

---

## Arithmetic Expressions in Prolog

Prolog comes with a range of predefined arithmetic functions and operators. Expressions such as  $3 + 5$ , for example, are valid Prolog terms.

So, what's happening here?

`?- 3 + 5 = 8.`

No

---

---

## Matching vs. Arithmetic Evaluation

The terms  $3 + 5$  and  $8$  *do not match*. In fact, when we are interested in the sum of the numbers 3 and 5, we can't get it through matching, but we need to use *arithmetic evaluation*.

We have to use the `is`-operator:

```
?- X is 3 + 5.
```

```
X = 8
```

```
Yes
```

---

---

## The is-Operator

The `is`-operator causes the term to its right to be evaluated as an arithmetic expressions and matches the result of that evaluation with the term on the operator's left. (The term on the left should usually be a variable.)

Example:

```
?- Value is 3 * 4 + 5 * 6, OtherValue is Value / 11.
```

```
Value = 42
```

```
OtherValue = 3.81818
```

```
Yes
```

---

---

## Example: Length of a List

Instead of using `length/2` we can now write our own predicate to compute the length of a list:

```
len([], 0).
```

```
len([_ | Tail], N) :-  
    len(Tail, N1),  
    N is N1 + 1.
```

---

## Functions

Prolog provides a number of built-in *arithmetic functions* that can be used with the `is`-operator. See manual for details.

Examples:

```
?- X is max(8, 6) - sqrt(2.25) * 2.
```

```
X = 5
```

```
Yes
```

```
?- X is (47 mod 7) ** 3.
```

```
X = 125
```

```
Yes
```

---

---

## Relations

*Arithmetic relations* are used to compare two arithmetic values.

Example:

```
?- 2 * 3 > sqrt(30).
```

Yes

The following relations are available:

<code>==</code> arithmetic equality	<code>=\=</code> arithmetic inequality
<code>&gt;</code> greater	<code>&gt;=</code> greater or equal
<code>&lt;</code> lower	<code>=&lt;</code> lower or equal

---

---

## Examples

Recall the difference between *matching* and *arithmetic evaluation*:

?- 3 + 5 = 5 + 3.

No

?- 3 + 5 ::= 5 + 3.

Yes

Recall the *operator precedence* of arithmetics:

?- 2 + 3 \* 4 ::= (2 + 3) \* 4.

No

?- 2 + 3 \* 4 ::= 2 + (3 \* 4) .

Yes

---

---

## Defining Operators

New operators are defined using the `op/3`-predicate. This can be done by submitting the operator definition as a query. Terms using the new operator will then be equivalent to terms using the operator as a normal functor, i.e. predicate definitions will work.

For the following example assume the big animals program has previously been compiled:

```
?- op(400, xfx, is_bigger).
```

```
Yes
```

```
?- elephant is_bigger dog.
```

```
Yes
```

---