Module sc

This is the 2011 revamp of scutil's erlang library.

Version: 2.5.757

Introduced in: September 14, 2007

Authors: John Haugeland (stonecypher@gmail.com).

Description

This is the 2011 revamp of scutil's erlang library.

I'm modernizing my library. Originally it got too big, so I split it up. However, that's proven more trouble than it's worth - I often found myself asking whether the thing I wanted was in `_math`, `_correlate`, `_statistics`, `_lists`, `_operators`, etc. No more. It's all getting dumped into `sc.erl` and `sc_tests.erl`. Prototypes are pushed into `sc_prototype.erl`; that is code which is either insufficiently tested or unfinished. It's moving to eunit; sc_test is being dumped. I'm taking a lot of the customization stuff out of the generated docs, too. There was always an emphasis on testing, but it's getting taken much more seriously this time. I am also no longer a language novice; I can replace at least some of the naive implementations with marginally less naive ones, break fewer conventions, replicate fewer extant library functions, choose better naming and a more appropriate argument order, et cetera.

This also means I can completely ditch all the remnants of the botched move to packages, provide radically less automation which is radically more effective, and generally do a better job of

presenting the actual volume of functionality present here. We can get the coverage tools out. We can run prototypical work in a single separate file, to distinguish quality. Etc.

Generally, this is intended to just be an all around improvement and replacement for the library I was pretty happy with, but which was showing strain.

Thanks

ScUtil has profited significantly from the donations of time, understanding and code given by a variety of generous friends and strangers. The list of small tweaks would be prohibitive, but significant influence on this library is due the following people, in alphabetical order (the least fair of all generic orderings):

- Alain O'Dea of Concise Software
- Alisdair Sullivan
- Aristid Breitkreuz / [MrN, MisterN]
- Ayrnieu
- Bryon Vandiver of Sublab Research and Design
- Chile
- Dave Murphy / WinterMute
- DizzyD
- Dylan Barrie / PhforSlayer
- Essen
- Geoff Cant / Archaelus
- GrizzlyAdams of The Waffle Iron
- Jeff Katz / Kraln
- John Sensebe of Bargaintuan
- Orbitz
- raleigh
- Richard Carlsson
- Robert Virding
- Steve Stair

- Steve Vinoski
- Torbj*rn T*rnkvist (those asterisks are o-umlauts, until I work out a bug in edoc) / Etnt
- Toby Opferman
- Vat Raghavan
- Vladimir Sessikov

Data Types

bw_rate()

```
bw_rate() = {Scale::bw_scale(), Rate::float()}
```

`bw_rate` - Bandwidth rate - is a rate-in-units-per-time notation for bandwidth measurement, eg `{{megabits,day},10.5}`.

bw_scale()

```
bw_scale() = {Unit::atom(), Timescale::atom()}
```

`bw_scale` - Bandwidth scale - is a units-per-time notation for bandwidth measurement, eg `{megabits,day}`.

byte()

```
byte() = integer()
```

A byte must be an integer in the range 0-255, inclusive. (Technically this is an octet, not a byte, but the word byte is extensively misused throughout the erlang documentation and standard library, which makes this an important concession, so we're when-in-Rome-ing.)

coord()

```
coord() = tuple()
```

Every member of a `coord()` is a `number()`. Represents a coordinate, which may imply a sized cartesian space. Many functions expect integer coordinates; the type does not require them. This type does not define member count. If your function requires a specific count of members, name it, as in a `coord2()` or `coord3()`.

coord2()

```
coord2() = {number(), number()}
```

Represents a coordinate, which may imply a sized rectangle. Many functions expect integer coordinates; the type does not require them.

coord2list()

```
coord2list() = list()
```

All members of a `coord2list()` must be `coord2()`s.

coord3()

```
coord3() = {number(), number(), number()}
```

Represents a coordinate, which may imply a sized 3d box region. Many functions expect integer coordinates; the type does not require them.

coord3list()

```
coord3list() = list()
```

All members of a `coord3list()` must be `coord3()`s.

coordlist()

```
coordlist() = list()
```

All members of a `coordlist()` must be `coord()`s. All member coordinates must be of the same size, though this type does not define what that size is. If your function requires a specific count of members, name it, as in a `coord2list()` or `coord3list()`.

filterfunction()

```
filterfunction() = function()
```

Filter functions are 1ary binary predicates - they accept an argument and return either true or false.

function_or_list()

```
function_or_list() = function() | list()
```

gridsize()

```
gridsize() = coord2() | integer()
```

Coordinates are the width and height of a (1,1) originated grid; as such, coordinates are of the range [1,X] , [1,Y] inclusive, and returned in the form {A,B}. The integer form implies a square grid.

hexchar()

```
hexchar() = integer()
```

Integer must be in the range $0 - $9, the range $a - $f, or the range $A - $F, all inclusive, for inputs; outputs will always use lower case.

hexstring()

```
hexstring() = list()
```

All elements of the list must be of type `hexchar()`.

io_list()

```
io_list() = list()
```

Every list member of an `io_list()` must be a `byte()`.

keylist()

```
keylist() = keylist()
```

All members of keylists are tuples of two-or-greater arity, and the first element is considered their key in the list. List keys are unique; therefore `[{a,1},{b,1}]` is a keylist, but `[{a,1},{a,1}]` is not.

list_of_lists()

```
list_of_lists() = list()
```

Every member of a list_of_lists() is a list().

list_or_binary()

```
list_or_binary() = list() | binary()
```

It's either a list() or a binary(). Duh.

non_negative_integer()

```
non_negative_integer() = integer()
```

A non_negative_integer() must be equal to or greater than zero.

numeric_tuple()

```
numeric_tuple() = tuple()
```

Every member of a numeric_tuple() must be a number().

numericlist()

```
numericlist() = list()
```

All members of a numeric list must be number()s.

nybble()

```
nybble() = integer()
```

A nybble must be an integer in the range 0-15, inclusive.

positive_integer()

```
positive_integer() = integer()
```

Positive integer must be greater than zero.

positive_integer()

```
positive_integer() = integer()
```

A positive_integer() must be greater than zero.

positive_integer()

```
positive_integer() = integer()
```

This integer must be greater than zero.

positive_integer_or_list()

```
positive_integer_or_list() = positive_integer() |
list()
```

ranking()

```
ranking() = {Ranking::number(), Value::any()}
```

Values are usually number()s, but do not have to be with custom ranking predicates.

rankinglist()

```
rankinglist() = list()
```

Members of a rankinglist() must be ranking()s.

relaxed_numeric_tuple()

```
relaxed_numeric_tuple() = numeric_tuple()
```

Relaxed numeric tuples are allowed to contain non-numeric elements, which are treated as zero for purposes of computation.

sanitizer()

```
sanitizer() = list() | filterfunction()
```

Sanitizers are used by sanitize_tokens/2 for input sanitization; they define what parts of an input list are valid, and the

remainder are removed. Sanitizers may either be a list of acceptable elements or a filter function.

seven_vector()

```
seven_vector() = vector()
```

A seven-vector always has seven elements, so this can be expressed as the alternation `{A::number(), B::number(), C::number(), D::number(), E::number(), F::number(), G::number()} | [A::number(), B::number(), C::number(), D::number(), E::number(), F::number(), G::number()]`.

sorted_keylist()

```
sorted_keylist() = keylist()
```

A sorted keylist is a keylist in the order provided by `lists:sort/1`. Because of erlang tuple ordering rules and the fact that keylist keys are unique, this means the list will be ordered by key.

stringlist()

```
stringlist() = list()
```

Every member of a stringlist() is a string().

three_or_seven_vector()

```
three_or_seven_vector() = three_vector() | seven_vector()
```

three_vector()

```
three_vector() = vector()
```

A three-vector always has three elements, so this can be expressed as the alternation `{A::number(), B::number(), C::number()} | [A::number(), B::number(), C::number()]`.

timestamp()

```
timestamp() = {Megaseconds::non_negative_integer(),
Seconds::non_negative_integer(),
MicroSeconds::non_negative_integer()}
```

typelabel()

```
typelabel() = [integer | float | list | tuple | binary
| bitstring | boolean | function | pid | port |
reference | atom | unknown]
```

Used by type_of(), this is just any single item from the list of erlang's primitive types, or the atom `unknown`.

unit_vector()

```
unit_vector() = vector()
```

The hypoteneuse of a unit vector is precisely one unit long. Unit vectors are also called normalized or magnitude-normalized vectors.

vector()

```
vector() = list() | tuple()
```

Every member element of a vector() is a `number()`.

vectorlist()

```
vectorlist() = list()
```

Every member element of a vectorlist() is a `vector()`.

weightedvalue()

```
weightedvalue() = {Value::any(), Weight::number()}
```

Used by functions like weighted_arithmetic_mean/1 and from_weighted/1, weightedvalue()s represent a value with an associated importance or "weight".

weightlist()

```
weightlist() = list()
```

All members of weightlists must be weightedvalue()s.

Function Index

absolute_difference/2	*Untested* *Stoch untested* Takes the absolute value of the difference between the two arguments.
abstract_attributes/1	*Untested* *Stoch untested*
abstract_function/2	*Untested* *Stoch untested*
abstract_functions/1	*Untested* *Stoch untested*
additive_factorial/1	*Untested* *Stoch untested*
adjust_counter_by/2	Adds to a counter's value; if the counter was not already defined, it will become the value in the `By` argument.
alarm_set/3	*Untested* *Stoch untested*
alarm_terminate/1	*Untested* *Stoch untested*
all_neighbor_pairs/1	*Untested* *Stoch untested*
all_neighbor_pairs/2	*Untested* *Stoch untested*
amean_vector_normal/1	*Untested* *Stoch untested* Returns the arithmetic mean of the elements of the unit vector for the vector provided.
arithmetic_mean/1	*Untested* *Stoch untested* Take the arithmetic mean (often called the average) of a list of numbers.
ascii_alphanum_list_subset/1	
bandwidth_calc/1	Equivalent to `bandwidth_calc(Data, all)`.
bandwidth_calc/2	*Untested* *Stoch untested* Calculates digital line bandwidth over timescales in converted units.
benchmark/1	
benchmark/2	
benchmark/3	
bin_to_hex_list/1	*Untested* *Stoch untested*

by_distance/2	*Untested* *Stoch untested*
by_distance_raw/2	*Untested* *Stoch untested*
byte_to_hex/1	*Untested* *Stoch untested* Convert a byte() into a hexstring().
caspers_jones_estimate/1	*Untested* *Stoch untested*
ceil/1	Equivalent to `ceiling(X)`.
ceiling/1	Returns the ceiling of a float.
central_moment/2	***BUGGY*** *Untested* *Stoch untested* Takes the Nth cetral moment of a list.
central_moments/1	*Untested* *Stoch untested*
central_moments/2	*Untested* *Stoch untested*
centroid/1	*Untested* *Stoch untested* Calculates the coordinate which represents the per-axis arithmetic mean of a set of points.
circ_within_origin_circ/2	*Untested* *Stoch untested*
circles_contact/2	*Untested* *Stoch untested*
circles_overlap/2	*Untested* *Stoch untested*
columnate/1	Equivalent to `columnate(List, 2, 3)`.
columnate/2	
columnated_rows/2	
columnated_text/2	
columns/2	
combinations/2	*Stoch untested* Provides a list of every unique combination of input terms, order-ignorant; contrast `permute/2`.
count_bits/1	Incomplete *Untested* *Stoch untested* Counts the number of bits turned on in a sufficiently sized unsigned two's compliment integer representation of `Num`.
count_of/2	*Untested* *Stoch untested* Counts the number of instances of Item in List.
counter_at/1	Checks a counter's value; if the counter was not already defined, it will report zero.
country_codes/0	
cross_product/2	Incomplete *Untested* *Stoch untested* Calculates the cross product of two vectors (Incomplete represented as

	three_vector()s - no support yet for seven).
cube/1	*Untested* *Stoch untested* Cubes the input; convenient in list comprehensions to prevent recalculation, and clear in the fashion of documentary functions.
dec_counter/1	Equivalent to adjust_counter_by(Name, -1).
dec_counter/2	Equivalent to adjust_counter_by(Name, -1 * By).
differences/1	*Untested* *Stoch untested*
distinct_neighbor_pairs/1	*Untested* *Stoch untested*
distinct_neighbor_pairs/2	*Untested* *Stoch untested*
dot_product/2	Incomplete *Untested* *Stoch untested* Calculates the dot product of two vectors (Incomplete represented as numeric lists; tuples not yet supported).
dstat/2	*Untested* *Stoch untested*
dstat_ex/2	*Untested* *Stoch untested*
elements/2	*Untested* *Stoch untested*.
elements/3	*Untested* *Stoch untested*
elements/4	*Untested* *Stoch untested*
ensure_started/1	
entrypoints/1	*Untested* *Stoch untested*
entrypoints/2	*Untested* *Stoch untested*
euclidean_distance/2	*Untested* *Stoch untested* Returns the distance between two coordinates in any N-space.
eval/1	***DANGEROUS*** *Untested* *Stoch untested* modified from http://www.trapexit.org/String_Eval.
eval/2	***DANGEROUS*** *Untested* *Stoch untested* from http://www.trapexit.org/String_Eval.
even_or_odd/1	Documentary convenience function (synonymous with parity/1) that returns the atoms `even` or `odd` for any integer.

every_flag_representation/1	*Untested* *Stoch untested* Returns every interpretation of the list as a set of boolean flags, including all-off and all-on.
every_member_representation/1	*Untested* *Stoch untested*
every_member_representation/2	*Untested* *Stoch untested* For a list of memberships, return every possible combination of one representative member from each list.
expand_labels/1	*Stoch untested* Expands a series of labels over lists to create a cartesian 2-ary tuple expansion.
expected_value/1	*Untested* *Stoch untested* Returns the expected value of infinite selection from a weighted numeric list.
explode/2	*Stoch untested*
explode/3	*Untested* *Stoch untested*
extrema/1	*Stoch untested* Returns the lowest and highest values in a list of one or more member in the form `{Lo,Hi}`.
factorial/1	*Untested* *Stoch untested*
factorize/1	*Untested* *Stoch untested* Generates a list of the factors of an integer.
file_to_binary_literal_as_string/1	
first_difference/1	*Untested* *Stoch untested*
first_pos/2	*Untested* *Stoch untested* Finds the 1-offset index of the first item in the list which passes the given predicate, or returns false if none pass.
first_pos/3	*Untested* *Stoch untested* Finds the 1-offset index of the first item in the list which passes the given predicate, or returns a default value if none is found.
first_row/1	
fk_readability/3	*Untested* *Stoch untested* Calculate the Flesch-Kincaid readability score of a set of text metrics.
fk_readability/4	*Untested* *Stoch untested* Calculate the Flesch-Kincaid readability score of a block of text.

flag_sets/1	*Stoch untested* Returns every interpretation of the list as a set of boolean flags, including all-off and all-on.
floor/1	1> sc:floor(0.5).
function_point_count/1	*Untested Stoch untested*
function_stats/1	*Untested Stoch untested*
gen_docs/0	(not testworthy) Generates library documentation using the paths appropriate for the author's PC; you almost certainly want `gen_docs/2` instead.
gen_docs/2	(not testworthy) Generates library documentation from and to the specified paths `WhereIsSrc` and `WhereToPutDocs` respectively.
geometric_mean/1	*Untested Stoch untested* Take the geometric mean of a list of numbers.
get_linked_processes/0	*Untested Stoch untested*.
gmean_vector_normal/1	*Untested Stoch untested* Returns the geometric mean of the elements of the unit vector for the vector provided.
grid_scatter/2	*Untested Stoch untested* Return a Count-length list of non-repeating coordinates in a grid of specified size; useful for feature generation.
halstead_complexity/4	*Untested Stoch untested*
halstead_complexity/5	*Untested Stoch untested*
harmonic_mean/1	*Untested Stoch untested* Take the harmonic mean of a list of numbers.
has_bit/2	Incomplete *Untested Stoch untested* Checks whether a given bit is on in a sufficiently sized unsigned two's compliment integer representation of `Num`.
hex_to_int/1	*Untested Stoch untested* Convert a hexstring() or hexchar() into its numeric value.
histograph/1	*Untested Stoch untested* Takes a histograph count of the items in the list.
hmac/3	*Untested Stoch untested* Shorthands for algorithms so you don't need to know block sizes.

hmac/4	*Semi-Untested* An implementation of RFC 2104, HMAC generic hash extension for any hash function and any key size.
hmac_md4/2	*Obsolete - legacy support only - do not use in new code Untested Stoch untested* HMAC wrapper built around MD4 as the core hash, frequently known as `hmac-md4` or `md4-hmac`.
hmac_md5/2	*Obsolete - legacy support only - do not use in new code Untested Stoch untested* HMAC wrapper built around MD5 as the core hash, frequently known as `hmac-md5` or `md5-hmac`.
hmac_sha1/2	Should be obsolete, but Erlang's standard library does not include SHA-2, and neither does `scutil` (yet) - *Untested Stoch untested* HMAC wrapper built around SHA1-160 (the only SHA-1) as the core hash, frequently known as `hmac-sha1`, `hmac-sha` or `sha1-hmac`.
hmean_vector_normal/1	*Untested Stoch untested* Returns the harmonic mean of the elements of the unit vector for the vector provided.
implode/2	*Stoch untested* Append strings with separating string inbetween - contrast `explode/2`.
in_range/3	
inc_counter/1	Equivalent to `adjust_counter_by(Name, 1)`.
inc_counter/2	Equivalent to `adjust_counter_by(Name, By)`.
index_of_first/2	*Stoch untested* Returns the index of the first instance of `Item` in the `List`, or `undefined` if `Item` is not present.
integer_to_radix_list/2	*Untested Stoch untested* Convert a number to a radix string using a radix list of your specification and any size.
io_list_to_hex_string/1	*Untested Stoch untested* Convert an io_list() to a hexstring().
is_between/3	

is_between/4	
is_numeric_char/1	
is_numeric_char/2	
is_numeric_string/1	*Untested Stoch untested*
is_numeric_string/2	*Untested Stoch untested*
is_postfix/2	*Untested Stoch untested*
is_repeated_list/1	
is_sorted_list/1	*Untested Stoch untested* Returns true if the list is sorted; false otherwise.
is_unique_list/1	*Untested Stoch untested* Returns true if the list is unique; false otherwise.
isolate_waveform/1	*Untested Stoch untested* Remove the baseline of a dataset, normalizing a waveform or other signal to its bottom peak.
kendall_correlation/1	*Untested Stoch untested* Compute the Kendall Tau Rank Correlation Coefficient of a list of coordinate tuples.
kendall_correlation/2	Equivalent to `kendall(lists:zip(List1, List2))`.
key_cluster/2	*Untested Stoch untested*
key_duplicate/1	Iterates a list of `{Count,Term}`, producing a list of `[Term,Term,...]`.
key_extrema/1	*Untested Stoch untested*.
key_extrema/2	*Untested Stoch untested*
key_group/2	*Untested Stoch untested*
key_group/3	*Untested Stoch untested*
key_max/1	*Untested Stoch untested*.
key_max/2	*Untested Stoch untested*
key_min/1	*Untested Stoch untested*.
key_min/2	*Untested Stoch untested*
kurtosis/1	***BUGGY*** *Untested Stoch untested*
labelled_fk_readability/1	*Untested Stoch untested* Provides mandated human-readable labels for Flesch-Kincaid readability calculations.

last_while_pos/2	*Untested* *Stoch untested* Returns the last element of the initial sequence where all items pass the predicate function.
last_while_pos/3	*Untested* *Stoch untested*
levenshtein/2	*Untested* *Stoch untested*, by fredrik svensson and adam lindberg, from http://www.merriampark.com/ lderlang.htm.
list_intersection/2	Equivalent to `list_intersection(List1, List2, unsorted)`.
list_intersection/3	*Stoch untested* Efficiently computes the intersection of two lists.
list_product/1	*Stoch untested* Takes the product of all numbers in the list.
list_to_number/1	*Untested* *Stoch untested* Converts a list into a number; integers will be returned if there is no mantissa in the list representation.
list_to_term/1	*Untested* *Stoch untested* Like binary_to_term, but not so much for binaries.
map_scanline/2	*Untested* *Stoch untested* Parses a string on all three newline types, discarding any empty lines; applies F as a functor to each line, and returns the tuple of the remainder and then a list of all results from the functor(s) issued.
map_scanline/3	*Untested* *Stoch untested* Third argument passes argument list as secondary argument to the functor; useful for passing ancillary state.
markhov_chain/2	Generates a markhov chain from a list of lists.
median/1	*Untested* *Stoch untested* Takes the median (central) value of a list.
median_absolute_deviation/1	*Untested* *Stoch untested* Calculate the median absolute deviation of a `numericlist()`.
member_sets/1	Equivalent to `member_sets(Memberships, no_absence)`.

member_sets/2	*Stoch untested* For a list of memberships, return every possible combination of one representative member from each list.
merge_settings/2	
mersenne_prime/1	*Untested Stoch untested*
mod/2	*Untested Stoch untested* Takes the modulus of an integer by another integer.
mode/1	*Untested Stoch untested* Takes the mode (most common) value(s) of a list, as a list.
module_abstract_representation/1	*Untested Stoch untested*
module_abstract_representation/2	*Untested Stoch untested*
module_atoms/1	*Untested Stoch untested*
module_attribute/1	*Untested Stoch untested* Look up all attributes of a given module.
module_attribute/2	***BUGGY*** *Untested Stoch untested* Look up an Erlang module attribute value by title.
module_feature/2	*Untested Stoch untested*
module_is_loaded/1	*Untested Stoch untested*
moment/2	*Untested Stoch untested* Takes the Nth moment of a list.
moments/1	*Untested Stoch untested*
moments/2	*Untested Stoch untested*
months_as_short_atoms/0	
multi_deck/2	Incomplete *Untested Stoch untested* Makes a number of instances of a deck, and applies a different back to each.
multi_do/3	Equivalent to `multi_do(C, M, F, [])`.
multi_do/4	*Untested Stoch untested* Take an iteration count, a module name, a function name and an argument list, and repeatedly apply the argument list to the module/function, count times.
naive_bayes_likelihood/4	*Untested Stoch untested* Calculates the contributing difference probability, feature likelihood and non-feature likelihood of an event by the naive Bayes likelihood method.
nearest_to/2	*Untested Stoch untested*
neighbors/2	

ngrams/1	
ngrams/2	
notebook_contains/2	
notebook_contents/1	
notebook_create/1	
notebook_destroy/1	
notebook_read/2	
notebook_remove/2	Removes an item from a notebook.
notebook_validate/1	
notebook_write/3	
now_str_utc24/0	
nth_difference/2	*Untested* *Stoch untested*
null_postpad_bin_to/2	
nybble_to_hex/1	*Untested* *Stoch untested* Convert a nybble() to a hexchar().
out_of_range/3	
paper_3d_basic_depth/4	*Untested* *Stoch untested*
paper_3d_render/1	*Untested* *Stoch untested*
paper_3d_render/2	*Untested* *Stoch untested*
paper_3d_render/3	*Untested* *Stoch untested*
paper_3d_render/4	*Untested* *Stoch untested*
parity/1	Documentary convenience function (synonymous with even_or_odd/1) that returns the atoms `even` or `odd` for any integer.
partition_by_residue/2	*Untested* *Stoch untested*
pearson_correlation/1	*Untested* *Stoch untested* Compute the Pearson Correlation Coefficient of a list of coordinate tuples.
pearson_correlation/2	Equivalent to `pearson(lists:zip(List1, List2))`.
permute/1	Equivalent to `permute(List, length(List))`.
permute/2	*Stoch untested* Calculate either the full or the depth-limited permutations of a list, order sensitive; contrast `combinations/2`.
power_set/1	*Untested* *Stoch untested*

probability_all/1	*Untested Stoch untested* Calculates the likelihood that all items in a list of probabilities expressed on the real interval will occur.
probability_any/1	*Incomplete Todo Comeback Untested Stoch untested*
qsp_average/2	Incomplete *Untested Stoch untested* Takes the quadratic scalar product average of a vector `W` and a list of vectors `X`.
rand/1	*Untested Stoch untested* Returns a pseudorandom integer on the range `[0 - (Range-1)]` inclusive.
random_from/1	Equivalent to `from(1, List, no_remainder)`.
random_from/2	Equivalent to `from(N, List, no_remainder)`.
random_from/3	*Untested Stoch untested* Take N non-repeating random elements from a list in undefined order.
random_from_weighted/1	*Untested Stoch untested* Take a random single item from a list with weighted probabilities.
random_unicode_char/0	
range_scale/1	*Untested Stoch untested* Get the scale of a same-sign numeric range.
receive_one/0	*Untested Stoch untested* Pop the front of the message queue and return it as `{item,X}`, or return nothing_there for empty queues; do not block.
record_member/2	*Untested Stoch untested* TODO: Needs Example Checks whether E is a member element of record R, analogous to `lists::member(E, L)`.
replace/3	
reset_counter/1	*Untested Stoch untested* Resets a counter's value to zero.
reverse_filter/2	*Untested Stoch untested*
reverse_map/2	*Untested Stoch untested*
reverse_map_filter/3	*Untested Stoch untested*

root_mean_square/1	*Untested* *Stoch untested* Calculates the root mean square of the values in the list.
root_sum_square/1	*Untested* *Stoch untested* Calculate the magnitude (also known as the root sum square) of a vector.
rotate_list/2	Rotates the front `Distance` elements of a list to the back, in order.
rotate_to_first/2	*Stoch untested* Rotates the list to the first instance of Item.
rotate_to_last/2	*Stoch untested* Rotates the list so that the first instance of Item becomes the last element in the list.
sanitize_filename/1	*Untested* *Stoch untested* Sanitize an arbitrary string to be appropriate for Windows and Unix filesystems, and URLs.
sanitize_tokens/2	*Stoch untested* Remove unacceptable elements from an input list, as defined by another list or a filter function.
second_difference/1	*Untested* *Stoch untested*
send_receive/2	*Untested* *Stoch untested* (Blocking) First send a message to an entity.
send_receive/3	*Untested* *Stoch untested* (Non-Blocking) First send a message to an entity.
send_receive_masked/3	*Untested* *Stoch untested* (Blocking) First send a message to an entity.
send_receive_masked/4	*Untested* *Stoch untested* (Non-Blocking) First send a message to an entity.
set_counter_value/2	*Untested* *Stoch untested* Sets a counter's value to a specific value.
shared_keys/1	*Stoch untested* Create sorted list X of 3-ary tuples {K,Ai,Bi} from sorted lists A, B of 2ary {K,Ai}/{K,Bi} tuples, where key K appears in both A and B.
shared_keys/2	*Stoch untested* Equivalent to `shared_keys/1`, but skips sorting the lists (and thus requires pre-sorted lists), which may save significant work repetition.
shared_keys/3	*Stoch untested* Equivalent to `shared_keys/2`, but skips sorting the lists

	(and thus requires pre-sorted lists), which may save significant work repetition.
show/1	*Untested Stoch untested*
shuffle/1	*Untested Stoch untested* Return a list with the original list's shallow members in a random order.
simple_ranking/1	*Untested Stoch untested* Returns a ranked ordering of the list without tie rankings.
skewness/1	*Untested Stoch untested*
spearman_correlation/1	*Untested Stoch untested* Compute the Spearman's Rank Correlation Coefficient of a list of coordinate tuples.
spearman_correlation/2	Equivalent to `spearman_correlation(lists:zip(List1, List2))`.
split_at/2	*Untested Stoch untested*
square/1	*Untested Stoch untested* Squares the input; convenient in list comprehensions to prevent recalculation, and clear in the fashion of documentary functions.
srand/0	*Untested Stoch untested (Called automatically)* Instantiates the random source, destroying a prior source if needed, and seeds the source with the clock, returning the seed used.
srand/3	*Untested Stoch untested (Called automatically)* Instantiates the random source, destroying a prior source if needed, and seeds the source with the three integer seed you provide, returning the seed used.
standard_card_backs/0	Incomplete *Untested Stoch untested* Returns the list of colors which are used, in order, as the standard back colors of a series of decks for `multi_deck/2`.
standard_card_backs/1	Incomplete *Untested Stoch untested* Returns the front of the list of colors which are used, in order, as the standard back colors of a series of decks for `multi_deck/2`.
start_register_if_not_running/2	*Untested Stoch untested*.

start_register_if_not_running/3	*Untested Stoch untested*.
start_register_if_not_running/4	*Untested Stoch untested*.
start_register_if_not_running/5	*Untested Stoch untested* Check whether a process is registered locally, and if not, spawn it with a give function and arguments.
starts_with/2	*Untested Stoch untested*
stretch_hash/3	*Untested Stoch untested* Stretches a hash with a list of salts.
svn_revision/1	*Untested Stoch untested* Scans a module for an attribute svn_revision, parses it in the format expected from the svn:keyword Revision, and returns the version number as an integer.
terminate_loop/0	*Untested Stoch untested*
test/0	(not testworthy) Runs the test suite in terse form.
test/1	(not testworthy) Runs the test suite in verbose form.
third_difference/1	*Untested Stoch untested*
tied_ordered_ranking/1	*Untested Stoch untested* Returns a tied ranked ordering of the list, ordered according to the input ordering rather than the sorted ordering.
tied_ranking/1	*Untested Stoch untested* Returns a ranked ordering of the list with tie rankings.
time_diff/2	Returns the difference, in seconds as a float, between two erlang timestamps as returned by `erlang:now()`.
to_lines/1	*Untested Stoch untested* Cuts a string according to any of the three newline conventions (even mixed), and discards empty strings.
to_list/1	*Untested Stoch untested*
triangle_index/1	*Untested Stoch untested*
triangle_index/2	*Untested Stoch untested*
tuple_member/2	*Untested Stoch untested* Checks whether E is a member element of tuple T, analogous to `lists::member(E, L)`.

tuple_sort/1	*Untested* *Stoch untested*
tuple_sum/1	*Untested* *Stoch untested* Returns the sum of the numeric elements of a tuple, treating non-numeric elements as zero.
type_of/1	*Untested* *Stoch untested* Fetch the type of the argument.
unfunnel/2	*Untested* *Stoch untested* Reverse a marketing funnel, to go from goal needed to input needed.
unfunnel/3	
union/1	*Untested* *Stoch untested*
union/2	*Untested* *Stoch untested*
union/3	*Untested* *Stoch untested*
union/4	*Untested* *Stoch untested*
unit_scale/1	
vector_magnitude/1	Equivalent to `root_sum_square(VX)`.
vector_normalize/1	Incomplete *Untested* *Stoch untested* Returns the magnitude of a vector.
walk_unique_pairings/2	*Untested* *Stoch untested*.
weighted_arithmetic_mean/1	*Untested* *Stoch untested* Take the weighted arithmetic mean of the input values.
zip_n/1	Equivalent to `zip_n(Ls, to_tuple)`.
zip_n/2	*Stoch untested* Computes a zip on any sized group of lists, rather than just two or three as offered by the lists module.
zipf_estimate_list/1	*Untested* *Stoch untested* Estimates the zipf baseline from each number in a numeric list.
zipf_nearness/1	*Untested* *Stoch untested* todo.
zipf_position_estimate/2	*Untested* *Stoch untested* Estimates the zipf baseline from a score and a rank position.

Function Details

absolute_difference/2

```
absolute_difference(A::number(), B::number()) ->
number()
```

Untested *Stoch untested* Takes the absolute value of the difference between the two arguments. Offered mostly to make dependant code clearer.

```
1> sc:absolute_difference(1.25, 1).
0.25

2> sc:absolute_difference(2,1).
1
3> sc:absolute_difference(1,2).
1
4> sc:absolute_difference(1,1).
0
5> sc:absolute_difference(100,35).
65
```

Introduced in: Version 504

abstract_attributes/1

```
abstract_attributes(Module) -> any()
```

Untested *Stoch untested*

Introduced in: version 532

abstract_function/2

```
abstract_function(Module, FName) -> any()
```

Untested *Stoch untested*

Introduced in: version 531

abstract_functions/1

```
abstract_functions(Module) -> any()
```

Untested *Stoch untested*

Introduced in: version 531

additive_factorial/1

```
additive_factorial(X) -> any()
```

Untested *Stoch untested*

Introduced in: Version 639

adjust_counter_by/2

```
adjust_counter_by(Name::any(), By::number()) ->
number()
```

Adds to a counter's value; if the counter was not already defined, it will become the value in the `By` argument.

```
1> sc:counter_at(hello).
0

2> sc:inc_counter(hello).
1

3> sc:adjust_by(hello, 3).
4
```

Introduced in: Version 680

alarm_set/3

```
alarm_set(Time, Lambda, Repeat) -> any()
```

Untested *Stoch untested*

Introduced in: Version 515

alarm_terminate/1

```
alarm_terminate(X1) -> any()
```

Untested *Stoch untested*

Introduced in: Version 515

all_neighbor_pairs/1

```
all_neighbor_pairs(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 536

all_neighbor_pairs/2

```
all_neighbor_pairs(List, WorkType) -> any()
```

Untested *Stoch untested*

Introduced in: Version 536

amean_vector_normal/1

```
amean_vector_normal(VX::numeric_list()) -> number()
```

Untested *Stoch untested* Returns the arithmetic mean of the elements of the unit vector for the vector provided.

Introduced in: Version 497

arithmetic_mean/1

```
arithmetic_mean(InputList::numericlist()) -> float()
```

Untested *Stoch untested* Take the arithmetic mean (often called the average) of a list of numbers.

```
1> sc:arithmetic_mean([1,2,3,4,5]).
3.0
```

Introduced in: Version 481

See also: amean_vector_normal/1, geometric_mean/1, harmonic_mean/1, weighted_arithmetic_mean/1.

ascii_alphanum_list_subset/1

```
ascii_alphanum_list_subset(List) -> any()
```

Introduced in: Version 651

bandwidth_calc/1

```
bandwidth_calc(Data) -> list_of_2ary_tuples()
```

Equivalent to bandwidth_calc(Data, all).

Introduced in: Version 478

bandwidth_calc/2

```
bandwidth_calc(BitsPerSecond::Data, Scale::bw_scale
| all) -> bw_rate() | [bw_rate()]
```

Untested Stoch untested Calculates digital line bandwidth over timescales in converted units.

```
1>
```

Also knows the shorthand notations `{X,meg}`, `{X,gig}` and `{X,t}` for input only in base-10 only.

```
5> sc:bandwidth_calc({10,meg}, {gigabits,day}).
{{gigabits,day},864.0}
```

Introduced in: Version 478

benchmark/1

```
benchmark(BareLambda) -> any()
```

Introduced in: Version 743

benchmark/2

```
benchmark(Fun, Args) -> any()
```

Introduced in: Version 743

benchmark/3

```
benchmark(Module, Func, Args) -> any()
```

bin_to_hex_list/1

```
bin_to_hex_list(Bin) -> any()
```

Untested *Stoch untested*

Introduced in: Version 640

by_distance/2

```
by_distance(Centers, Points) -> any()
```

Untested *Stoch untested*

Introduced in: Version 519

by_distance_raw/2

```
by_distance_raw(Centers, Points) -> any()
```

Untested *Stoch untested*

Introduced in: Version 518

byte_to_hex/1

```
byte_to_hex(TheByte::byte()) -> hexstring()
```

Untested *Stoch untested* Convert a byte() into a hexstring(). The hexstring() result will always be two characters (left padded with zero if necessary).

```
1> sc:byte_to_hex(7).
"07"

2> sc:byte_to_hex(255).
"ff"
```

Introduced in: Version 571

caspers_jones_estimate/1

```
caspers_jones_estimate(FunctionPoints) -> any()
```

Untested Stoch untested

Introduced in: Version 478

ceil/1

```
ceil(X) -> any()
```

Equivalent to `ceiling(X)`.

Introduced in: Version 510

ceiling/1

```
ceiling(X) -> any()
```

Returns the ceiling of a float.

```
1> 339> sc:ceiling(2.5).
3

340> sc:ceiling(3.0).
3

341> sc:ceiling(-2.5).
-2

342> sc:ceiling(-3.0).
-3

343> sc:ceiling(0).
0

344> sc:ceiling(0.0).
0
```

Introduced in: Version 510

central_moment/2

```
central_moment(List::list(), N::integer()) ->
float()
```

BUGGY *Untested Stoch untested* Takes the Nth cetral moment of a list. The Nth central moment of a list is the arithmetic mean of (the list items each minus the mean of the list, each taken to the Nth power). In a sense, this is the "normalized" moment. Fractional Ns are not defined. Not to be confused with `moment/2`. Thanks to Kraln and Chile for straightening me out on moments and central moments.

```
1> sc:central_moment([1,1,1], 2).
0.0

2> sc:central_moment([2,2,2], 2).
0.0

3> sc:central_moment([1,2,3], 2).
0.666666666666666

4> sc:central_moment([1,2,3], 3).
0.0
```

Thanks to Chile and Kraln for straightening me out on moments and central moments

Introduced in: Version 492

central_moments/1

```
central_moments(List) -> any()
```

Equivalent to `[central_moment(List, N) || N <- [2, 3, 4]]`.

Untested Stoch untested

Introduced in: Version 492

central_moments/2

```
central_moments(List, Moments) -> any()
```

Equivalent to `[central_moment(List, N) || N <- Moments]`.

Untested Stoch untested

Introduced in: Version 492

centroid/1

```
centroid(InputList::coord_list()) -> coord()
```

Untested *Stoch untested* Calculates the coordinate which represents the per-axis arithmetic mean of a set of points. Convenient in list comprehensions. To calculate the centroid of `[1,1]`, `[2,3]`, you gather the X coordinates `[1,2]`, then use their mean `1.5`; then do the same for the Y, `[1,3]` to `2`. The centroid would thus be `[1.5,2]`. You may pass any number of coordinates to this function, of any axis count, but they must all be the same axis count. The return value will be a coordinate with the same axis count. Negative and real values are fine; imaginary math is not implemented.

```
1> sc:centroid([[1]]).
[1.0]

2> sc:centroid([[1,1],[2,2]]).
[1.5,1.5]

3> sc:centroid([[1,1,1],[2,2,2],[3,3,3]]).
[2.0,2.0,2.0]

4> sc:centroid([[1,-1,1.0],[-2,-2,-2],[3,3,3],[4,4,4],[5,5,5]]).
[2.2,1.8,2.2]
```

Introduced in: Version 516

circ_within_origin_circ/2

```
circ_within_origin_circ(X1, X2) -> any()
```

Untested *Stoch untested*

circles_contact/2

```
circles_contact(X1, X2) -> any()
```

Untested *Stoch untested*

circles_overlap/2

```
circles_overlap(X1, X2) -> any()
```

Untested *Stoch untested*

columnate/1

```
columnate(List) -> any()
```

Equivalent to `columnate(List, 2, 3)`.

Introduced in: Version 734

columnate/2

```
columnate(List, Options) -> any()
```

Introduced in: Version 734

columnated_rows/2

```
columnated_rows(ColumnCount, List) -> any()
```

Introduced in: Version 731

columnated_text/2

```
columnated_text(List, Options) -> any()
```

Introduced in: Version 735

columns/2

```
columns(RowCount, List) -> any()
```

Introduced in: Version 730

combinations/2

```
combinations(OutputItemSize::positive_integer(),
Items::list()) -> list_of_lists()
```

Stoch untested Provides a list of every unique combination of input terms, order-ignorant; contrast `permute/2`. Permutations are all unique combinations of a set of tokens; the 2-permutations of `[a,b,c]` for example are `[a,b]`, `[a,c]` and

`[b,c]`. Note the absence of other orderings, such as `[b,a]`, which are provided by `permute/2`. Combinations are taken of a smaller count of tokens than the main set. Combinations are not ordered, but this implementation happens to provide answers in the same order as the input list. Mixed-type lists are safe; items are shallow evaluated, meaning that sublists within the list are treated as single elements, and will neither be rearranged nor will have elements selected from within them.

```
1> sc:combinations(2, [a,b,c,d]).
[ [a,b], [a,c], [a,d], [b,c], [b,d], [c,d] ]

2> sc:combinations(2, ["dave","kate","pat"]).
[ ["dave","kate"], ["dave","pat"], ["kate","pat"] ]

3> sc:combinations(2, [fast, strong, smart, lucky]).
[ [ fast,   strong ],
  [ fast,   smart  ],
  [ fast,   lucky  ],
  [ strong, smart  ],
  [ strong, lucky  ],
  [ smart,  lucky  ] ]
```

Thanks to Alisdair Sullivan for this implementation, which has been slightly but not significantly modified since receipt.

Introduced in: Version 473

count_bits/1

```
count_bits(Number::non_negative_integer()) ->
non_negative_integer()
```

Incomplete *Untested* *Stoch untested* Counts the number of bits turned on in a sufficiently sized unsigned two's compliment integer representation of `Num`.

```
1> sc:count_bits(5).
2
```

Introduced in: Version 727

count_of/2

```
count_of(Item::any(), List::list()) ->
non_negative_integer()
```

Untested *Stoch untested* Counts the number of instances of Item in List.

```
1> TestData = lists:duplicate(40, [healthy, nonsmoker] ) ++
              lists:duplicate(10, [healthy, smoker]    ) ++
              lists:duplicate(7,  [cancer,  nonsmoker] ) ++
              lists:duplicate(3,  [cancer,  smoker]    ).

[[healthy,nonsmoker], [healthy,nonsmoker], [healthy|...], [...]|...]

2> sc:count_of([healthy,smoker], TestData).
10

3> sc:count_of([healthy,nonsmoker], TestData).
40
```

Introduced in: Version 550

counter_at/1

```
counter_at(Name::any()) -> number()
```

Checks a counter's value; if the counter was not already defined, it will report zero.

```
1> sc:counter_at(hello).
0

2> sc:inc_counter(hello).
1

3> sc:inc_counter(hello).
2

4> sc:inc_counter(hello).
3

5> sc:counter_at(hello).
3
```

```
6> sc:reset_counter(hello).
0

7> sc:counter_at(hello).
0
```

Introduced in: Version 682

country_codes/0

```
country_codes() -> any()
```

Introduced in: Version 653

cross_product/2

```
cross_product(VX::three_vector(),
VY::three_vector()) -> three_vector()
```

Incomplete *Untested* *Stoch untested* Calculates the cross product of two vectors (Incomplete represented as `three_vector()`s - no support yet for seven).

```
1> sc:dot_product([1,1,1],[2,2,2]).
6

2> sc:dot_product([1,1,1],[3,3,3]).
9

3> sc:dot_product([-1,0,1],[3,3,3]).
0

4> sc:dot_product([-1,1,1],[3,3,3]).
3

5> sc:dot_product([0.5,1,2],[1,1,1]).
3.5
```

TODO: Implement seven-dimensional cross product

Introduced in: Version 80

cube/1

```
cube(Input::number()) -> number()
```

Untested *Stoch untested* Cubes the input; convenient in list comprehensions to prevent recalculation, and clear in the fashion of documentary functions.

```
1> sc:cube(2).
8

2> sc:cube(2.5).
6.25
```

Introduced in: Version 508

dec_counter/1

```
dec_counter(Name) -> any()
```

Equivalent to `adjust_counter_by(Name, -1)`.

Introduced in: Version 681

dec_counter/2

```
dec_counter(Name, By) -> any()
```

Equivalent to `adjust_counter_by(Name, -1 * By)`.

Introduced in: Version 681

differences/1

```
differences(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

distinct_neighbor_pairs/1

```
distinct_neighbor_pairs(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 535

distinct_neighbor_pairs/2

```
distinct_neighbor_pairs(List, MakeType) -> any()
```

Untested *Stoch untested*

Introduced in: Version 535

dot_product/2

```
dot_product(VX::numeric_list(), VY::numeric_list())
-> number()
```

Incomplete *Untested* *Stoch untested* Calculates the dot product of two vectors (Incomplete represented as numeric lists; tuples not yet supported).

```
1> sc:dot_product([1,1,1],[2,2,2]).
6

2> sc:dot_product([1,1,1],[3,3,3]).
9

3> sc:dot_product([-1,0,1],[3,3,3]).
0

4> sc:dot_product([-1,1,1],[3,3,3]).
3

5> sc:dot_product([0.5,1,2],[1,1,1]).
3.5
```

TODO: The tuple variation of vectors has not yet been implemented in this function.

Introduced in: Version 80

dstat/2

```
dstat(NumericList, PopulationOrSample) -> any()
```

Untested *Stoch untested*

Introduced in: Version 487

dstat_ex/2

```
dstat_ex(NumericList, PopulationOrSample) -> any()
```

Untested *Stoch untested*

Introduced in: Version 487

elements/2

```
elements(Config, Requested) -> any()
```

Untested *Stoch untested*

Introduced in: Version 546

elements/3

```
elements(Config, Requested, KeyIdx) -> any()
```

Untested *Stoch untested*

elements/4

```
elements(Config, Requested, KeyIdx, X4) -> any()
```

Untested *Stoch untested*

ensure_started/1

```
ensure_started(App) -> any()
```

Introduced in: Version 758

entrypoints/1

```
entrypoints(Module) -> any()
```

Untested *Stoch untested*

Introduced in: Version 530

entrypoints/2

```
entrypoints(Module, FName) -> any()
```

Untested *Stoch untested*

Introduced in: Version 530

euclidean_distance/2

```
euclidean_distance(Coordinate1::coord(),
Coordinate2::coord()) -> number()
```

Untested *Stoch untested* Returns the distance between two coordinates in any N-space. In two dimensions, this is known as the Pythagorean theorem. The coordinates may be of any positive integer dimensionality (2d, 3d, but no -1d or 2.5d), but both coordinates must be of the same dimensionality. The coordinates may have real-valued or negative components, but imaginary math is not implemented. This function tolerates tuple coordinates by converting them to lists; list coordinates are thus slightly faster.

```
1> sc:distance([0,0],[1,1]).
1.4142135623730951

2> sc:distance({0,0},[-1,1.0]).
1.4142135623730951

3> sc:distance([0,0,0,0],[1,-1,1,-1]).
2.0
```

Introduced in: Version 575

eval/1

```
eval(S) -> any()
```

DANGEROUS *Untested* *Stoch untested* modified from http://www.trapexit.org/String_Eval

Introduced in: Version 556

eval/2

```
eval(S, Environ) -> any()
```

DANGEROUS *Untested* *Stoch untested* from http://www.trapexit.org/String_Eval

Introduced in: Version 556

even_or_odd/1

```
even_or_odd(Num::integer()) -> even | odd
```

Documentary convenience function (synonymous with parity/1) that returns the atoms `even` or `odd` for any integer.

```
1> sc:even_or_odd(3).
odd
```

Introduced in: Version 489

every_flag_representation/1

```
every_flag_representation(Flags::list()) ->
list_of_lists()
```

Untested *Stoch untested* Returns every interpretation of the list as a set of boolean flags, including all-off and all-on.

```
1> sc:every_flag_representation([1,2,3,4]).
[ [], [4], [3], [3,4], [2], [2,4], [2,3], [2,3,4], [1], [1,4], [1,3]

2> length(sc:every_flag_representation(lists:seq(1,16))).
65536

3> SourceOfPowers = sc:every_flag_representation([magic,technology,e
[[],                                % Batman
 [alien],                           % Superman
 [evil],                            % Darkseid
 [evil,alien],                      % Sinestro
 [technology],                      % Mister Terrific (Michael Holt)
 [technology,alien],                % The Blue Beetle
 [technology,evil],                 % The OMACs
 [technology,evil,alien],           % Braniac
```

```
    [magic],                          % Shazam
    [magic,alien],                    % Green Lantern (Alan Scott)
    [magic,evil],                     % Lucifer Morningstar
    [magic,evil,alien],               % pre-crisis Star Sapphire
    [magic,technology],               % Alexander Luthor Jr.
    [magic,technology,alien],         % Mister Miracle
    [magic,technology,evil],          % pre-crisis Sinestro
    [magic,technology,evil,alien]]    % Granny Goodness
```

Introduced in: Version 552

every_member_representation/1

```
every_member_representation(Memberships) -> any()
```

Equivalent to every_member_representation(Memberships, no_absence).

Untested *Stoch untested*

every_member_representation/2

```
every_member_representation(Memberships::list_of_lists(),
AllowAbsence::atom()) -> list_of_lists()
```

Untested *Stoch untested* For a list of memberships, return every possible combination of one representative member from each list. The parameter `AllowAbsence` controls whether memberships may be unrepresented; if unrepresented memberships are possible, then one possible representation becomes the empty list.

```
1> sc:every_member_representation([ [a,b],[1,2,3],[i,ii,iii] ], no_a
[[a,1,i], [a,1,ii], [a,1,iii], [a,2,i], [a,2,ii], [a,2,iii], [a,3,i]

2> sc:every_member_representation([ [a,b],[1,2],[i,ii] ], allow_abse
[ [], [i], [ii], [1], [1,i], [1,ii], [2], [2,i], [2,ii], [a], [a,i],

3> Format = fun(Person, Place, Weapon) -> "It was " ++ Person ++ " i
#Fun<erl_eval.18.105910772>

4> { People, Places, Weapons } = { ["Col. Mustard", "Mr. Green"], ["
{["Col. Mustard","Mr. Green"],
 ["the billiards room","the kitchen"],
```

```
 ["a lead pipe","a knife","a gun"]}

5> Places.
["the billiards room","the kitchen"]

6> Format("Mrs. Scarlett", "the observatory", "a noose").
"It was Mrs. Scarlett in the the observatory with the a noose!"

7> EveryClueOutcome = [ Format(ThisPerson, ThisPlace, ThisWeapon) ||
["It was Col. Mustard in the the billiards room with the a lead pipe
 "It was Col. Mustard in the the billiards room with the a knife!",
 "It was Col. Mustard in the the billiards room with the a gun!",
 "It was Col. Mustard in the the kitchen with the a lead pipe!",
 "It was Col. Mustard in the the kitchen with the a knife!",
 "It was Col. Mustard in the the kitchen with the a gun!",
 "It was Mr. Green in the the billiards room with the a lead pipe!",
 "It was Mr. Green in the the billiards room with the a knife!",
 "It was Mr. Green in the the billiards room with the a gun!",
 "It was Mr. Green in the the kitchen with the a lead pipe!",
 "It was Mr. Green in the the kitchen with the a knife!",
 "It was Mr. Green in the the kitchen with the a gun!"]
```

Introduced in: Version 551

expand_labels/1

```
expand_labels(List::[{Label::any(), List::list()}])
-> list_of_lists()
```

Stoch untested Expands a series of labels over lists to create a cartesian 2-ary tuple expansion.

```
1> sc:expand_labels([{villain,[lex_luthor,sinistar,gargamel]}]).
[{villain,lex_luthor},{villain,sinistar},{villain,gargamel}]

2> sc:expand_labels([ {hero,[superman,tinyship,papa_smurf]}, {villai
[{hero,superman},
 {hero,tinyship},
 {hero,papa_smurf},
 {villain,lex_luthor},
 {villain,sinistar},
 {villain,gargamel}]
```

Introduced in: Version 474

expected_value/1

```
expected_value(List::mixed_weight_list()) ->
number()
```

Untested *Stoch untested* Returns the expected value of infinite selection from a weighted numeric list.

```
1> sc:expected_value([1,2,3,4,5,6]).
3.50000
```

Wolfram Alpha confirms

```
   2> sc:expected_value([ {1,5}, {10,1} ]).
2.5

3> sc:expected_value([ {-1,37}, {35,1} ]).
-5.26316e-2
```

Introduced in: Version 502

explode/2

```
explode(Separator, Term) -> any()
```

Stoch untested

Introduced in: Version 622

explode/3

```
explode(Separator, Term, Max) -> any()
```

Untested *Stoch untested*

Introduced in: Version 622

extrema/1

```
extrema(List::non_empty_list()) -> {Low::any(),
Hi::any()}
```

Stoch untested Returns the lowest and highest values in a list of one or more member in the form `{Lo,Hi}`. Undefined over the empty list. Mixed-type safe; sorts according to type order rules.

```
1> sc:extrema([1,2,3,4]).
{1,4}

2> sc:extrema([1,2,3,a,b,c]).
{1,c}
```

3> sc:extrema([]). ** exception error: no function clause matching sc:extrema([])'''

Introduced in: Version 460

factorial/1

```
factorial(X) -> any()
```

Untested *Stoch untested*

Introduced in: Version 509

factorize/1

```
factorize(N) -> any()
```

Untested *Stoch untested* Generates a list of the factors of an integer. Not an awesome implementation. Thanks for noticing that I was repeating sqrt unnecessarily, Forest.

file_to_binary_literal_as_string/1

```
file_to_binary_literal_as_string(PathAndFilename) ->
any()
```

Introduced in: Version 661

first_difference/1

```
first_difference(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

first_pos/2

```
first_pos(List, Predicate) -> any()
```

Untested *Stoch untested* Finds the 1-offset index of the first item in the list which passes the given predicate, or returns false if none pass.

```
1> sc_lists:first_pos([a,b,c,d,2,f],fun erlang:is_integer/1).
5

2> sc_lists:first_pos([a,b,c,d,e,f],fun erlang:is_integer/1).
false
```

Introduced in: Version 539

first_pos/3

```
first_pos(List, Predicate, Default) -> any()
```

Untested *Stoch untested* Finds the 1-offset index of the first item in the list which passes the given predicate, or returns a default value if none is found. See `first_pos/2` for details.

first_row/1

```
first_row(Columns) -> any()
```

Introduced in: Version 733

fk_readability/3

```
fk_readability(Words, Sentences, Syllables) -> any()
```

Untested *Stoch untested* Calculate the Flesch-Kincaid readability score of a set of text metrics. See Wikipedia and readabilityformulas.com.

Introduced in: Version 706

fk_readability/4

```
fk_readability(Data, WordCounter, SentenceCounter,
SyllableCounter) -> any()
```

Untested *Stoch untested* Calculate the Flesch-Kincaid readability score of a block of text. Also takes three lambdas to do text parsing. @see http://en.wikipedia.org/wiki/Flesch-Kincaid_Readability_Test

Introduced in: Version 708

flag_sets/1

```
flag_sets(Flags::list()) -> list of lists()
```

Stoch untested Returns every interpretation of the list as a set of boolean flags, including all-off and all-on.

```
1> sc:flag_sets([1,2,3,4]).
[ [], [4], [3], [3,4], [2], [2,4], [2,3], [2,3,4], [1], [1,4], [1,3]

2> length(sc:flag_sets(lists:seq(1,16))).
65536

3> sc:flag_sets([]).
[ [] ]

4> SourceOfPowers = sc:flag_sets([magic,technology,evil,alien]).
[[],                                % Batman
 [alien],                           % Superman
 [evil],                            % Darkseid
 [evil,alien],                      % Sinestro
 [technology],                      % Mister Terrific (Michael Holt)
 [technology,alien],                % The Blue Beetle
 [technology,evil],                 % The OMACs
 [technology,evil,alien],           % Braniac
 [magic],                           % Shazam
 [magic,alien],                     % Green Lantern (Alan Scott)
 [magic,evil],                      % Lucifer Morningstar
 [magic,evil,alien],                % pre-crisis Star Sapphire
 [magic,technology],                % Alexander Luthor Jr.
 [magic,technology,alien],          % Mister Miracle
```

```
    [magic,technology,evil],          % pre-crisis Sinestro
    [magic,technology,evil,alien]]    % Granny Goodness
```

Introduced in: Version 465

floor/1

```
floor(X) -> any()
```

1> sc:floor(0.5). 0

2> sc:floor(0). 0

3> sc:floor(0.0). 0

4> sc:floor(1.0). 1

5> sc:floor(-1.0). -1

6> sc:floor(-1.5). -2

7> sc:floor(-1). -1

8> sc:floor(1). 1'''

Introduced in: Version 511

function_point_count/1

```
function_point_count(Module) -> any()
```

Untested *Stoch untested*

Introduced in: Version 525

function_stats/1

```
function_stats(Module) -> any()
```

Untested *Stoch untested*

Introduced in: Version 524

gen_docs/0

```
gen_docs() -> ok | {'EXIT', any()}
```

Equivalent to gen_docs("/projects/scutil/erl/src", "/projects/scutil/erl/src/docs").

(not testworthy) Generates library documentation using the paths appropriate for the author's PC; you almost certainly want gen_docs/2 instead.

```
1> sc:gen_docs().
ok
```

Introduced in: 458

gen_docs/2

```
gen_docs(WhereIsSrc::string(),
WhereToPutDocs::string()) -> ok | {'EXIT', any()}
```

(not testworthy) Generates library documentation from and to the specified paths `WhereIsSrc` and `WhereToPutDocs` respectively. Do not use trailing slashes. Windows paths are okay; remember to double your backslashes, as backslashes in strings need to be quoted.

```
1> sc:gen_docs("/projects/scutil", "/projects/scutil/erl/src/docs").
ok
```

Introduced in: 458

geometric_mean/1

```
geometric_mean(InputList::numericlist()) -> float()
```

Untested *Stoch untested* Take the geometric mean of a list of numbers.

```
1> sc:geometric_mean([1,2,3,4,5]).
2.6051710846973517
```

WolframAlpha Confirms

The geometric mean is not defined for lists including 0.

The naive approach `geometric_mean(List) -> math:pow(sc:list_product(List), 1/length(List))` is not used because it accumulates error very quickly, and is as such unsuited to huge lists. This is the same as the expected function nth-root(prod, 1/n), but calculated differently for machine reasons.'''

Thanks to Forest (anonymous by choice) for help resolving 0-correctness.

Introduced in: Version 482

See also: arithmetic_mean/1, gmean_vector_normal/1, harmonic_mean/1.

get_linked_processes/0

```
get_linked_processes() -> any()
```

Untested *Stoch untested*

Introduced in: Version 617

gmean_vector_normal/1

```
gmean_vector_normal(VX::numeric_list()) -> number()
```

Untested *Stoch untested* Returns the geometric mean of the elements of the unit vector for the vector provided.

Introduced in: Version 498

grid_scatter/2

```
grid_scatter(Count::integer(), Size::gridsize()) ->
coordlist()
```

Untested *Stoch untested* Return a Count-length list of non-repeating coordinates in a grid of specified size; useful for feature generation.

Introduced in: Version 599

halstead_complexity/4

```
halstead_complexity(DistinctOperators,
DistinctOperands, TotalOperators, TotalOperands) ->
any()
```

Untested *Stoch untested*

Introduced in: Version 564

halstead_complexity/5

```
halstead_complexity(DistinctOperators,
DistinctOperands, TotalOperators, TotalOperands, X5)
-> any()
```

Untested *Stoch untested*

Introduced in: Version 564

harmonic_mean/1

```
harmonic_mean(InputList::numericlist()) -> float()
```

Untested *Stoch untested* Take the harmonic mean of a list of numbers.

```
1> sc:harmonic_mean([1,2,3,4,5]).
2.18978102189781
```

WolframAlpha Confirms

The harmonic mean is not defined for lists including 0.

Thanks to Forest (anonymous by choice) for help resolving 0-correctness.

Introduced in: Version 483

See also: arithmetic_mean/1, geometric_mean/1, hmean_vector_normal/1.

has_bit/2

```
has_bit(Number::non_negative_integer(),
Bit::non_negative_integer()) -> true | false
```

Incomplete *Untested* *Stoch untested* Checks whether a given bit is on in a sufficiently sized unsigned two's compliment integer representation of `Num`.

```
1> sc:has_bit(5,0).
true

2> scutil:has_bit(5,1).
false
```

Introduced in: Version 727

hex_to_int/1

```
hex_to_int(HexChar::hexstring() | hexchar()) ->
integer()
```

Untested *Stoch untested* Convert a hexstring() or hexchar() into its numeric value.

```
1> sc:hex_to_int("c0ffEE").
12648430

2> sc:hex_to_int($e).
14

3> sc:hex_to_int("100").
256
```

Introduced in: Version 572

histograph/1

```
histograph(List::list()) -> weightlist()
```

Untested *Stoch untested* Takes a histograph count of the items in the list. Mixed type lists are safe. Input lists do not need to be sorted. The histograph is shallow - that is, the histograph of `[`

[1,2], [1,2], [2,2]] is [{[1,2],2}, {[2,2],1}], not [{1,2}, {2,4}].

```
1> sc:histograph([1,2,a,2,b,1,b,1,b,2,a,2,2,1]).
[{1,4},{2,5},{a,2},{b,3}]

2> sc:histograph([ sc:rand(10) || X <- lists:seq(1,100000) ]).
[{0,10044}, {1,9892}, {2,10009}, {3,10016}, {4,10050}, {5,10113}, {6

3> ChessBoard = [ rook,  knight, bishop, king,  queen, bishop, knigh
                  pawn,  pawn,   pawn,   pawn,  pawn,  pawn,   pawn,
                  empty, empty,  empty,  empty, empty, empty,  empty
                  empty, empty,  empty,  empty, empty, empty,  empty
                  empty, empty,  empty,  empty, empty, empty,  empty
                  empty, empty,  empty,  empty, empty, empty,  empty
                  pawn,  pawn,   pawn,   pawn,  pawn,  pawn,   pawn,
                  rook,  knight, bishop, king,  queen, bishop, knigh
[rook,knight,bishop,king,queen,bishop,knight,rook,pawn,pawn,
 pawn,pawn,pawn,pawn,pawn,pawn,empty,empty,empty,empty,empty,
 empty,empty,empty,empty,empty,empty,empty,empty|...]

4> sc:histograph(ChessBoard).
[ { bishop, 4  },
  { empty,  32 },
  { king,   2  },
  { knight, 4  },
  { pawn,   16 },
  { queen,  2  },
  { rook,   4  } ]
```

Introduced in: Version 496

hmac/3

```
hmac(X1, Key, Data) -> any()
```

Untested *Stoch untested* Shorthands for algorithms so you don't need to know block sizes.

Introduced in: Version 646

hmac/4

```
hmac(HashFun, Key, Data, BlockSize) -> any()
```

Semi-Untested An implementation of RFC 2104, HMAC generic hash extension for any hash function and any key size.

The reason this exists is to bring HMAC access to any hashing algorithm, as was the RFC's purpose. There are HMAC functions in Erlang's `crypto:` module, but they are bound to specific hashers which are beginning to show their age, and they fix block size.

The block size should be at most the block size of the hashing algorithm, but may be reduced (to the detriment of the safety of the result.) Ideally, the block size should be the same as the hashing algorithm's block size, but many systems use variously truncated block sizes, so we support them all. Jerks.

This implementation was

The key should be at least as long as the hash residue. For example, if you're using MD5, which has 16-byte residues, the key should be at least 16 bytes. As the specification requires, if the key is larger than the algorithm selected block size, the key will be hashed then null post-padded to the algorithm selected block size.

```
1> sc:bin_to_hex_list(sc:hmac(fun erlang:md5/1, "hello", "world", 64
"0e2564b7e100f034341ea477c23f283b"

2> sc:bin_to_hex_list(crypto:md5_mac("hello","world")).
"0e2564b7e100f034341ea477c23f283b"

C:\Users\John>php
<?php /* php api is reversed of erlang's */
  echo hash_hmac('md5', 'world','hello');
?> ^Z
0e2564b7e100f034341ea477c23f283b

% Also, one of the RFC test sets

3> sc:bin_to_hex_list(sc:hmac(fun erlang:md5/1, "Jefe", "what do ya
"750c783e6ab0b503eaa86e310a5db738"

4> sc:bin_to_hex_list(crypto:md5_mac("Jefe", "what do ya want for no
"750c783e6ab0b503eaa86e310a5db738"

C:\Users\John>php
```

```
<?php echo hash_hmac('md5', 'what do ya want for nothing?', 'Jefe');
750c783e6ab0b503eaa86e310a5db738
```

Introduced in: Version 645

hmac_md4/2

```
hmac_md4(Key, Data) -> any()
```

Obsolete - legacy support only - do not use in new code Untested Stoch untested HMAC wrapper built around MD4 as the core hash, frequently known as `hmac-md4` or `md4-hmac`.

Introduced in: Version 647

hmac_md5/2

```
hmac_md5(Key, Data) -> any()
```

Obsolete - legacy support only - do not use in new code Untested Stoch untested HMAC wrapper built around MD5 as the core hash, frequently known as `hmac-md5` or `md5-hmac`.

Introduced in: Version 647

hmac_sha1/2

```
hmac_sha1(Key, Data) -> any()
```

Should be obsolete, but Erlang's standard library does not include SHA-2, and neither does `scutil` (yet) - *Untested Stoch untested* HMAC wrapper built around SHA1-160 (the only SHA-1) as the core hash, frequently known as `hmac-sha1`, `hmac-sha` or `sha1-hmac`.

Introduced in: Version 647

hmean_vector_normal/1

```
hmean_vector_normal(VX::numeric_list()) -> number()
```

Untested Stoch untested Returns the harmonic mean of the elements of the unit vector for the vector provided.

Introduced in: Version 499

implode/2

```
implode(Separator, Data) -> any()
```

Stoch untested Append strings with separating string inbetween - contrast explode/2.

```
1> sc:implode(",", ["a", "b", "c"]).
"a,b,c"

2> sc:implode(",", ["ab", "cd", "ef"]).
"ab,cd,ef"

3> sc:implode(",", ["", "", ""]).
",,"

4> sc:implode("-wop ", ["do", "do", "do"]).
"do-wop do-wop do"

5> sc:implode("", ["", "", ""]).
[]
```

thanks for a much better implementation, etnt

Introduced in: Version 621

in_range/3

```
in_range(List, Lo, Hi) -> any()
```

Introduced in: Version 649

inc_counter/1

```
inc_counter(Name) -> any()
```

Equivalent to adjust_counter_by(Name, 1).

Introduced in: Version 681

inc_counter/2

```
inc_counter(Name, By) -> any()
```

Equivalent to adjust_counter_by(Name, By).

Introduced in: Version 681

index_of_first/2

```
index_of_first(Item, List) -> integer() | undefined
```

Stoch untested Returns the index of the first instance of `Item` in the `List`, or `undefined` if `Item` is not present.

```
1> sc:index_of_first(c, [a,b,c,d,e]).
3

2> sc:index_of_first(j, [a,b,c,d,e]).
undefined
```

Introduced in: Version 463

integer_to_radix_list/2

```
integer_to_radix_list(Number::number(),
Radix::tuple()) -> list()
```

Untested *Stoch untested* Convert a number to a radix string using a radix list of your specification and any size. When appropriate, prefer the system provided `erlang:integer_to_list/2`. Lists are accepted, but converted to tuples before use, so are inefficient.

```
1> sc_convert:integer_to_radix_list(1111, "0123456789abcdef").
"457"

2> sc_convert:integer_to_radix_list(1111, "0123456789").
"1111"

3> sc_convert:integer_to_radix_list(1234567890, "abcdefghij").
"bcdefghija"

4> sc_convert:integer_to_radix_list(12648430, {$0, $1, $2, $3, $4, $
```

```
"C0FFEE"

5> sc_convert:integer_to_radix_list(1234567890, [alpha, beta, gamma,
[beta,gamma,delta,epsilon,zeta,eta,theta,kappa,lambda,alpha]
```

Introduced in: Version 566

io_list_to_hex_string/1

```
io_list_to_hex_string(Input::io_list()) ->
hexstring()
```

Untested *Stoch untested* Convert an io_list() to a hexstring().

```
1> sc:io_list_to_hex_string("a").
"61"

2> sc:io_list_to_hex_string("a08n408nbqa").
"6130386e3430386e627161"
```

Introduced in: Version 569

is_between/3

```
is_between(X, A, B) -> any()
```

Introduced in: Version 685

is_between/4

```
is_between(X, A, B, X4) -> any()
```

Introduced in: Version 685

is_numeric_char/1

```
is_numeric_char(Ch) -> any()
```

Introduced in: Version 737

is_numeric_char/2

```
is_numeric_char(Ch, X2) -> any()
```

is_numeric_string/1

```
is_numeric_string(Str) -> any()
```

Untested *Stoch untested*

Introduced in: Version 625

is_numeric_string/2

```
is_numeric_string(Str, X2) -> any()
```

Untested *Stoch untested*

Introduced in: Version 625

is_postfix/2

```
is_postfix(Postfix, String) -> any()
```

Untested *Stoch untested*

Introduced in: Version 543

is_repeated_list/1

```
is_repeated_list(Rem) -> any()
```

Introduced in: Version 650

is_sorted_list/1

```
is_sorted_list(List::list()) -> true | false
```

Untested *Stoch untested* Returns true if the list is sorted; false otherwise. List sortedness is typesafe, and defined equivalently to how defined by the language and `lists:sort()`.

```
1> sc:is_sorted_list([1,2,3]).
true

2> sc:is_sorted_list([1,2,3,1]).
false
```

```
3> sc:is_sorted_list([1,2,3,false]).
true

4> sc:is_sorted_list([false,1,2,3]).
false
```

Introduced in: Version 514

is_unique_list/1

```
is_unique_list(List::list()) -> true | false
```

Untested *Stoch untested* Returns true if the list is unique; false otherwise. List uniqueness is defined as whether any member of the list compares equally to any other member; deep list inspection is not performed. Comparison is type-safe.

```
2> sc:is_unique_list([1,2,3]).
true

2> sc:is_unique_list([1,2,3,1]).
false

3> sc:is_unique_list([1,2,3,{1}]).
true

4> sc:is_unique_list([1,2,3,[1]]).
true

5> sc:is_unique_list([1,2,3,[1],[1]]).
false
```

Introduced in: Version 514

isolate_waveform/1

```
isolate_waveform(Waveform) -> any()
```

Untested *Stoch untested* Remove the baseline of a dataset, normalizing a waveform or other signal to its bottom peak.

Introduced in: Version 716

kendall_correlation/1

```
kendall_correlation(TupleList::coordlist()) -> {tau,
Correlation::number()}
```

Untested *Stoch untested* Compute the Kendall Tau Rank Correlation Coefficient of a list of coordinate tuples.

```
1> sc:kendall([ {1,1}, {2,2}, {3,3}, {4,4}, {5,5} ]).
{tau,1.0}

2> sc:kendall([ {1,5}, {2,4}, {3,3}, {4,2}, {5,1} ]).
{tau,-1.0}

3> sc:kendall([ {1,3}, {2,3}, {3,3}, {4,3}, {5,3} ]).
{tau,1.0}

4> sc:kendall([ {1,2}, {2,2.5}, {3,3}, {4,3.5}, {5,4} ]).
{tau,1.0}

5> sc:kendall([ {1,2}, {2,2.4}, {3,3}, {4,3.6}, {5,4} ]).
{tau,1.0}
```

Introduced in: Version 557

kendall_correlation/2

```
kendall_correlation(List1, List2) -> any()
```

Equivalent to kendall(lists:zip(List1, List2)).

key_cluster/2

```
key_cluster(Index, List) -> any()
```

Untested *Stoch untested*

key_duplicate/1

```
key_duplicate(CvList::list()) -> [any()]
```

Iterates a list of `{Count,Term}`, producing a list of `[Term,Term,...]`.

```
1> sc:key_duplicate([ {3,bork} ]).
[bork,bork,bork]

2> sc:key_duplicate([ {3,sunday}, {2,monster}, {2,truck}, {1,'MADNES
[sunday,sunday,sunday,monster,monster,truck,truck,'MADNESS']
```

Introduced in: Version 462

key_extrema/1

```
key_extrema(List) -> any()
```

Equivalent to key_extrema(1, List).

Untested *Stoch untested*

Introduced in: Version 606

key_extrema/2

```
key_extrema(Pos, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 601

key_group/2

```
key_group(Pos, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 538

key_group/3

```
key_group(Pos, List, X3) -> any()
```

Untested *Stoch untested*

Introduced in: Version 538

key_max/1

```
key_max(List) -> any()
```

Equivalent to key_max(1, List).

Untested *Stoch untested*

Introduced in: Version 605

key_max/2

```
key_max(Pos, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 604

key_min/1

```
key_min(List) -> any()
```

Equivalent to key_min(1, List).

Untested *Stoch untested*

Introduced in: Version 607

key_min/2

```
key_min(Pos, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 607

kurtosis/1

```
kurtosis(List) -> any()
```

Equivalent to central_moment(List, 4).

BUGGY *Untested* *Stoch untested*

Wrong! todo comeback

Introduced in: Version 494

labelled_fk_readability/1

```
labelled_fk_readability(R) -> any()
```

Untested *Stoch untested* Provides mandated human-readable labels for Flesch-Kincaid readability calculations. @see http://en.wikipedia.org/wiki/Flesch-Kincaid_Readability_Test and http://www.readabilityformulas.com/graphics/fleschresults.gif .

Introduced in: Version 707

last_while_pos/2

```
last_while_pos(Predicate, List) -> any()
```

Untested *Stoch untested* Returns the last element of the initial sequence where all items pass the predicate function.

```
1> sc_lists:last_while_pos(fun erlang:is_atom/1, [a,b,c,d,2,f]).
4

2> sc_lists:last_while_pos(fun erlang:is_atom/1, [a,b,c,d,r,f]).
6

3> sc_lists:last_while_pos(fun erlang:is_atom/1, [1,a,b,c,d,r,f]).
false
```

Introduced in: Version 538

last_while_pos/3

```
last_while_pos(Predicate, List, Default) -> any()
```

Untested *Stoch untested*

levenshtein/2

```
levenshtein(Same, String) -> any()
```

Untested *Stoch untested*, by fredrik svensson and adam lindberg, from http://www.merriampark.com/lderlang.htm

Introduced in: Version 626

list_intersection/2

```
list_intersection(List1, List2) -> any()
```

Equivalent to list_intersection(List1, List2, unsorted).

list_intersection/3

```
list_intersection(List1::list(), List2::list(),
IsSorted::atom()) -> list()
```

Stoch untested Efficiently computes the intersection of two lists. The third parameter, which is optional and defaults to `unsorted`, is either the atom `sorted` or `unsorted`. If `sorted` is used, the function will sort both inputs before proceeding, as it requires sorted lists; as such, if you already know your lists to be sorted, passing `unsorted` will save some time. The return list will be reverse sorted.

```
1> sc:list_intersection([1,2,3,4,5,2,3,10,15,25,30,40,45,55],[1,3,5,
[55,40,30,15,5,3,1]

2> sc:list_intersection([1],[2]).
[]
```

Thanks to Ayrnieu for catching a defect in the initial implementation.

Introduced in: Version 471

list_product/1

```
list_product(A::numericlist()) -> number()
```

Stoch untested Takes the product of all numbers in the list. Offered mostly to make dependant code clearer.

```
1> sc:list_product([1,2,5.4]).
10.8
```

Introduced in: Version 476

list_to_number/1

```
list_to_number(X::list()) -> number()
```

Untested *Stoch untested* Converts a list into a number; integers will be returned if there is no mantissa in the list representation.

```
1> sc:list_to_number("2").
2

2> sc:list_to_number("2.0").
2.0

3> sc:list_to_number("2.1").
2.1
```

Introduced in: Version 574

list_to_term/1

```
list_to_term(List) -> any()
```

Untested *Stoch untested* Like binary_to_term, but not so much for binaries. Thanks, dizzyd (modified for error reporting)

Introduced in: Version 568

map_scanline/2

```
map_scanline(F, L) -> any()
```

Untested *Stoch untested* Parses a string on all three newline types, discarding any empty lines; applies F as a functor to each line, and returns the tuple of the remainder and then a list of all results from the functor(s) issued

thanks ayrnieu

Introduced in: Version 626

map_scanline/3

```
map_scanline(F, L, A) -> any()
```

Untested *Stoch untested* Third argument passes argument list as secondary argument to the functor; useful for passing ancillary state

Modified from map_scaline/2 by ayrnieu

Introduced in: Version 626

markhov_chain/2

```
markhov_chain(Depth, Sources) -> any()
```

Generates a markhov chain from a list of lists.

Introduced in: Version 703

median/1

```
median(List::numericlist()) -> number()
```

Untested *Stoch untested* Takes the median (central) value of a list. Sorts the input list, then finds and returns the middle value.

```
1> sc:median([1,2,999]).
2
```

Introduced in: Version 488

See also: arithmetic_mean/1, mode/1.

median_absolute_deviation/1

```
median_absolute_deviation(List::numericlist()) ->
number()
```

Untested *Stoch untested* Calculate the median absolute deviation of a numericlist().

```
1> sc:median_absolute_deviation([1,1,2,2,4,6,9]).
1
```

Introduced in: Version 501

member_sets/1

```
member_sets(Memberships) -> any()
```

Equivalent to member_sets(Memberships, no_absence).

member_sets/2

```
member_sets(Memberships::list_of_lists(),
AllowAbsence::atom()) -> list_of_lists()
```

Stoch untested For a list of memberships, return every possible combination of one representative member from each list. The parameter `AllowAbsence` controls whether memberships may be unrepresented; if unrepresented memberships are possible, then one possible representation becomes the empty list.

```
1> sc:member_sets([ [a,b],[1,2,3],[i,ii,iii] ], no_absence).
[ [a,1,i], [a,1,ii], [a,1,iii], [a,2,i], [a,2,ii], [a,2,iii], [a,3,i
  [b,1,i], [b,1,ii], [b,1,iii], [b,2,i], [b,2,ii], [b,2,iii], [b,3,i

2> sc:member_sets([ [a,b],[1,2],[i,ii] ], allow_absence).
[ [], [i], [ii], [1], [1,i], [1,ii], [2], [2,i], [2,ii], [a], [a,i],
  [a,1,ii], [a,2], [a,2,i], [a,2,ii], [b], [b,i], [b,ii], [b,1], [b,
  [b,2,i], [b,2,ii] ]

3> sc:member_sets([ [toast,pancakes], [sausage,bacon] ] ).
[[toast,sausage],
 [toast,bacon],
 [pancakes,sausage],
 [pancakes,bacon]]

4> sc:member_sets([ [toast,pancakes], [sausage,bacon] ], no_absence
[[toast,sausage],
 [toast,bacon],
 [pancakes,sausage],
 [pancakes,bacon]]

5> sc:member_sets([ [toast,pancakes], [sausage,bacon] ], allow_absen
 [[],
 [sausage],
 [bacon],
```

```
 [toast],
 [toast,sausage],
 [toast,bacon],
 [pancakes],
 [pancakes,sausage],
 [pancakes,bacon]]

6> Format = fun(Person, Place, Weapon) -> "It was " ++ Person ++ " i
#Fun<erl_eval.18.105910772>

7> [ Format(Pe,Pl,WW) || [Pe,Pl,WW] <- sc:member_sets( [ ["Col. Must
["It was Col. Mustard in the conservatory with the lead pipe!",
 "It was Col. Mustard in the hallway with the lead pipe!",
 "It was Col. Mustard in the kitchen with the lead pipe!",
 "It was Ms. Scarlett in the conservatory with the lead pipe!",
 "It was Ms. Scarlett in the hallway with the lead pipe!",
 "It was Ms. Scarlett in the kitchen with the lead pipe!"]
```

Introduced in: Version 466

merge_settings/2

```
merge_settings(S1, S2) -> any()
```

Introduced in: Version 576

mersenne_prime/1

```
mersenne_prime(Which) -> any()
```

Untested *Stoch untested*

Introduced in: Version 512

mod/2

```
mod(Base::integer(), Range::integer()) -> integer()
```

Untested *Stoch untested* Takes the modulus of an integer by another integer. Luckily, erlang calls what most languages refer to as modulus by its correct name, remainder (c's `%`, erlang's `rem`). Modulus is implemented incorrectly in nearly every language, because chip vendors implement remainder and the wrong name stuck. The difference is in how the operator reacts

to a negative `Base`: -10 modulo 3 is 2, whereas -10 rem 3 is -1. Remainder takes the residue of dividing the base by the lowest (nearest negative infinity) integer N adjacent the real valued divisor; modulo returns the highest, which is less CPU efficient but always provides an answer on [0..Range-1].

```
1> sc:mod(10,3).
1

2> [ sc:mod(X,4) || X <- lists:seq(-10,10) ].
[2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2]
```

Introduced in: Version 507

mode/1

```
mode(List::numericlist()) -> any()
```

Untested *Stoch untested* Takes the mode (most common) value(s) of a list, as a list. If there are more than one value tied for most common, all tied will be returned. This function is safe for mixed-type lists, and does not perform deep traversal (that is, the mode of `[ [2,2] ]` is `[2,2]`, not `2`).

```
sc:mode([1,2,1,3,1,4]).
[1]

2> sc:mode([ [1,2,3], [2,3,4], [3,4,5], [2,3,4] ]).
[[2,3,4]]

3> sc:mode([ a,b, 1, a,b, 2, a,b, 3 ]).
[a,b]
```

Introduced in: Version 497

See also: arithmetic_mean/1, median/1.

module_abstract_representation/1

```
module_abstract_representation(Module) -> any()
```

Untested *Stoch untested*

Introduced in: Version 553

module_abstract_representation/2

```
module_abstract_representation(Module, DoStrip) ->
any()
```

Untested *Stoch untested*

Introduced in: Version 553

module_atoms/1

```
module_atoms(Module) -> any()
```

Untested *Stoch untested*

Introduced in: Version 533

module_attribute/1

```
module_attribute(Module::atom()) -> AttributeList |
{error, no_such_module}
```

Untested *Stoch untested* Look up all attributes of a given module.

```
1> sc:module_attribute(sc).
[{author,"John Haugeland <stonecypher@gmail.com>"},
 {bugtracker,"http://crunchyd.com/forum/project.php?projectid=7"},
 {currentsource,"http://crunchyd.com/release/scutil.zip"},
 {description,"StoneCypher's utility library."},
 {library_requirements,[{testerl,16}]},
 {license,[{mit_license,"http://scutil.com/license.html"}]},
 {publicforum,"http://crunchyd.com/forum/scutil-discussion/"},
 {publicsvn,"svn://crunchyd.com/scutil/"},
 {svn_head,"$HeadURL$"},
 {svn_id,"$Id$"},
 {svn_revision,"$Revision$"},
 {testerl_export,[{[],scutil_testsuite}]},
 {vsn,[134633400955530778836494569152232539093]},
 {webpage,"http://scutil.com/"}]
```

Introduced in: Version 520

module_attribute/2

```
module_attribute(Module::atom(), Attribute::atom())
-> {value, {Attribute, Value}} | {error,
no_such_attribute} | {error, no_such_module}
```

BUGGY *Untested Stoch untested* Look up an Erlang module attribute value by title. Originally found at Mastering Erlang Part 3; subsequently cleaned up and given error reporting.

```
1> sc:module_attribute(scutil, author).
"John Haugeland <stonecypher@gmail.com>"

2> sc:module_attribute(scutil, license).
[{mit_license,"http://scutil.com/license.html"}]
```

Thanks to Alain O'Dea for pointing out defects in this routine regarding repeated module elements, and available improvements to the provided API. Mr. O'Dea's insightful advice will be implemented, but that time has not yet come.

Found at http://www.astahost.com/info.php/mastering-erlang-part-3-erlang-concurrent_t6632.html Reformatted for clarity, removed unnessecary framing list Added error handling behavior

Introduced in: Version 520

module_feature/2

```
module_feature(Module, Feature) -> any()
```

Untested Stoch untested

Introduced in: Version 521

module_is_loaded/1

```
module_is_loaded(ModuleName) -> any()
```

Untested Stoch untested

Introduced in: Version 666

moment/2

```
moment(List::list(), N::number()) -> float()
```

Untested *Stoch untested* Takes the Nth moment of a list. The Nth moment of a list is the arithmetic mean of the list items, each taken to the Nth power. Fractional Ns are well defined and have obscure uses, though most will only ever use this with integer values of N; this function is valid for both. Not to be confused with `central_moment/2`. Thanks to Kraln and Chile for straightening me out on moments and central moments.

```
1> sc:moment([1,1,1], 2).
1.0

2> sc:moment([2,2,2], 2).
4.0

3> sc:moment([1,2,3], 2).
4.666666666666667

4> sc:moment([1,2,3], 3).
12.0

5> sc:moment([1,2,3], 3.5).
19.693026767781483
```

Thanks to Chile and Kraln for straightening me out on moments and central moments

Introduced in: Version 491

moments/1

```
moments(List) -> any()
```

Equivalent to `[moment(List, N) || N <- [2, 3, 4]]`.

Untested *Stoch untested*

Introduced in: Version 491

moments/2

```
moments(List, Moments) -> any()
```

Equivalent to `[moment(List, N) || N <- Moments]`.

Untested *Stoch untested*

Introduced in: Version 491

months_as_short_atoms/0

```
months_as_short_atoms() -> any()
```

Introduced in: Version 665

multi_deck/2

```
multi_deck(Backs::positive_integer_or_list(),
DeckGenerator::function_or_list()) -> list()
```

Incomplete *Untested* *Stoch untested* Makes a number of instances of a deck, and applies a different back to each. The first parameter may be a `positive_integer()`, at which point the color sequence from `standard_backs/0` will be used; otherwise, a list may be used, which will be used as the card backs (there is no requirement regarding their type or uniqueness, only that they be presented as a list.) The second parameter may be a `function()`, which will be called to generate a list of cards, or a `list` of cards which will be used directly.

Introduced in: Version 729

multi_do/3

```
multi_do(C, Module, Func) -> any()
```

Equivalent to `multi_do(C, M, F, [])`.

Introduced in: Version 620

multi_do/4

```
multi_do(Count::integer(), Module::atom(),
Function::atom(), Args::list()) -> list()
```

Untested *Stoch untested* Take an iteration count, a module name, a function name and an argument list, and repeatedly apply the argument list to the module/function, count times. This is primarily useful with nondeterministic functions whose result might change despite identical arguments, such as functions with random behavior; for example, this function is invoked to implement stochastic testing in TestErl.

```
1> sc:multi_do(10, scutil, rand, [100]).
[9,94,4,82,77,44,89,19,45,92]

2> sc:multi_do(10, scutil, rand, [10000]).
[2377,2559,1713,8489,4468,3261,3344,3751,380,2525]
```

Introduced in: Version 620

naive_bayes_likelihood/4

```
naive_bayes_likelihood(FeatureEvident::non_negative_integer(),
FeatureTotal::positive_integer(),
NonFeatureEvident::non_negative_integer(),
NonFeatureTotal::positive_integer()) ->
Result::list()
```

Untested *Stoch untested* Calculates the contributing difference probability, feature likelihood and non-feature likelihood of an event by the naive Bayes likelihood method.

Introduced in: Version 478

nearest_to/2

```
nearest_to(Centers, Point) -> any()
```

Untested *Stoch untested*

Introduced in: Version 517

neighbors/2

```
neighbors(X1, X2) -> any()
```

Introduced in: Version 683

ngrams/1

```
ngrams(List) -> any()
```

Introduced in: Version 652

ngrams/2

```
ngrams(List, X2) -> any()
```

Introduced in: Version 652

notebook_contains/2

```
notebook_contains(NotebookName, Key) -> any()
```

Introduced in: Version 752

notebook_contents/1

```
notebook_contents(NotebookName) -> any()
```

Introduced in: Version 757

notebook_create/1

```
notebook_create(NotebookName) -> any()
```

Introduced in: Version 747

notebook_destroy/1

```
notebook_destroy(NotebookName) -> any()
```

Introduced in: Version 748

notebook_read/2

```
notebook_read(NotebookName, Key) -> any()
```

Introduced in: Version 751

notebook_remove/2

```
notebook_remove(NotebookName, Key) -> any()
```

Removes an item from a notebook. Like most other notebook functions, if the referred notebook does not already exist, it will be created. notebook_write/3 for details.

```
1> sc:notebook_write("Test", "test", "test").
ok

2> sc:notebook_read("Test", "test").
{value,"test"}

3> sc:notebook_remove("Test", "test").
ok

4> sc:notebook_read("Test", "test").
undefined

5> sc:notebook_read("Not an existing notebook", "test").
ok
```

Introduced in: Version 753

notebook_validate/1

```
notebook_validate(NotebookName) -> any()
```

Introduced in: Version 749

notebook_write/3

```
notebook_write(NotebookName, Key, Value) -> any()
```

Introduced in: Version 750

now_str_utc24/0

```
now_str_utc24() -> any()
```

Introduced in: Version 744

nth_difference/2

```
nth_difference(N, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

null_postpad_bin_to/2

```
null_postpad_bin_to(Bin, ToLength) -> any()
```

Introduced in: Version 644

nybble_to_hex/1

```
nybble_to_hex(Nyb::nybble()) -> integer()
```

Untested *Stoch untested* Convert a nybble() to a hexchar().

```
1> sc:nybble_to_hex(7).
55

2> sc:nybble_to_hex(15).
102
```

Introduced in: Version 570

out_of_range/3

```
out_of_range(List, Lo, Hi) -> any()
```

Introduced in: Version 650

paper_3d_basic_depth/4

```
paper_3d_basic_depth(X, Z, SliderPos, DepthConstant)
-> any()
```

Untested *Stoch untested*

Introduced in: Version 633

paper_3d_render/1

```
paper_3d_render(Bitmap3dList) -> any()
```

Untested *Stoch untested*

Introduced in: Version 634

paper_3d_render/2

```
paper_3d_render(Bitmap3dList, DepthConstant) ->
any()
```

Untested *Stoch untested*

Introduced in: Version 634

paper_3d_render/3

```
paper_3d_render(Bitmap3dList, SliderPos,
DepthConstant) -> any()
```

Untested *Stoch untested*

Introduced in: Version 634

paper_3d_render/4

```
paper_3d_render(Bitmap3dList, SliderPos,
DepthConstant, DepthFun) -> any()
```

Untested *Stoch untested*

Introduced in: Version 634

parity/1

```
parity(Num::integer()) -> even | odd
```

Equivalent to `even_or_odd(Num)`.

Documentary convenience function (synonymous with even_or_odd/1) that returns the atoms `even` or `odd` for any integer.

```
1> sc:parity(3).
odd
```

Thanks for the suggestion, Forest.

Introduced in: Version 648

partition_by_residue/2

```
partition_by_residue(Data, Function) -> any()
```

Untested *Stoch untested*

Introduced in: Version 537

pearson_correlation/1

```
pearson_correlation(TupleList::coordlist()) -> {r,
Correlation::number()}
```

Untested *Stoch untested* Compute the Pearson Correlation Coefficient of a list of coordinate tuples.

```
1> sc_correlate:pearson([ {1,1}, {2,2}, {3,3}, {4,4}, {5,5} ]).
{r,1.0}

2> sc_correlate:pearson([ {1,5}, {2,4}, {3,3}, {4,2}, {5,1} ]).
{r,-1.0}

3> sc_correlate:pearson([ {1,3}, {2,3}, {3,3}, {4,3}, {5,3} ]).
{r,0.0}

4> sc_correlate:pearson([ {1,2}, {2,2.5}, {3,3}, {4,3.5}, {5,4} ]).
{r,1.0}

5> sc_correlate:pearson([ {1,2}, {2,2.4}, {3,3}, {4,3.6}, {5,4} ]).
{r,0.9970544855015818}
```

Introduced in: Version 559

pearson_correlation/2

```
pearson_correlation(List1, List2) -> any()
```

Equivalent to pearson(lists:zip(List1, List2)).

permute/1

```
permute(List) -> any()
```

Equivalent to permute(List, length(List)).

permute/2

```
permute(List::list(), Depth::positive_integer()) ->
list()
```

Stoch untested Calculate either the full or the depth-limited permutations of a list, order sensitive; contrast combinations/2. Permutations are all valid orderings of a set of tokens; the permutations of `[a,b]` for example are `[a,b]` and `[b,a]`. Depth limitation means the permutations of a smaller count of tokens from the main set; the 2-limited permutations of `[a,b,c]` for example are `[a,b]`, `[a,c]`, `[b,a]`, `[b,c]`, `[c,a]` and `[c,b]`. Permutations are not ordered. Mixed-type lists are safe; items are shallow evaluated, meaning that sublists within the list are treated as single elements, and will neither be rearranged nor will have elements selected from within them.

```
1> sc:permute(["dave","kate","pat"]).
[ { "pat",  "kate", "dave" },
  { "kate", "pat",  "dave" },
  { "pat",  "dave", "kate" },
  { "dave", "pat",  "kate" },
  { "kate", "dave", "pat"  },
  { "dave", "kate", "pat"  } ]

2> sc:permute([fast, strong, smart, lucky], 2).
[ { strong, fast   },
  { smart,  fast   },
  { lucky,  fast   },
  { fast,   strong },
  { smart,  strong },
```

```
{ lucky,  strong },
{ fast,   smart  },
{ strong, smart  },
{ lucky,  smart  },
{ fast,   lucky  },
{ strong, lucky  },
{ smart,  lucky  } ]
```

Introduced in: Version 474

power_set/1

```
power_set(L) -> any()
```

Untested Stoch untested

Introduced in: Version 589

probability_all/1

```
probability_all(ListOfProbabilities) -> any()
```

Untested Stoch untested Calculates the likelihood that all items in a list of probabilities expressed on the real interval will occur.

```
1> sc:probability_all([ 0.5, 0.4, 0.3 ]).
0.06
2> sc:probability_all([ 0.5, 0.5, 0.5 ]).
0.125
3> sc:probability_all([ 0.9, 0.9, 0.9, 0.9, 0.9 ]).
0.5904900000000002
```

Notice the accumulated float rounding error.

And then, a probability result which surprises most people:

```
4> sc:probability_all([ 0.8, 0.8, 0.8, 0.8, 0.8, 0.8 ]).
0.2621440000000001
```

That's right, six 0.8s is 1 in 4. Do the math.

Thanks for the idea, Forest.

Introduced in: Version 651

probability_any/1

```
probability_any(ListOfProbabilities) -> any()
```

Incomplete Todo Comeback Untested Stoch untested

Introduced in: Version 652

qsp_average/2

```
qsp_average(W::numericlist(),
InputVecs::vectorlist()) -> float()
```

Incomplete *Untested Stoch untested* Takes the quadratic scalar product average of a vector `W` and a list of vectors `X`. The QSP Average is the arithmetic mean of the result set Y, where Y is generated as the square of the magnitude of the dot product of W and each individual vector in X. @see http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/K5.pdf pdf-page 15.

```
1> sc:qsp_average([1,2,3], [[0,0,0],[0,0,0]]).
0.0

2> sc:qsp_average([1,2,3], [[0,0,1],[0,0,0]]).
4.5

3> sc:qsp_average([1,2,3], [[0,1,0],[0,0,0]]).
2.0

4> sc:qsp_average([1,2,3], [[1,0,0],[0,0,0]]).
0.5

5> sc:qsp_average([1,2,3], [[1,1,1],[0,0,0]]).
18.0

6> sc:qsp_average([1,2,3], [[0,0,0],[1,1,1]]).
18.0

7> sc:qsp_average([1,2,3], [[1,1,1],[1,1,1]]).
36.0
```

The linked documentation incorrectly uses the notation ||Foo|| instead of |Foo| to present the algorithm. ||Foo|| is the vector

magnitude - the root sum square of vector elements - but as the input is the dot product of two 1d vectors, which will always be a single number, the vector magnitude serves no purpose other than to normalize the sign slowly and counterintuitively; thus we switch to abs despite the documentation. Thanks to Steve Stair for helping straighten this out. Thanks to the following for help with qsp_average and dependencies: Asterick, Chile, John Sensebe, PfhorSlayer, Raleigh.

Introduced in: Version 726

rand/1

```
rand(Range::integer()) -> integer()
```

Untested *Stoch untested* Returns a pseudorandom integer on the range `[0 - (Range-1)]` inclusive.

```
1> sc:rand(100).
9

2> [ sc:rand(100) || X <- lists:seq(1,10) ].
[12,27,99,86,20,96,28,36,28,15]

3> sc:histograph([ sc:rand(10) || X <- lists:seq(1,10000) ]).
[{0,992}, {1,990}, {2,992}, {3,1033}, {4,1017}, {5,1003}, {6,996}, {

4> sc:histograph([ sc:rand(10) || X <- lists:seq(1,10000) ]).
[{0,1028}, {1,979}, {2,934}, {3,970}, {4,1035}, {5,1007}, {6,986}, {
```

Introduced in: Version 595

random_from/1

```
random_from(List) -> any()
```

Equivalent to `from(1, List, no_remainder)`.

Introduced in: Version 593

random_from/2

```
random_from(N, List) -> any()
```

Equivalent to `from(N, List, no_remainder)`.

Introduced in: Version 593

random_from/3

```
random_from(N::integer(), List::list(),
WantRemainder::want_remainder()) -> list()
```

Untested *Stoch untested* Take N non-repeating random elements from a list in undefined order. If the atom `remainder` is passed in as the third argument, the unused portion of the source list will be returned as the second member of a 2ary tuple with the results; the default is no_remainder, which only returns the result set. Mixed type input lists are perfectly safe, and membership for random selection is shallow (ie, `[ [1,2], [3,4] ]` as an input list would only generate outputs of lists, never integers.)

```
1> sc:random_from([monday,tuesday,wednesday,thursday,friday]).
friday

2> sc:random_from(4, lists:seq(1,20)).
[6,3,15,12]

3> sc:random_from(3, [warrior, mage, cleric, thief, paladin, ranger,
[cleric,warrior,ranger]

4> sc:random_from(6, [mixed, [1,2,3], 4, {five,5}, 3, 67.2, <<"Hello
[[1,2,3],{five,5},4,mixed,<<"Hello">>,67.2]

5> {Team1, Team2} = sc:random_from(3, [alice,bob,cathy,dave,edward,f
{[cathy,fawn,dave],[bob,edward,alice]}

6> Team1.
[cathy,fawn,dave]

7> Where_Food = fun() -> sc:random_from([deli, fastfood, chinese, me
#Fun<erl_eval.20.67289768>

8> Where_Food().
thai
```

Introduced in: Version 593

random_from_weighted/1

```
random_from_weighted(InputList::weightlist()) ->
any()
```

Untested *Stoch untested* Take a random single item from a list with weighted probabilities. Probabilities may be any numeric type, and may be any non-negative value (items with zero probability will be omitted). Input is a `weightlist()`, which is a list in the form `[{Item,Probability}, {I2,P2}, ...]`. There is no requirement to normalize probabilities to any range, though probabilities normalized to ranges will still work as expected.

```
1> sc:from([ {quad,4}, {double,2}, {single,1} ]).
quad

2> [ sc:from_weighted([ {quad,4}, {double,2}, {single,1} ]) || X <-
[single,quad,quad,double,quad,double,quad,quad,quad,double]

3> sc:histograph([ sc:from_weighted([ {quad,4}, {double,2}, {single,
[{double,222200},{quad,444165},{single,111412}]
```

Introduced in: Version 592

random_unicode_char/0

```
random_unicode_char() -> any()
```

Introduced in: Version 660

range_scale/1

```
range_scale(NumList::numeric_list()) -> number()
```

Untested *Stoch untested* Get the scale of a same-sign numeric range. Gives nonsense results for non-numeric lists, or for lists which have both positive and negative members. For a numeric list [4,5,6,12], the scale of the range 4..12 is 3:1, which is represented as 3.0 .

```
1> sc:range_scale([3, 4, 5, 6]).
2.0

2> sc:range_scale([3, 6]).
```

```
2.0

3> sc:range_scale([6, 5, 3]).
2.0

4> sc:range_scale([3, 4, 5, 6, 7, 7.5]).
2.5

5> sc:range_scale([3, 10, 12, 99]).
33.0

6> sc:range_scale([3, 3, 3]).
1.0
```

Introduced in: Version 479

receive_one/0

```
receive_one() -> {item, any()} | nothing_there
```

Untested *Stoch untested* Pop the front of the message queue and return it as `{item,X}`, or return nothing_there for empty queues; do not block.

```
1> sc:receive_one().
nothing_there

2> self() ! message.
message

3> sc:receive_one().
{item,message}

4> sc:receive_one().
nothing_there
```

Introduced in: Version 582

record_member/2

```
record_member(E::any(), R::record()) -> true | false
```

Untested *Stoch untested* TODO: Needs Example Checks whether E is a member element of record R, analogous to

`lists::member(E, L)`. This function does not have examples because the shell does not correctly handle records; todo: add examples later

Introduced in: Version 616

replace/3

```
replace(Source, Pattern, Replacement) -> any()
```

reset_counter/1

```
reset_counter(Name) -> any()
```

Equivalent to `set_counter_value(Name, 0)`.

Untested *Stoch untested* Resets a counter's value to zero.

```
1> sc:counter_at(hello).
0

2> sc:set_counter_value(hello,4).
4

3> sc:counter_at(hello).
4

4> sc:reset_counter(hello).
0

5> sc:counter_at(hello).
0
```

Introduced in: Version 679

reverse_filter/2

```
reverse_filter(Workload, Fun) -> any()
```

Untested *Stoch untested*

Introduced in: Version 545

reverse_map/2

```
reverse_map(Workload, Fun) -> any()
```

Untested *Stoch untested*

Introduced in: Version 546

reverse_map_filter/3

```
reverse_map_filter(Workload, MapFun, FilterFun) ->
any()
```

Untested *Stoch untested*

Introduced in: Version 544

root_mean_square/1

```
root_mean_square(Values::numericlist()) -> float()
```

Untested *Stoch untested* Calculates the root mean square of the values in the list.

```
1> sc:root_mean_square([1,2,3,4,5]).
3.3166247903554

2> sc:root_mean_square([2,2,2]).
2.0
```

Introduced in: Version 505

root_sum_square/1

```
root_sum_square(VX::vector()) -> number()
```

Untested *Stoch untested* Calculate the magnitude (also known as the root sum square) of a vector.

Introduced in: Version 506

rotate_list/2

```
rotate_list(Distance::integer(), ListData::list()) -
> list()
```

Rotates the front `Distance` elements of a list to the back, in order. Negative distances rotate the back towards the front. Distances over the length of the list wrap in modulus.

```
1> sc:rotate_list(2, [1,2,3,4,5,6,7,8]).
[3,4,5,6,7,8,1,2]

2> sc:rotate_list(-2, [1,2,3,4,5,6,7,8]).
[7,8,1,2,3,4,5,6]

3> sc:rotate_list(0, [1,2,3,4,5,6,7,8]).
[1,2,3,4,5,6,7,8]

4> sc:rotate_list(16, [1,2,3,4,5,6,7,8]).
[1,2,3,4,5,6,7,8]
```

Introduced in: Version 463

rotate_to_first/2

```
rotate_to_first(Item, List) -> list()
```

Stoch untested Rotates the list to the first instance of Item.

```
1> sc:rotate_to_first(c, [a,b,c,d,e]).
[c,d,e,a,b]

2> sc:rotate_to_first(j, [a,b,c,d,e]).
no_such_element
```

Introduced in: Version 464

rotate_to_last/2

```
rotate_to_last(Item, List) -> list()
```

Stoch untested Rotates the list so that the first instance of Item becomes the last element in the list.

```
1> sc:rotate_to_last(c, [a,b,c,d,e]).
[d,e,a,b,c]

2> sc:rotate_list(j, [a,b,c,d,e]).
no_such_element
```

Introduced in: Version 464

sanitize_filename/1

```
sanitize_filename(Filename::string()) -> string()
```

Untested *Stoch untested* Sanitize an arbitrary string to be appropriate for Windows and Unix filesystems, and URLs.

```
1> sc:sanitize_filename("\h/e~l%lo! w^o@r#l*d.").
"helloworld"
```

Introduced in: Version 628

See also: sanitize_tokens/2.

sanitize_tokens/2

```
sanitize_tokens(InputList::list(),
Allowed::sanitizer()) -> list()
```

Stoch untested Remove unacceptable elements from an input list, as defined by another list or a filter function. Common reasons for sanitization include reducing arbitrary or bulk data to key format (such as using an original filename and new size to generate a new filename or database key) and removing malformed items from a list before processing.

```
1> sc:sanitize_tokens("ae0z4nb'wc-04bn ze0e 0;4ci ;e0o5rn;", "ace").
"aeceece"

2> Classifier = fun(apple) -> true; (banana) -> true; (cherry) -> tr
#Fun<erl_eval.6.13229925>

3> sc:sanitize_tokens([apple, boat, cherry, dog, elderberry], Classi
[apple,cherry,elderberry]

4> Vowels = fun($a)->true; ($e)->true; ($i)->true; ($o)->true; ($u)-
```

```
#Fun<erl_eval.6.13229925>

5> sc:sanitize_tokens("A quick brown fox jumped over the lazy dog",
"Auiooueoeeao"

6> sc:sanitize_tokens("A quick brown fox jumped over the lazy dog",
"Acbfedeead"

7> BobcatGoldthwait = fun(X) -> sc:sanitize_tokens(X, "aeiouAEIOU")
#Fun<erl_eval.6.13229925>

8> BobcatGoldthwait("A quick brown fox jumped over the lazy dog").
"Auiooueoeeao"
```

Introduced in: Version 477

See also: sanitize_filename/1.

second_difference/1

```
second_difference(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

send_receive/2

```
send_receive(ToWhom::pid() | atom(), What::any()) ->
{item, any()}
```

Untested *Stoch untested* (Blocking) First send a message to an entity. Then pop the front of the message queue and return it as `{item,X}`; block.

```
1> sc:send_receive(self(), message).
{item,message}
```

Introduced in: Version 578

send_receive/3

```
send_receive(ToWhom::pid() | atom(), What::any(),
HowLong::non_negative_integer() | infinity) ->
{item, any()} | nothing_there
```

Untested *Stoch untested* (Non-Blocking) First send a message to an entity. Then pop the front of the message queue and return it as `{item,X}`, or return nothing_there for empty queues; do not block.

```
1> sc:send_receive(self(), message).
{item,message}
```

Introduced in: Version 579

send_receive_masked/3

```
send_receive_masked(Mask::any(), ToWhom::pid() |
atom(), What::any()) -> {Mask, any()}
```

Untested *Stoch untested* (Blocking) First send a message to an entity. Then pop the first message queue item matching the mask as a 2-tuple, and return it as `{Mask,X}`; block.

```
1> sc:send_receive(self(), message).
{item,message}
```

Introduced in: Version 580

send_receive_masked/4

```
send_receive_masked(Mask::any(), ToWhom::pid() |
atom(), What::any(), HowLong::non_negative_integer()
| infinity) -> {item, any()} | nothing_there
```

Untested *Stoch untested* (Non-Blocking) First send a message to an entity. Then pop the front of the message queue and return it as `{Mask,X}`, or return nothing_there for empty queues; do not block.

```
1> sc:send_receive(self(), message).
{item,message}
```

Introduced in: Version 581

set_counter_value/2

```
set_counter_value(Name::any(), To::number()) -> 0
```

Untested *Stoch untested* Sets a counter's value to a specific value.

```
1> sc:counter_at(hello).
0

2> sc:set_counter_value(hello,4).
4

3> sc:counter_at(hello).
4

4> sc:reset_counter(hello).
0

5> sc:counter_at(hello).
0
```

Introduced in: Version 678

shared_keys/1

```
shared_keys(TupleList::sorted_keylist()) ->
sorted_keylist()
```

Stoch untested Create sorted list X of 3-ary tuples {K,Ai,Bi} from sorted lists A, B of 2ary {K,Ai}/{K,Bi} tuples, where key K appears in both A and B.

```
1> sc:shared_keys([{1,a},{2,a},{3,a}],[{1,b},{3,b},{4,b}]).
[{1,a,b},{3,a,b}]

2>sc:shared_keys([{1,a},{2,a}],[{3,b},{4,b}]).
[]
```

Introduced in: Version 475

shared_keys/2

```
shared_keys(TupleList::sorted_keylist(),
Presorted::presorted) -> sorted_keylist()
```

Equivalent to shared_keys(lists:zip(lists:sort(A), lists:sort(B))).

Stoch untested Equivalent to shared_keys/1, but skips sorting the lists (and thus requires pre-sorted lists), which may save significant work repetition.

shared_keys/3

```
shared_keys(A::sorted_keylist(),
B::sorted_keylist(), Presorted::presorted) ->
sorted_keylist()
```

Equivalent to shared_keys(lists:sort(A), lists:sort(B)).

Stoch untested Equivalent to shared_keys/2, but skips sorting the lists (and thus requires pre-sorted lists), which may save significant work repetition.

show/1

```
show(X) -> any()
```

Untested *Stoch untested*

Introduced in: Version 637

shuffle/1

```
shuffle(List::list()) -> list()
```

Untested *Stoch untested* Return a list with the original list's shallow members in a random order. Deep lists are not shuffled; `[ [a,b,c], [d,e,f], [g,h,i] ]` will never produce sublist reorderings (`[b,c,a]`) or list mixing (`[b,g,e]`), only reordering of the three top level lists. The output list will always be the same length as the input list. Repeated items and mixed types in input lists are safe.

```
1> sc:shuffle(lists:seq(1,9)).
[8,4,7,9,5,2,6,1,3]

2> {TheFaces, TheSuits} = {  [ace] ++ lists:seq(2,10) ++ [jack,queen
{[ace,jack,queen,king,2,3,4,5,6,7,8,9,10],
 [hearts,spades,clubs,diamonds]}

3> Deck = sc:shuffle([ {Face,Suit} || Face <- TheFaces, Suit <- TheS
[ {6,spades}, {7,hearts}, {8,clubs}, {queen,spades}, {6,diamonds}, {

4> sc:shuffle([ duck,duck,duck,duck, goose ]).
[duck,goose,duck,duck,duck]
```

Originally found at http://wiki.trapexit.org/index.php/RandomShuffle; refactored for clarity, and unnecessary repeat nesting behavior removed.

Introduced in: Version 590

simple_ranking/1

```
simple_ranking(Values::numericlist()) ->
rankinglist()
```

Untested *Stoch untested* Returns a ranked ordering of the list without tie rankings.

```
1> sc_rank:simple([10,90,20,80,30,70,40,60,50]).
[{1,90}, {2,80}, {3,70}, {4,60}, {5,50}, {6,40}, {7,30}, {8,20}, {9,

2> sc_rank:simple([10,10,10,10]).
[{1,10},{2,10},{3,10},{4,10}]
```

Introduced in: Version 560

skewness/1

```
skewness(List) -> any()
```

Equivalent to central_moment(List, 3).

Untested *Stoch untested*

Introduced in: Version 493

spearman_correlation/1

```
spearman_correlation(TupleList::coordlist()) ->
{rsquared, Correlation::number()}
```

Untested *Stoch untested* Compute the Spearman's Rank Correlation Coefficient of a list of coordinate tuples.

```
1> sc:spearman([ {1,1}, {2,2}, {3,3}, {4,4}, {5,5} ]).
{rsquared,1.0}

2> sc:spearman([ {1,5}, {2,4}, {3,3}, {4,2}, {5,1} ]).
{rsquared,-1.0}

3> sc:spearman([ {1,3}, {2,3}, {3,3}, {4,3}, {5,3} ]).
{rsquared,0.5}

4> sc:spearman([ {1,2}, {2,2.5}, {3,3}, {4,3.5}, {5,4} ]).
{rsquared,1.0}

5> sc:spearman([ {1,2}, {2,2.4}, {3,3}, {4,3.6}, {5,4} ]).
{rsquared,1.0}
```

Introduced in: Version 558

spearman_correlation/2

```
spearman_correlation(List1, List2) -> any()
```

Equivalent to spearman_correlation(lists:zip(List1, List2)).

split_at/2

```
split_at(N, List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 541 TODO

square/1

```
square(Input::number()) -> number()
```

Untested *Stoch untested* Squares the input; convenient in list comprehensions to prevent recalculation, and clear in the fashion of documentary functions.

```
1> sc:square(2).
4

2> sc:square(2.5).
6.25
```

Introduced in: Version 508

srand/0

```
srand() -> {ok, {seeded, Seed}}
```

Untested *Stoch untested* *(Called automatically)* Instantiates the random source, destroying a prior source if needed, and seeds the source with the clock, returning the seed used. Generally speaking, you do not need this function; this is used manually when you want to know what seed was used, for purposes of recreating identical pseudorandom sequences. Otherwise, rand() will call this once on its own. ***Because the scutil random system spawns a utility process to maintain random state, this function should be considered to have side effects for purposes of testing.*** (Indeed, in a sense, this function's entire purpose is to cause a side effect.)

```
1> sc:srand().
{ok,{seeded,{1227,902172,685000}}}

2> sc:srand().
{ok,{seeded,{1227,902173,231000}}}
```

Introduced in: Version 598

srand/3

```
srand(A::integer(), B::integer(), C::integer()) ->
{ok, {seeded, Seed}}
```

Untested *Stoch untested* *(Called automatically)* Instantiates the random source, destroying a prior source if needed, and seeds the source with the three integer seed you provide, returning the

seed used. Generally speaking, you do not need this function; this is used manually when you want set what seed is used, for purposes of recreating identical pseudorandom sequences. Otherwise, rand() will call this once on its own. ***Because the scutil random system spawns a utility process to maintain random state, this function should be considered to have side effects for purposes of testing.*** (Indeed, in a sense, this function's entire purpose is to cause a side effect.)

```
1> sc:srand(1,2,3).
{ok,{seeded,{1,2,3}}}

2> sc:srand().
{ok,{seeded,{1227,902568,604600}}}

3> sc:srand(1,2,3).
{ok,{seeded,{1,2,3}}}
```

Introduced in: Version 598

standard_card_backs/0

```
standard_card_backs() -> list()
```

Incomplete *Untested* *Stoch untested* Returns the list of colors which are used, in order, as the standard back colors of a series of decks for multi_deck/2. Each color is presented as an atom.

```
1> sc:standard_card_backs().
[ red, blue, green, black, purple, orange, brown, yellow,
  teal, gray, cyan, indigo, pink, white, tan, maroon,
  navy, forest, leaf, sky, brick ]

2> length(sc:standard_card_backs()).
24
```

Introduced in: Version 728

standard_card_backs/1

```
standard_card_backs(Count::positive_integer()) ->
list()
```

Incomplete *Untested* *Stoch untested* Returns the front of the list of colors which are used, in order, as the standard back colors of a series of decks for multi_deck/2. Each color is presented as an atom. If you request more colors than are in the list, the list `[1,2...Count]` is provided instead.

```
1> sc:standard_card_backs(5).
[ red, blue, green, black, purple ]

2> sc:standard_card_backs(29).
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,2
```

Introduced in: Version 728

start_register_if_not_running/2

```
start_register_if_not_running(Name, FunctionLambda)
-> any()
```

Untested *Stoch untested*

Introduced in: Version 635

start_register_if_not_running/3

```
start_register_if_not_running(Name, Module,
Function) -> any()
```

Equivalent to start_register_if_not_running(node(), Name, Module, Function, []).

Untested *Stoch untested*

Introduced in: Version 618

start_register_if_not_running/4

```
start_register_if_not_running(Name, Module,
Function, Args) -> any()
```

Equivalent to start_register_if_not_running(node(), Name, Module, Function, Args).

Untested *Stoch untested*

start_register_if_not_running/5

```
start_register_if_not_running(Node::atom(),
Name::atom(), Module::atom(), Function::atom(),
Args::list()) -> pid() | ok
```

Untested *Stoch untested* Check whether a process is registered locally, and if not, spawn it with a give function and arguments.

```
1> whereis(test).
undefined

2> sc:start_register_if_not_running(node(), test, scutil, wait_until
{ ok, <0.726.0> }

3> whereis(test).
<0.726.0>

4> test ! terminate.
terminate

5> whereis(test).
undefined

6> sc:start_register_if_not_running(node(), test, scutil, wait_until
{ ok, <0.731.0> }

7> whereis(test).
<0.731.0>

8> sc:start_register_if_not_running(node(), test, scutil, wait_until
{ ok, <0.731.0> }

9> whereis(test).
<0.731.0>
```

Introduced in: Version 618

starts_with/2

```
starts_with(Remain, PRemain) -> any()
```

Untested *Stoch untested*

Introduced in: Version 624

stretch_hash/3

```
stretch_hash(State::list_or_binary(),
HashFun::function(), ListOfSalts::list()) ->
binary()
```

Untested *Stoch untested* Stretches a hash with a list of salts. Some people incorrect refer to this as key strentghening. The process is a simple key-derivation function: repeat the application of a hash with a different pre-pend salt each time.

```
1> Res = sc:stretch_hash("abc", fun erlang:md5/1, ["def", "ghi", "jk
<<129,166,92,224,108,140,78,205,151,136,77,203,166,229,62,186>>

2> sc:bin_to_hex_list(Res).
"81a65ce06c8c4ecd97884dcba6e53eba"

C:\projects\scutil\res>php
<?php echo md5("mno" . md5("jkl" . md5("ghi" . md5("defabc", true),
^Z
81a65ce06c8c4ecd97884dcba6e53eba
```

Thanks Josh, Davr, Vat.

Introduced in: Version 641

svn_revision/1

```
svn_revision(ModuleName::atom()) -> integer()
```

Untested *Stoch untested* Scans a module for an attribute svn_revision, parses it in the format expected from the svn:keyword Revision, and returns the version number as an integer. To use, add a module attribute to your module as follows: `-svn_revision("$+Revision$).`, after removing the plus (if the plus wasn't there, the example would get corrupted when I updated the module `;)`). Then set the svn keyword "Revision" on the file, and check it in. After that, your version is magically updated every time you check in! `:D` The sole argument

to this function is the name of the module to be scanned, as an atom.

```
1> sc:scan_svn_revision(testerl).
16
```

Introduced in: Version 523

terminate_loop/0

```
terminate_loop() -> any()
```

Untested Stoch untested

Introduced in: Version 636

test/0

```
test() -> ok | error
```

(not testworthy) Runs the test suite in terse form.

```
1> sc:test().
  All 9 tests passed.
ok
```

Introduced in: 458

test/1

```
test(Style::verbose) -> ok | error
```

(not testworthy) Runs the test suite in verbose form. Also responds to [verbose] to be more familiar to eunit devs. An (ancient) example of output:

```
1> sc:test(verbose).
======================== EUnit ========================
module 'sc'
  module 'sc_tests'
    Index of first tests
      sc_tests:73: index_of_first_test_ (0,   [ ])...ok
      sc_tests:74: index_of_first_test_ (b,   [ a,b,c ])...ok
      sc_tests:75: index_of_first_test_ (g,   [ a,b,c ])...ok
```

```
      [done in 0.046 s]
    Rotate list tests
      sc_tests:52: rotate_list_test_ (0,  [ ])...ok
      sc_tests:53: rotate_list_test_ (1,  [ ])...ok
      sc_tests:54: rotate_list_test_ (-1, [ ])...ok
      sc_tests:56: rotate_list_test_ (0,  [ a,b,c ])...ok
      sc_tests:57: rotate_list_test_ (1,  [ a,b,c ])...ok
      sc_tests:58: rotate_list_test_ (-1, [ a,b,c ])...ok
      sc_tests:59: rotate_list_test_ (3,  [ a,b,c ])...ok
      sc_tests:60: rotate_list_test_ (-3, [ a,b,c ])...ok
      sc_tests:61: rotate_list_test_ (9,  [ a,b,c ])...ok
      [done in 0.141 s]
    Key duplicate tests
      sc_tests:38: key_duplicate_test_ ([ ])...ok
      sc_tests:39: key_duplicate_test_ ([ {2,a} ])...ok
      sc_tests:40: key_duplicate_test_ ([ {2,a},{3,b} ])...ok
      [done in 0.047 s]
    Extrema tests
      sc_tests:19: extrema_test_ (1,2,3,4)...ok
      sc_tests:20: extrema_test_ (-1,-2,-3)...ok
      sc_tests:21: extrema_test_ (-1.1,0,1.1)...ok
      sc_tests:22: extrema_test_ (a,b,c)...ok
      sc_tests:23: extrema_test_ (1,a,{})...ok
      sc_tests:24: extrema_test_ (1)...ok
      sc_tests:26: extrema_test_ ([] error)...ok
      [done in 0.109 s]
    [done in 0.343 s]
  [done in 0.343 s]
=======================================================
  All 22 tests passed.
ok
```

Introduced in: 460

third_difference/1

```
third_difference(List) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

tied_ordered_ranking/1

```
tied_ordered_ranking(Values::numericlist()) ->
rankinglist()
```

Untested *Stoch untested* Returns a tied ranked ordering of the list, ordered according to the input ordering rather than the sorted ordering. As with tied/1, all rankings are floats, and ties are represented as the centers of ranges.

```
1> sc:ordered([10,90,20,80,30,70,40,60,50]).
[{9.0,10}, {1.0,90}, {8.0,20}, {2.0,80}, {7.0,30}, {3.0,70}, {6.0,40

2> sc:ordered([100,200,200,300]).
[{4.0,100},{2.5,200},{2.5,200},{1.0,300}]
```

Introduced in: Version 562

tied_ranking/1

```
tied_ranking(Values::numericlist()) -> rankinglist()
```

Untested *Stoch untested* Returns a ranked ordering of the list with tie rankings. As such, for uniformity, all rankings are floats. Ties are represented as the centers of ranges.

```
1> sc:tied([10,90,20,80,30,70,40,60,50]).
[{1.0,90}, {2.0,80}, {3.0,70}, {4.0,60}, {5.0,50}, {6.0,40}, {7.0,30

2> sc:tied([100,200,200,300]).
[{1.0,300},{2.5,200},{2.5,200},{4.0,100}]
```

needs significant refactoring; work is being repeated

Introduced in: Version 561

time_diff/2

```
time_diff(A::timestamp(), B::timestamp()) -> float()
```

Returns the difference, in seconds as a float, between two erlang timestamps as returned by `erlang:now()`. Negative differences are returned if the latter timestamp `B` is earlier than the former timestamp `A`.

```
1> A = now().
{1232,947675,340000}

2> B = now().
{1232,947679,412000}

3> sc:time_diff(A,B).
4.072

4> sc:time_diff(B,A).
-4.072
```

Introduced in: Version 742

to_lines/1

```
to_lines(Text::string()) -> stringlist()
```

Untested *Stoch untested* Cuts a string according to any of the three newline conventions (even mixed), and discards empty strings. Mostly convenience and documentary.

```
1> sc:to_lines("one\rtwo\nthree\r\nfour\r\r\rfive").
["one","two","three","four","five"]
```

Introduced in: Version 705

to_list/1

```
to_list(X) -> any()
```

Untested *Stoch untested*

Introduced in: Version 638

triangle_index/1

```
triangle_index(X) -> any()
```

Untested *Stoch untested*

Introduced in: Version 632

triangle_index/2

```
triangle_index(X1, Y) -> any()
```

Untested *Stoch untested*

Introduced in: Version 632

tuple_member/2

```
tuple_member(E::any(), T::tuple()) -> true | false
```

Untested *Stoch untested* Checks whether E is a member element of tuple T, analogous to `lists::member(E, L)`.

```
1> sc:tuple_member(b, {a,b,c}).
true

2> sc:tuple_member(d, {a,b,c}).
false

3> sc:tuple_member([1,2], {[1,2]}).
true
```

Introduced in: Version 615

tuple_sort/1

```
tuple_sort(T) -> any()
```

Untested *Stoch untested*

Introduced in: 610

tuple_sum/1

```
tuple_sum(T::relaxed_numeric_tuple()) -> number()
```

Untested *Stoch untested* Returns the sum of the numeric elements of a tuple, treating non-numeric elements as zero.

```
1>
```

Introduced in: Version 609

type_of/1

```
type_of(Argument::any()) -> typelabel()
```

Untested *Stoch untested* Fetch the type of the argument. Valid for any term. Fails before erlang 12, due to use of `is_bitstring()` .

```
1> sc:type_of(1).
integer

2> sc:type_of({hello,world}).
tuple
```

Introduced in: Version 722

unfunnel/2

```
unfunnel(Tgt, ProbPropList) -> any()
```

Untested *Stoch untested* Reverse a marketing funnel, to go from goal needed to input needed.

```
1> % Using the data from http://www.forentrepreneurs.com/lessons-fro
1> sc:unfunnel(300, [{1/4,"Web activity scoring"}, {1/3,"Telemarketi
[ { 14400, "Input Needed" },
  { 3600,  "Web activity scoring", 0.25 },
  { 1200,  "Telemarketing",        0.3333333333333333 },
  { 300,   "Inside Sales",         0.25 },
  { 300,   "Result" } ]
```

Introduced in: Version 691

unfunnel/3

```
unfunnel(Tgt, ProbPropList, MaybeCeil) -> any()
```

union/1

```
union(L) -> any()
```

Untested *Stoch untested*

Introduced in: Version 586

union/2

```
union(L1, L2) -> any()
```

Equivalent to `union([L1, L2])`.

Untested *Stoch untested*

Introduced in: Version 586

union/3

```
union(L1, L2, L3) -> any()
```

Equivalent to `union([L1, L2, L3])`.

Untested *Stoch untested*

Introduced in: Version 586

union/4

```
union(L1, L2, L3, L4) -> any()
```

Equivalent to `union([L1, L2, L3, L4])`.

Untested *Stoch untested*

Introduced in: Version 586

unit_scale/1

```
unit_scale(Waveform) -> any()
```

Introduced in: Version 720

vector_magnitude/1

```
vector_magnitude(VX) -> any()
```

Equivalent to `root_sum_square(VX)`.

Introduced in: Version 506

vector_normalize/1

```
vector_normalize(Vector::vector()) -> unit_vector()
```

Incomplete *Untested* *Stoch untested* Returns the magnitude of a vector. A vector's magnitude is the length of its hypoteneuse. A vector can be seen as the product of its unit vector and its magnitude; as such many people see a vector's magnitude as its scale. The normal of the zero vector is undefined, in the way that dividing by zero is undefined, and will throw an arithmetic exception.

```
1> sc:vector_normalize([0,3,4]).
[0.0,0.6,0.8]
```

TODO: When tuple comprehensions are introduced to the language, convert this to using them.

Introduced in: Version 725

walk_unique_pairings/2

```
walk_unique_pairings(R, F) -> any()
```

Untested *Stoch untested*

Introduced in: Version 547

weighted_arithmetic_mean/1

```
weighted_arithmetic_mean(InputList::weightlist()) ->
float()
```

Untested *Stoch untested* Take the weighted arithmetic mean of the input values.

```
1> sc:weighted_arithmetic_mean([ {8,1}, {3,4}, {16,1} ]).
6.0
```

Introduced in: Version 484

See also: amean_vector_normal/1, arithmetic_mean/1.

zip_n/1

```
zip_n(Ls::list()) -> list_of_tuples()
```

Equivalent to zip_n(Ls, to_tuple).

zip_n/2

```
zip_n(Ls::list(), ResultType::atom()) ->
list_of_tuples()
```

Stoch untested Computes a zip on any sized group of lists, rather than just two or three as offered by the lists module.

```
1> sc:zip_n([ [1,2,3], [a,b,c], [i,ii,iii] ]).
[{1,a,i},{2,b,ii},{3,c,iii}]

2> sc:zip_n([ [1,2,3], [a,b,c], [i,ii,iii], [x,y,z], [red,blue,green
[{1,a,i,x,red,april},
 {2,b,ii,y,blue,may},
 {3,c,iii,z,green,june}]
```

This is actually more efficient than one might expect at first glance. I ran a benchmark of 100,000 transformations of a list of lists into a list of tuples using benchmark/3 and multi_do/4 against both zip_n and the library function zip3; the library function won at 150 seconds to 175, which is a far smaller difference than I expected.

```
3> Testy = [ [1,2,3], [1,2,3], [1,2,3] ].
[[1,2,3],[1,2,3],[1,2,3]]

4> sc:benchmark(sc, multi_do, [100000, sc, zip_n, [Testy]]).
{174.95563, [[{1,1,1},{2,2,2},{3,3,3}], [{1,1,1},{2,2,2},{3,3,3}], .

5> sc:benchmark(sc, multi_do, [100000, lists, zip3, Testy]).
{149.605, [[{1,1,1},{2,2,2},{3,3,3}], [{1,1,1},{2,2,2},{3,3,3}], ...
```

Thanks Thanks to Vladimir Sessikov for contributing this to and thus allowing conscription from the mailing list.

Introduced in: Version 472

zipf_estimate_list/1

```
zipf_estimate_list(PosNumericList::positive_numeric_list())
-> positive_numeric_list()
```

Untested *Stoch untested* Estimates the zipf baseline from each number in a numeric list.

```
1> sc:zipf_estimate_list([ 120, 60, 40, 30, 24, 20 ]).
[120, 120, 120, 120, 120, 120]

2> sc:zipf_estimate_list([411,198,135,101,82]).
[411, 396, 405, 404, 410]

3> sc:zipf_estimate_list([630,298,231,180,118]).
[630, 596, 693, 720, 590]
```

Introduced in: Version 480

zipf_nearness/1

```
zipf_nearness(PosNumericList::positive_numeric_list())
-> number()
```

Untested *Stoch untested* todo.

```
1> sc:zipf_nearness([ 120, 60, 40, 30, 24, 20 ]).
[[ {strength,1.0}, {center,120.0} ],
 [ {strength,1.0}, {center,120.0} ],
 [ {strength,1.0}, {center,120.0} ],
 [ {strength,1.0}, {center,120.0} ],
 [ {strength,1.0}, {center,120.0} ],
 [ {strength,1.0}, {center,120.0} ]]

2> sc:zipf_nearness([ 411, 198, 135, 101, 82 ]).
[[{strength,0.9635036496350365}, {center,405.2}],
 [{strength,0.9658536585365854}, {center,403.75}],
 [{strength,0.9853658536585366}, {center,406.3333333333333}],
 [{strength,0.9853658536585366}, {center,407.0}],
 [{strength,1.0},                {center,410.0}]]

3> sc:zipf_nearness([640,244,231,180,148]).
[[{strength, 0.6594594594594595}, {center,656.2}],
 [{strength, 0.6594594594594595}, {center,660.25}],
```

```
[{strength, 0.9364864864864865}, {center,717.6666666666666}],
[{strength, 0.972972972972973},  {center,730.0}],
[{strength, 1.0},                {center,740.0}]]
```

Introduced in: Version 480

zipf_position_estimate/2

```
zipf_position_estimate(Score::number(),
Rank::positive_integer()) -> number()
```

Untested *Stoch untested* Estimates the zipf baseline from a score and a rank position.

```
1> sc:zipf_position_estimate(120, 3).
360
```

Introduced in: Version 480

Generated by EDoc, Nov 26 2011, 14:07:08.