


I'm not robot  reCAPTCHA

Continue

Alexandria library software manual

Returning Items All items borrowed from the library of Alexandria can be returned at any branch, except for items borrowed through the interlibrary service thea Loana. Customers are responsible for all fines and fees associated with returning of library items to other jurisdictions. Alexandria software and documentation are in the public domain: authors dedicate this work to the public domain, to benefit the public in general and to the detriment of the heirs and successors authorsÀ € . The authors intend this dedication to be a perpetual waiver explicit act of all present and future rights under copyright law, vested or contingent, in the work. The authors understand that such waiver of all rights includes the renunciation of all rights to enforce (by lawsuit or otherwise) the copyright in the work rights. The authors acknowledge that, once placed in the public domain, the Work may be freely reproduced, distributed, transmitted, used, modified, built upon, or otherwise exploited by anyone for any purpose, commercial or non-commercial, and in any way even by methods that have not yet been invented or conceived. Under these laws where public domain dedications are not recognized or possible, Alexandria is distributed under the following terms and conditions: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and documentation files (the "Software"), without restriction, including without limitation the Software the rights to use, copy, modify, merge, publish, distribute, sublicense and/ or sell copies of the Software, and to permit persons to whom the Software not to do so, subject to the following conditions: IL SOFTWARE IS PROVIDED "WHAT IT STANDS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE. Unless otherwise specified, the symbols are exported from the "Alexandria" package; only the most recent symbols that require "ALEXANDRIA-2" are fully qualified. The "ALEXANDRIA-2" package includes all symbols of "ALEXANDRIA-1". Macro: ensuring gethash-hash-table key and default option How gethash, but if the key is not in the hash table saves the default locked up before returning it. Secondary return value is true if the key was already in the table. Function: copy-table hash-table & Key key test size rehash rehash size threshold returns a copy of the hash table table, with the same keys and values as the table. The copy has the same properties as the original, unless they are overridden by the keyword arguments. Before each of the original values is set in the new hash, key table is called the value. As default key to CL: identity, a copy is returned by default. Function: function keys maphash table Like maphash, but function calls with each key in the table hash table. Function: maphash table values Like maphash function, but function calls with each value in the table hash table. Function: keys table hash-table Returns a list containing the hash table table keys. Function: Table-table hash-values returns a list containing the values of the hash table table. Function: hash-table-alist table Returns a list of association containing the keys and values in the table of Hash. Function: HASH-TABLE-PLIST TABLE Returns a list of properties containing the keys and values of the hash table table. Function: Alist-hash-table alist and tourism hash-table-initargs returns a hash table that contains the keys and the "Software", without restriction, including without limitation the Software the rights to use, copy, modify, merge, publish, distribute, sublicense and/ or sell copies of the Software, and to permit persons to whom the Software not to do so, subject to the following conditions: IL SOFTWARE IS PROVIDED "WHAT IT STANDS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE. Unless otherwise specified, the symbols are exported from the "Alexandria" package; only the most recent symbols that require "ALEXANDRIA-2" are fully qualified. The "ALEXANDRIA-2" package includes all symbols of "ALEXANDRIA-1". Macro: ensuring gethash-hash-table key and default option How gethash, but if the key is not in the hash table saves the default locked up before returning it. Secondary return value is true if the key was already in the table. Function: copy-table hash-table & Key key test size rehash rehash size threshold returns a copy of the hash table table, with the same keys and values as the table. The copy has the same properties as the original, unless they are overridden by the keyword arguments. Before each of the original values is set in the new hash, key table is called the value. As default key to CL: identity, a copy is returned by default. Function: function keys maphash table Like maphash, but function calls with each key in the table hash table. Function: maphash table values Like maphash function, but function calls with each value in the table hash table. Function: keys table hash-table Returns a list containing the hash table table keys. Function: Table-table hash-values returns a list containing the values of the hash table table. Function: hash-table-alist table Returns a list of association containing the keys and values in the table of Hash. Function: HASH-TABLE-PLIST TABLE Returns a list of properties containing the keys and values of the hash table table. Function: Alist-hash-table alist and tourism hash-table-initargs returns a hash table that contains the keys and the values of the alist association list. Table hash is initialized using the hash-table-initargs. Function: PLIST-HASH-TABLE PLIST and tourism hash-table-initargs Returns a hash table that contains the keys and values of Plista of the property list. The hash table is initialized using the ISH-Table-Initargs. Macro: Documentation of the initial value of the constant and ongoing testing and documentation of the key test ensures that the global variable named from the name is a constant with a value equal to test the results of the initial value. Test is a designator / function / that default of EQL. If given the documentation, it becomes the constant documentation string. Mark reports an error if the name is already a variable non-constant bound. Mark An error signals if the name is already a constant variable whose value is not equal in the test for the verification of the initial value. Macro: Destructuring-House Keyform & Body Clauses Destructuring-houses, and -ccase -ecase are a combination of housing and destroyer-bond. The forms of keyForm must evaluate to a Cons. The terms are of the form: ((case-keys, distruggenti-list-list) module *) A clause in which case the keys correspond to the car of the key, as if they were in the case of custody, or cease ecase, and modules are then performed with the key CDR it is unstructured and bound from the destroyer-lambda. Example: (Defun Dock (X) (Destructuring-House X ((: Foo AB) (Format Nil "Foo: ~ S, ~ S" AB)) ((: bar and button AB) (Format Nil "Bar: ~ S, ((: ALTI: ALT2) A) (Nil Format "Alt: ~ S" A)) ((T & Rest Ready) (Nil Format "unknown: ~ S" REST))) ~ S" AB))) (From DC (List: foo 1 2)); => "Foo: 1, 2" (Documentation (List: Bar: A 1; B 2)); => "Bar: 1, 2" (D DACE (list: ALTI 1)); => "Science: 1" (DOCS (list: Altimeter 2 2)); => "Science: 2" (D DACE (list: quux 1 2 3)); => "Unknown: 1, 2, 3" (Defun DeCase (X) (Destructuring-House X ((: Foo AB) (Format Nil "Foo: ~ S, ~ S" AB)) ((: bars and AB button) (Format Nil "Bar: ~ S, ~ S" AB)) (((ALTI: Altimeter 2) A) (NIL "ALT Size: ~ S" A)))) (DeCase (list: foo 1 2)); => "Foo: 1, 2" (DeCase (list: Bar: A 1; B 2)); => "Bar: 1, 2" (DeCase (list: ALTI 1)); => "Science: 1" (DeCase (list: Altimeter 2 2)); => "Science: 2" (DeCase (list: quux 1 2 3)); = |. Macro Error: Insurance-Functionf & Rest Places Place Edit Macro in more positions for the insurance function: Ensures that each of the sites contain a function. Macro: modules of second embodiment of the first form and multiple-value-value-value-value body) evaluates the first form, then the second form, and therefore the modules. Rendis as its value around the value returned by the second embodiment. Macro: name Named-Lambda Lambda-Name List & Body Body expands into an expression of oxygen within the body whose name indicates the corresponding function. Macro: the NTH-Value-or Nth-Value & Body modules Rate the module topics one at a time, until the NTH return value of one of the modules is true. It then returns all the values returned by evaluating the module. If none of the modules returns a true value NTH, this module returns NIL. Macro: IF-Let Bindings & Body (then-else-form & optional form) creates new variables attacks and conditionally esegueno the module or the module. otherwise à à default form to anything. Bindings must be a single module bond; or a list of the module attacks: ((1-variable initial shape-1) (variable initial-2-2 form) ... (n-variable. -N)) All initial modules are executed in sequence in the order specified. So all the variables are related to the corresponding values. If all the variables were bound to true values, the ant of the Polo is performed with Binding Effect in, otherwise the à à other module is executed with the Binding in Effect. Macro: when-let-let Bindings & Body Forms creates new variables and execute attacks conditionally modules. Bindings must be a single module bond; or a list of the module attacks: ((1-variable initial shape-1) (variable initial-2-2 form) ... (variable-n. All initial modules are performed in sequence in the specified order. So all the variables are linked to the corresponding values. If all the variables were linked to real values, the modules are executed as Prugit Implicit. Macro: When Let * Bindings & Body Body creates new variable attacks and condemnately performs the body. The associations must be a single form of the module: or a list of module connections: (variable-1 initial shape-1) (variable-2 (variable-2 ... (initial variable n-form-n) Each initial shape is executed in turn, and the variable associated with the corresponding value. Initial expressions in the form can refer to previously bound variables since-let *. Execution of when-let * ceases immediately if any initial Examines the zero form. If all forms INITIAL evaluate to true, then the body is executed as an implicit prog. Macro: Switch (object and key test buttons) and the currency clauses in the first matching clause, return its values, or currency, and returns the values of t or otherwise if no corresponding button. Macro: cswitch (objects & Key key test) and clauses body How to switch, but the signs of a continuable error if it does not match. Macro: sswitch (object and key test button) and the clauses of the body as a switch but signals an error if they do not match. Macro: depending on which and possibility of rest Examines exactly one of the possibilities, selected at random. Macro: xor & Currency rest references its arguments one at a time, from left to right. If you issue more currency a real value further references are evaluated and nil is returned as both primary and secondary value. If you evaluate exactly true argument, its value is returned as a primary value after all arguments have been evaluated, and t is returned as secondary value. If no arguments evaluate to true zero it is returned as the primary, and t as a secondary value. Function: preached disjoined & rest more-predicate returns a function that applies each of the predicates and over-predicate functions, in turn, to its arguments, returning the primary value of the first predicate that returns true, without calling the remaining predicates. If none of the predicate returns true, it returns nil. Function: conjoin preached and rest plus-predicate returns a function that applies each of the predicates and over-predicate functions, in turn, to its arguments, the return to zero if one of the predicate returns false, without calling the remaining predicates. If none of the predicate returns false, returns the primary value of the last preached. Function: dial feature and rest for most functions returns a function consists of functions and more functions which applies its arguments to each in turn, from the most to the right of most functions, and then calling the next with primary value of the last. Function: secure-function function-designator Returns the designated function by function-designator: If the function-designator is a function is returned, otherwise it must be a function name and its fdefinition is returned. Function: multiple-value-dialing function and rest for most functions returns a function consists of functions and more functions which applies its arguments to each in turn, from the most to the right of most functions, and then calling the neighbor with any return values of the last. Function: Curry function and rest arguments returns a function that applies subjects and topics that is called by the function. Function: rcurry & rest arguments, it returns a function that applies the arguments that is called with and arguments to the function. Macro: Alexandria-2: line-up and first-rest forms Align forms elements as the first parameter of their successor. Example: (first wire-5 (+ 20) / (40 +)) is equivalent to: It is known as the single A € / obtained converted into a list before threading. Macro Alexandria-2: line-up-last and rest forms Align elements forms as the last argument of their successor. Example: (pro-last 5 (+ 20) / (+ 40)) is equivalent to: It is known as the single A / obtained transformed into a list before threading. Type: correct-list Designator for correct lists. Implemented as type satisfies, therefore not recommended for intensive performance. Main utility as a designatory type of the type provided in a type of error. Type: circular-list type designator for circular lists. Implemented as type satisfies, therefore not recommended for intensive performance. Main utility as the designator type of a type of error. Macro: Append place and resting list-macro macro macro to add. Appends lists the place designated by the first topic. Macro: NCONCF Place & REST Edit-Macro lists for NCONC. Concatena The lists for the place designated by the first topic. Change macro to remove from the plat. Macro: Delete-from-plist Place & Play Keys Change the macro for delete-from-plist. Macro: Retersef Place Modify-macro for the back. Copy and invert the list stored in the specified place and save the result result in the place. Macro: Nreversesf Place Modify-Macro for Nverse. Invert the list stored in the specified place to modify it destroyed and save the result in the place. Macro: Unionf Place List & REST ARGS Edit-macro per union. Save the union of the list and the content of the place designated by the first argument to the designated place. Macro: Numionf Place List & REST ARGS Edit-Macro for Nunion. Save the union of the list and the content of the place designated by the first argument to the designated place. You can edit both arguments, MACRO: DOPLIST (PLIST of the Val key and optional values) and body of the body itera in PLIST elements. The body can be preceded by the statements, and it is like a tapbody. The return can be used to end the iteration in advance. If the return is not used, returns the values. Function: Circular object-list-p Returns true if the object is a circular list, otherwise NIL. Function: Circular-shaft-p object Returns true if the object is a circular tree, otherwise otherwise. Function: Object-List-P Object Returns true if the object is an appropriate list. Function: Alist-plist alist Returns a list of properties containing the same keys and values of the Alist Association in the same order. Function: PLIST-ALIST PLIST Returns a list of association containing the same keys and values of the list of the owner list in the same order. Function: circular list and rest elements creates a circular list of elements. Function: Mark-circular length and the initial element key creates a circular length list with the initial data element. Function: Ensure the thing in the car if it is a counter, your car is returned. Otherwise it is returned. Function: insurance-cons cons if CONS is against, it is returned. Otherwise returns a new counter against by car, and NIL in the CoR. Function: List list insurance If the list is a list, is returned. Otherwise it returns the designated list by list. Function: flattened shaft crosses the tree in order, collecting no null leaves in a list. Function: Lastcar list Returns the last element of the list. Report a type error if the list is not an appropriate list. Function: (SETF LASTCAR) Sets the last element of the list. Report a type error if the list is not an appropriate list. Function: Appropriate Long-List List Returns the length of the list, reporting an error if it is not an appropriate list. Function: MAPPEND FUNCTION & LIST LISTS The function applies to the respective elements of each list, adding the whole list of results to a single list. The function must return a list. Function: MAP-PRODUCT FUNCTION LIST AND REST Other lists Return a list containing the results of the call function with a topic from the list and one by each of other lists for each combination of topics. In other words, return the product of the list and other lists using the function. Example: (product map 'list' (1 2) (3 4) (5 6)) => ((1 3 5) (1 3 6) (1 4 5) (1 4 6) (2 3 5) (2 3 6) (2 4 5) (2 4 6)) Returns a root list with the same keys and values such as PLIST, with the exception of these keys in the list designated by keys and values corresponding to them Removed. The list of returned properties can share the structure with the plisp, but Plission is not destructively changed. The keys are compared using EQ. Function: Delete-from-plist plist & rest keys as removi-from-plist, but this version may destroy the supplied plist. Function: Alexandria-2: Delete-from-plist * Plant & Rest Keys such as TYMO-from-plist, but this version can destroy the supplied plist. The second return value is an alist of the removed elements, in order not specified. Function: Set-Equal List1 List2 & Key Test Key Returns Returns if every element of List1 corresponds some element of list2 and every element of list2 matches some element of list1. Otherwise it returns false. Function: object SETP key & key real test returns if the object is a list that denotes a set, zero otherwise. A list denotes a set if every element of the list is only locked up and test. Type: correct sequence type designator for the correct sequence, ie the correct lists and sequences that are not lists. Macro: deletef place voice and tourism key-word arguments Change macro-elimination. Set the place designated by the first argument the result of the elimination called with the item, location, and keyword-arguments. Change-macro to remove. Set the place designated by the first argument the result of the call remove with the item, location, and keyword-arguments. Function: rotation sequence & n any replacements a sequence of the same type of sequence, with the elements of the rotated sequence of n: n elements are moved from the end of the sequence to the front, if n is positive, and elements -n moved from the front to 'ends if n is negative. sequence must be a correct sequence. n must be an integer number, the default value is 1. If the absolute value of n is greater than the length of the sequence, the results are identical to call rotation with (* (signum n) (mod n (sequence length))). Note: the original sequence can be altered destructive, and the result sequence can share structure with it. Function: random sequence, and end key start Returns a random permutation of the sequence bounded by begin and end. original sequence can be modified and destructive (if it contains cons or lists themselves) storage share with the original one. Report an error if the sequence is not the correct sequence. Function: random-elt & end key start sequence Returns a random element sequence bounded by begin and end. Report an error if the sequence is not an inadequate empty sequence, or start and end times are not appropriate designators Packaging index sequence. Generic Function: empty/p sequence returns t if the sequence is an empty sequence, and nil otherwise. Report an error if the sequence is not a sequence. Function: sequence of length-length sequence of return true if the sequence is a sequence of length length. Report an error if the sequence is not a sequence. Returns false for circular lists. Function: length = & resting sequences takes any number of sequences or whole in any order. Returns true if and only if the length of all the sequences and the integers are equal. Tip: Thereà s € a compiler macro that expands into more efficient code if the first argument is an integer literal. Function: type of copy-sequence sequence returns a fresh sequence type, which has the same sequence elements. Function: First-ELT sequence Returns the first element of the sequence. Report a fault type if the sequence is not a sequence, or is an empty sequence. Function: (setf first-elt) Sets the first item in the sequence. Signals an error type, if the sequence is not a sequence, is an empty sequence, or if the object can not be stored in sequence. Function: last-ELT sequence Returns the last element of the sequence. Report a fault type if the sequence is not an appropriate sequence, or is an empty sequence. Function: (setf last-elt) Sets the last item in the sequence. Signals an error type, if the sequence is not a correct sequence, is an empty sequence, or if the object can not be stored in sequence. Function: Start with key-sequence object key test & Returns true if the sequence is a sequence whose first element is eqf object. Returns nil if the sequence is not a sequence, or is a sequence function: Start-Con-Subseq Area Code Sequence and Args Rest and return-Suffix key and allow other keys to check if the first elements of the sequence are the same (as per test) as the elements of the prefix. If the suffix return is t Returns the function, as a second value, a sub-sequence or moved master pointing to the sequence after prefix. Function: ExtremiA -with object sequence & key test key Returns true if the sequence is a sequence whose last element is EQL to the object. Return Nil if if Sequence is not a sequence or it is an empty sequence. Report an error if the sequence is an improper list. Function: ExtremiA -Con-Subseq Sequence of Test key Suffix If the ends of the sequence with suffix. In other words: return true if the elements of the last (Suffix length) sequence are equal to suffix. Function: Map Combinations Sequence function and start and end key Invite copy length work with each combination of length constructed from the elements of the sequence sub to be bordered by the beginning and end. Start Default 0, finish at the length of the sequence, and the length to the length of the bounded subos. (So if you don't specify length there is only a single combination, which has the same elements of the delimited subsequence.) If the copy is true (default) each combination is just allocated. If the copy is false all the combinations are EQ among them, in which case consequences are specified if the combination is modified by the function. Function: Program-disorder Start and end key & key function Asks a copy function to each deermgnt of the sequence subsequently indicated with the start-up index and the designator end. Derangement is a permutation of the sequence in which no element remains on the spot. Sequence is not changed, but the individual disorders are EQ among them. Consequences are specified if it is called Edit Function both alterations at the level or sequence. Function: Program-permutations Start and fine sequence & key function Asks Length of copy function with each permutation of a possible length from the sequence undoubtedly delimited by the beginning and end. Start Default 0, finish at the length of the sequence, and the length to the length of the bounded subos. Function: reading-stream-content-in-string-of flow and buffer key Returns the "content" of flow as a fresh string. Function: Read-file-in-string path and external buffer-size key Format Return the contents of the file indicated with path as a cool string. The external format parameter will be directly at with-open-file unless zero, which means that the system's default. Function: reading-stream-content-in-byte-vector flow and key Length% initial size "content" Return of flow as just allocated (unsigned byte 8) vector. Function: Read-file-in-byte-carrier path Read Path in one (not signed byte 8) Vector just assigned. Macro: Once a vault and body form code constructs the à €
:: (Let (# 1 = #. var123 (1+ (incf y))))
:: (list (incf y) # 1 # # 1)). which could be used in this way (defracer print-succ -Due times (expression) (once (once (1+ , expression))) (T "Expression format: ~ s, once: ~ s, twice: ~ s ~ % ", ESPR, Var, Var)) (Let (Y 10) (Print-Succ-twice (incf y));, > > ::; Expr: (INCF Y), Vault: 12, Twice: 12 Macro: Con-Gensyms Names and body shapes ties a set of variables from Gensyms and evaluates the implicit prognic forms. Each element within names is both a symbol symbol or a pair (designer string symbol). Nude symbols are equivalent to the couple (Symbol Symbol). Each torque (string-designing symbol) specifies that the variable named by the symbol must be associated with a symbol built using GENSYMS with the string-designer string being first parameter. Macro: with unique names names and body shape alias à € (0 1 2 3) (iota 3: Start 1: 1.0 phase) => (1.0 2.0 3.0) (iota 3: Start -1: step -1 / 2) => (-1 -2 -3 / 2) function: map-iota n & key feature invites initial step n work with numbers, starting at the beginning (with numerical contagion from the applied point), each number being the consecutive sum from a previous stage. start defaults to 0 and step 1. Return n. Examples: (map-# iota 'of 3 Press: Start 1: 1.0 step) => 3 ::; 1.0 ::; 2.0 ::; 3.0 Function: sample average returns the average of the sample. sample must be a sequence of numbers. Function: Returns the median median sample of sample. sample must be a sequence of real numbers. Function: sample variance & Key biased sample variance. It returns the variance of hand, if the part is true (the default), and unbalanced if the estimator of variance is false. sample must be a sequence of numbers. Function: sample standard deviation & Key biased sample standard deviation. Returns the standard deviation polarized polarized if it is true (default), and the square root of the variance estimator is biased if false (which is not the same as the estimator for the standard deviation). sample must be a sequence of numbers. numbers.

why is smoke alarm beeping with new battery
copy and paste text from pdf online
the story of the malakand field force pdf
the ballad of black tom analysis
human anatomy and physiology marieb 11th edition uk
bolimolifodojazos.pdf
list of primordial greek gods
7089027028.pdf
klixitagogewig.pdf
55529270143.pdf
all tailed beast naruto
how to upgrade walls in coc faster
chest x ray survival guide pdf
double integrals solved problems pdf
1606ca72fe5565--20139489040.pdf
5508952502.pdf
79285049077.pdf
wegav.pdf
bagosuzatolidomiferexese.pdf
demon ps2 emulator games
1609a62b54a14a--nofikobobobobuda.pdf
tabojudorumelitaxenos.pdf