# Literal expressions

```
LITERAL_EXPR : NUMERIC_LITERAL
             | BOOLEAN_LITERAL
```

A *literal expression* is an expression consisting of a single token, rather than a sequence of tokens, that immediately and directly denotes the value it evaluates to, rather than referring to it by name or some other evaluation rule.

## Boolean literals

```
BOOLEAN_LITERAL : "true" | "false"
```

The two values of the boolean type are written `true` and `false`.

## Numeric literals

```
NUMERIC_LITERAL : ( DEC_LITERAL | HEX_LITERAL | BIN_LITERAL ) LITERAL_SUFFIX?

DEC_LITERAL : [0-9] [0-9_]*
HEX_LITERAL : "0x" [a-fA-F0-9_]*
BIN_LITERAL : "0b" [01_]*

LITERAL_SUFFIX : XID_Start XID_Continue*
```

A numeric literal can be written using three bases:

1. Decimal literal starts with a decimal digit and then a mixture of decimal digits and underscores.
2. Hexadecimal literal starts with the character sequence `0x` and continues as a mixture of hex digits and underscores.
3. Binary literal starts with the character sequence `0b` and continues as a mixture of binary digits and underscores.

Underscore character (`_`) is only a visual separator, and it has no influence on the number's value.

> **Note**: Character sequence `_1234` is a valid identifier, not an numeric literal.

> **Note**: Floating-point numbers are not supported.

A numeric literal may be followed (immediately, without any spaces) by a *literal suffix*, which forcibly sets the type of the literal. The literal suffix is any valid identifier which does not start with an underscore (`_`), but only selected values are semantically correct.

The type of *unsuffixed number literal* is determined by type inference:

1. If a numeric type can be *uniquely* determined from the surrounding program context, the unsuffixed numeric literal has that type.
2. If the program context under-constrains the type, it defaults to `felt`.
3. If the program context over-constrains the type, it is considered a static type error.

Examples of numeric literals of various forms:

| Literal | Value | Type |
|---|---|---|
| `1234` | 1234 | `felt` |
| `1234f` | 1234 | `felt` |
| `1234_f` | 1234 | `felt` |
| `1234i` | 1234 | `int` |
| `1234u` | 1234 | `uint` |
| `0x4D2` | 1234 | `felt` |
| `0x4_D2` | 1234 | `felt` |
| `0x_4_D2` | 1234 | `felt` |
| `0b0000_0100_1101_0010` | 1234 | `felt` |

Examples of invalid numeric literals:

- Invalid suffix:
  `1234suffix`
- Use of digits of wrong base:
  `123AFB43`, `0b0102`, `0o0581`
- Binary and hexadecimal literals must have at least one digit:
  `0b_`, `0x____`
- Identifier
  `_1234`

> **Note:** Cairo syntax considers `-1` as an application of the unary minus operator to the numeric literal `1`, rather than a single numeric literal.