Indian comma format (lakh/crore)

Number in Indian grouping: 1,00,000 not 100,000.

```
^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$
```

Token-by-token breakdown

Token	Meaning
^	start of string (or line in multiline mode)
\d{1,2}	between 1 and 2 times: digit (0–9)
(	start of a capturing group
,	literal text “,”
\d{2}	exactly 2 times: digit (0–9)
)*	end of group, zero or more
,	literal text “,”
\d{3}	exactly 3 times: digit (0–9)
(	start of a capturing group
\.	literal “.”
\d+	one or more: digit (0–9)
)?	end of group, optional (zero or one)
$	end of string (or line in multiline mode)

Quick usage in 7 languages

JavaScript

```
const re = /^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$/;
re.test(value);
```

Python

```
import re
re.match(r"^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$", value)
```

Java

```
Pattern.compile("^\\d{1,2}(,\\d{2})*,\\d{3}(\\.\\d+)?$").matcher(value).matches();
```

C# / .NET

```
Regex.IsMatch(value, @"^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$");
```

Go

```
regexp.MustCompile(`^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$`).MatchString(value)
```

Ruby

```
/^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$/.match?(value)
```

PHP

```
preg_match('~^\d{1,2}(,\d{2})*,\d{3}(\.\d+)?$~', $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.
- **Watch for backtracking.** This pattern nests quantifiers; on adversarial input that can cause exponential backtracking (ReDoS). Test with long non-matching strings, or use an RE2-based engine.
- **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 indian comma format (lakh/crore) against a source of truth (database, API, or checksum) where it matters.