Kozmos Operators, Constants, and Data Structures

The following chart lists the built-in operators, constants, and data structures of the **Kozmos** programming language:

Category	Purpose	Operator
Arithmetic	addition	`a + b`
	subtraction	`a - b`
	multiplication	`a * b`
	division	`a / b`
		`a div b`
	modulus	`a mod b`
	extrema	`a min b` -or- `min(a, b, c, ...)`
		`a max b` -or- `max(a, b, c, ...)`
Relational	equals	`a = b`
	not equals	`a <> b`
	less than	`a < b`
	greater than	`a > b`
	at least	`a >= b`
	at most	`a <= b`
Logical	conjunction	`a and b`
	disjunction	`a or b`
	equivalence	`a eqv b`
	implication	`a imp b`
	negation	`not a`
Bitwise	and	`a & b`
	inclusive or	`a | b`
	exclusive or	`a ! b`
	not	`~a`
	left shift	`a << b`
	right shift	`a >> b`

The following are the assignment operators:

Category	Syntax
Simple	`a := b`
Compound	`A[i] :<operator>= v`
	`{l:_} :<operator>= v`

Kozmos provides the following built-in constants:

Category	Constant
Arithmetic	`+Inf`
	`-Inf`
	`NaN`
Boolean	`True`
	`False`
Reference	`Nil`

Kozmos has the following built-in data structures:

Category	Declaration
Array	`Array<T>`
Linked list	`List<T>`
Hash Map	`Map<K -> V>`
Hash Set	`Set<T>`
Priority Queue	`Queue<T>`
Minima Heap	`Heap<T>`

> **NOTE**: **Kozmos** does not support general-purpose *generics*. Instead, it has *trait compliance* where the type parameter has to be with one of the built-in root traits such as `Eq`, `Ord`, `Sync`, etc., or a user designed trait that implements those root traits.
> In a data structure declaration, the type parameter has to be a known trait, e.g. `<Ord>`; a descendant of a trait, e.g. `<T: Ord>`; or a descendant of multiple traits (i.e. a *union* of them), e.g. `<T: Ord | Sync>`.

The following operations are available for built-in data structures

Category	Subcategory	Operator
Catenation	Two arrays (Strings incl.)	`a ++ b`

Category	Subcategory	Operator
	Array with single item	`a ++ [b]`
Slicing		`a[i .. j]`
Range		`[a .. b]`
Indexing		`a[i]`
List access	Head	`{l:_}`
	Tail	`{_:l}`
	Append	`{a} ++ b`
	Prepend	`b ++ {a}`

Kozmos does **not** support operator *overloading*. Instead, it supports operator *extension*: operators can be extended for a certain type (i.e. a `record`, a `trait`, or a `class`) only on the condition that the expression reduces to one of the built-in uses.