## Numbers

| | |
|---|---|
| `parse- integer` | Parses an integer from a string |
| `floor`, `ceiling` | Truncates toward negative/positive infinity |
| `round` | Rounds to the nearest integer |
| `mod`, `rem` | Modulus/remainder of a truncating division |
| `1+`, `1-` | Returns a new value (inc/dec)remented by a number |
| `incf`, `decf` | ❗ Macros to (inc/dec)rement in place |
| `=`, `/=` | Mathematical value equality (types ignored) |
| `>`, `>=`, `<`, `<=` | Order comparison |

## Characters

## Strings

| | |
|---|---|
| `format` | Returns a formatted output string when the *destination* is `nil` |

## Sequences

| | |
|---|---|
| `elt` | Access elements with an integer index |
| `length` | |
| `count` | Count the appearances of an element |
| `find` | Finds an item or returns `nil` |
| `position` | The first index of an element |
| `search` | Returns the start position of a sequence in another sequence |
| `remove` | Removes all the occurrences of an item |
| `substitute` | Replaces all occurrences of an item |

## Lists

| | |
|---|---|
| `mapcar` | Successively applies a function to each element of a list |
| `maplist` | Like `mapcar`, but its function receives a **cons** cell instead of its element (therefore, the function has access to the rest of the list) |
| `mapcan` | Like `mapcar`, but the results are combined into a list |
| `mapcon` | Like `mapcan`, but applied to `maplist` |

## Vectors

## Hash Tables

## Things to remember

| | |
|---|---|
| Object Oriented | All values in a Common Lisp program are instances of some **class**. Furthermore, all classes are organized into a single hierarchy rooted at the class `T`. |
| Starting a project | SBCL and Quicklisp installation and a project template |

## Equality

| | |
|---|---|
| `eq` | Object identity (don't use with numbers or chars) |
| `eql` | 👍 Considers equivalent two objects of the same class with the same numeric or char value |
| `equal` | Lists with the same structure and contents, strings with the same characters |
| `equalp` | Ignores differences in case or in numerical type (`1 == 1.0`) |

## Functions

| | |
|---|---|
| `defun` | |
| `lambda` | Returns an anonymous function |
| `return -from` | Immediately returns a value from a function (or a `block`) |
| `funcall` | Invokes a function from a function object |
| `apply` | Works like `funcall`, but receives the arguments as a list |
| `&o ptional`, `&key`, `&a llo w-o the r-keys` | Different ways to capture function arguments |

## Standard Control Constructs

| | |
|---|---|
| `if` | The *else* form is optional |
| `when` | Like `if`, but returns `nil` if the condition is *falsy* and evaluates multiple body forms |
| `unless` | Like `when`, but executes its body only when the condition is *falsy* |

## IO

| | |
|---|---|
| `format` | Produces formatted output to *stdout* when the *destination* is `t` |
| **`format` language** | |
| `a`, `s`, | Interpolate argument in human readable format; interpolate as *Lisp* readable |
| `%`, `&` | Newline; newline if not at the beginning of a line |

## Object-Oriented

| | |
|---|---|
| `defgeneric` | Defines an abstract operation (*polymorphism*) |
| `defmethod` | Defines an implementation of a generic function |
| `call-n ext -meth od` | Similar to an invocation to a super-class method |
| `defclass` | New named class; some slot options are: `:reader`, `:writer`, `:accessor`, `:initar g`, `:initform`, `:docum ent ation`, `:al loc ation` |
| `slot-value` | Returns the value of slot in the object (**`setf`**-able) |
| `with-slots`, `with-a cce ssors` | Binds a slot/accessor to a symbol that can be used in its body |

By **Andre Boechat**
(boechat107)

cheatography.com/boechat107/
boechat107.github.io/