REGEX CHEAT SHEET

regexguide.com/patterns/mac-address-cisco.html

MAC address (Cisco format)

Cisco-style dotted MAC: XXXX.XXXX.XXXX.

```
^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$
```

Token-by-token breakdown

Token	Meaning
`^`	start of string (or line in multiline mode)
`[0-9a-fA-F]{4}`	exactly 4 times: any of: digits, a–f, A–F
`\.`	literal “.”
`[0-9a-fA-F]{4}`	exactly 4 times: any of: digits, a–f, A–F
`\.`	literal “.”
`[0-9a-fA-F]{4}`	exactly 4 times: any of: digits, a–f, A–F
`$`	end of string (or line in multiline mode)

Quick usage in 7 languages

JavaScript

```
const re = /^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$/;
re.test(value);
```

Python

```
import re
re.match(r"^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$", value)
```

Java

```
Pattern.compile("^[0-9a-fA-F]{4}\\.[0-9a-fA-F]{4}\\.[0-9a-fA-F]{4}$").matcher(value).matches();
```

C# / .NET

```
Regex.IsMatch(value, @"^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$");
```

Go

```
regexp.MustCompile(`^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$`).MatchString(value)
```

Ruby

```
/^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$/.match?(value)
```

PHP

```
preg_match('~^[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}\.[0-9a-fA-F]{4}$~', $value);
```

Common pitfalls

- **Anchored to the whole string.** This pattern uses ^ and $, so it requires the entire input to match. To find it inside a longer text, drop the anchors and use the global (g) flag.
- **Case sensitivity.** Letter ranges are case-sensitive — use the i flag if the input case can vary.
- **Escape it correctly per language.** In Java and JavaScript strings each backslash must be doubled (\\d); in Python, Go, and C# use raw/verbatim strings so the backslashes survive.
- **Validate beyond format.** Matching the format doesn't guarantee the value is real. Confirm the mac address (cisco format) against a source of truth (database, API, or checksum) where it matters.