:`

Positional pattern (with type)	Matches values based on a positional condition with a value of a specific type.	`case int x, int y:`
		`case int x, double y when x`
		`> y:`

Name	Description	Examples

Property pattern (with subpatterns)	Matches an object and applies subpatterns to its properties.	```case Point { X: > 0 and < 10, Y: > 0 and < 10 }:```
		```case { Length: > 0, Capacity: > 10 }:```
		```case { Request.Uri.Scheme: "http" or "https" }```

Recursive pattern (with subpatterns)	Matches a pattern against nested subpatterns, including recursive patterns.	```case List<int> { Capacity: 0, [0]: 0, [1]: 1, [2]: 2 }:``` ```case (1, (2, (3, _))) when _ == 4:```
Relational pattern (with constant pattern)	Matches values based on a relational condition with a constant pattern.	```case < 10:``` ```case > "hello":```
Relational pattern (with type pattern)	Matches values based on a relational condition with a pattern of a specific type.	```case < (int)DateTime.Now:``` ```case > (IEnumerable<int>)new List<int>():```
Size pattern (with range)	Matches an array or collection with a specific size range.	```case int[] { Length: > 0 and <= 10 }:``` ```case List<int> { Count: > 0 and <= 10 }:```
Logical pattern (AND)	Matches a pattern if all subpatterns match.	```case int x and string s:``` ```case int x and (x > 0 and x < 10):```
Logical pattern (OR)	Matches a pattern if at least one subpattern matches.	```case int x or string s:``` ```case int x or (x > 0 and x < 10):```
Logical pattern (NOT)	Matches a pattern if the given pattern does not match.	```case not int x:``` ```case not (x > 0 and x < 10):```

List pattern	Matches a pattern against a sequence of subpatterns.	```case [1, 2, 3]:```
		```case [1, .., 3]:```
		```case [1, .. var s, 3]```