

# Funcons-beta: Integers \*

The P<sub>L</sub>anCompS Project

`Integers.cbs` | PLAIN | PRETTY

## OUTLINE

### Integers

- Subtypes of integers

- Natural numbers

- Arithmetic

- Comparison

- Conversion

---

\*Suggestions for improvement: [plancomps@gmail.com](mailto:plancomps@gmail.com).  
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

## Integers

[	Type	integers
	Alias	ints
	Type	integers-from
	Alias	from
	Type	integers-up-to
	Alias	up-to
	Type	bounded-integers
	Alias	bounded-ints
	Type	positive-integers
	Alias	pos-ints
	Type	negative-integers
	Alias	neg-ints
	Type	natural-numbers
	Alias	nats
Funcon		natural-successor
	Alias	nat-succ
Funcon		natural-predecessor
	Alias	nat-pred
Funcon		integer-add
	Alias	int-add
Funcon		integer-subtract
	Alias	int-sub
Funcon		integer-multiply
	Alias	int-mul
Funcon		integer-divide
	Alias	int-div
Funcon		integer-modulo
	Alias	int-mod
Funcon		integer-power
	Alias	int-pow
Funcon		integer-absolute-value
	Alias	int-abs
Funcon		integer-negate
	Alias	int-neg
Funcon		integer-is-less
	Alias	is-less
Funcon		integer-is-less-or-equal
	Alias	is-less-or-equal
Funcon		integer-is-greater
	Alias	is-greater
Funcon		integer-is-greater-or-equal
	Alias	is-greater-or-equal
Funcon		binary-natural
	Alias	binary
Funcon		octal-natural
	Alias	octal
Funcon		decimal-natural
	Alias	decimal
Funcon		hexadecimal-natural
	Alias	hexadecimal
Funcon		integer-sequence ]

*Built-in Type*   `integers`

*Alias*   `ints = integers`

`integers` is the type of unbounded integers. Decimal notation is used to express particular integer values.

### Subtypes of integers

*Built-in Type*   `integers-from(_ : integers) <: integers`

*Alias*   `from = integers-from`

`integers-from(M)` is the subtype of integers greater than or equal to *M*.

*Built-in Type*   `integers-up-to(_ : integers) <: integers`

*Alias*   `up-to = integers-up-to`

`integers-up-to(N)` is the subtype of integers less than or equal to *N*.

*Type*   `bounded-integers(M : integers, N : integers)`  
           $\rightsquigarrow$  `integers-from(M) & integers-up-to(N)`

*Alias*   `bounded-ints = bounded-integers`

`bounded-integers(M, N)` is the subtype of integers from *M* to *N*, inclusive.

*Type*   `positive-integers  $\rightsquigarrow$  integers-from(1)`

*Alias*   `pos-ints = positive-integers`

*Type*   `negative-integers  $\rightsquigarrow$  integers-up-to(-1)`

*Alias*   `neg-ints = negative-integers`

### Natural numbers

*Type*   `natural-numbers  $\rightsquigarrow$  integers-from(0)`

*Alias*   `nats = natural-numbers`

*Built-in Funcon*   `natural-successor(N : natural-numbers) :  $\Rightarrow$  natural-numbers`

*Alias*   `nat-succ = natural-successor`

*Built-in Funcon*   `natural-predecessor(_ : natural-numbers) :  $\Rightarrow$  natural-numbers?`

*Alias*   `nat-pred = natural-predecessor`

*Assert*   `natural-predecessor(0) == ( )`

### Arithmetic

*Built-in Funcon*   `integer-add(_ : integers*) :  $\Rightarrow$  integers`

*Alias*   `int-add = integer-add`

*Built-in Funcon*   `integer-subtract(_ : integers, _ : integers) :  $\Rightarrow$  integers`

*Alias*   `int-sub = integer-subtract`

*Built-in Funcon* integer-multiply( $_ : \text{integers}^*$ ) :  $\Rightarrow \text{integers}$

*Alias* int-mul = integer-multiply

*Built-in Funcon* integer-divide( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{integers}^?$

*Alias* int-div = integer-divide

*Assert* integer-divide( $_ : \text{integers}, 0$ ) == ( )

*Built-in Funcon* integer-modulo( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{integers}^?$

*Alias* int-mod = integer-modulo

*Assert* integer-modulo( $_ : \text{integers}, 0$ ) == ( )

*Built-in Funcon* integer-power( $_ : \text{integers}, _ : \text{natural-numbers}$ ) :  $\Rightarrow \text{integers}$

*Alias* int-pow = integer-power

*Built-in Funcon* integer-absolute-value( $_ : \text{integers}$ ) :  $\Rightarrow \text{natural-numbers}$

*Alias* int-abs = integer-absolute-value

*Funcon* integer-negate( $N : \text{integers}$ ) :  $\Rightarrow \text{integers}$

$\rightsquigarrow$  integer-subtract(0,  $N$ )

*Alias* int-neg = integer-negate

## Comparison

*Built-in Funcon* integer-is-less( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{booleans}$

*Alias* is-less = integer-is-less

*Built-in Funcon* integer-is-less-or-equal( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{booleans}$

*Alias* is-less-or-equal = integer-is-less-or-equal

*Built-in Funcon* integer-is-greater( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{booleans}$

*Alias* is-greater = integer-is-greater

*Built-in Funcon* integer-is-greater-or-equal( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{booleans}$

*Alias* is-greater-or-equal = integer-is-greater-or-equal

## Conversion

*Built-in Funcon* binary-natural( $_ : \text{strings}$ ) :  $\Rightarrow \text{natural-numbers}^?$

*Alias* binary = binary-natural

*Built-in Funcon* octal-natural( $_ : \text{strings}$ ) :  $\Rightarrow \text{natural-numbers}^?$

*Alias* octal = octal-natural

*Built-in Funcon* decimal-natural( $_ : \text{strings}$ ) :  $\Rightarrow \text{natural-numbers}^?$

*Alias* decimal = decimal-natural

Literal natural numbers  $N$  are equivalent to **decimal-natural** “ $N$ ”.

*Built-in Funcon*   **hexadecimal-natural**( $_ : \text{strings}$ ) :  $\Rightarrow \text{natural-numbers?}$

*Alias*   **hexadecimal** = **hexadecimal-natural**

*Funcon*   **integer-sequence**( $_ : \text{integers}, _ : \text{integers}$ ) :  $\Rightarrow \text{integers}^*$

**integer-sequence**( $M, N$ ) is the sequence of integers from  $M$  to  $N$ , except that if  $M$  is greater than  $N$ , it is the empty sequence.

*Rule*   
$$\frac{\text{is-greater}(M, N) == \text{false}}{\text{integer-sequence}(M : \text{integers}, N : \text{integers}) \rightsquigarrow (M, \text{integer-sequence}(\text{integer-add}(M, 1), N))}$$

*Rule*   
$$\frac{\text{is-greater}(M, N) == \text{true}}{\text{integer-sequence}(M : \text{integers}, N : \text{integers}) \rightsquigarrow ( )}$$