



# Array Functions

```
? $a =~ $b
```

Determine if two scalars have the same identity (value). Scalar identity is a means of determining if two scalars are equivalent values or not. The identity algorithm compares references for object scalars and function scalars. The string representation is used to compare other scalars.

```
@ add(@array, $scalar, [position])
```

Inserts a scalar into an array at a certain position.

```
@ add(%hash, key => value, ...)
```

Adds any number of key/value pairs to the specified hash.

```
@ addAll(@a, @b)
```

Adds all of the non-present elements of @b into @a. Essentially this function computes the union of @ and @b.

```
$ cast(@array, 't', ...)
```

Casts @array into an object scalar representing a native java array.

```
$ cast("string", 'b'|'c')
```

Casts the specified string of byte data into a 1-dimensional native java byte or character array

```
clear(@array)
```

Removes all of the contents from @array.

## Array Functions

`clear(%hash)`

Removes all of the contents from *%hash*.

`@ concat(@a, @b, [...])`

Concatenates the specified arrays into one.

`@ copy(@array)`

Returns a shallow copy of the specified array.

`$ copy($scalar)`

Returns a shallow copy of the specified scalar.

`% copy(%hash)`

Returns a shallow copy of the specified hash.

`@ filter(&closure, @|&)`

Applies the specified closure to each element of the second argument and returns an array of all non-*\$null* return values.

`@ flatten(@array)`

Returns a shallow copy of the specified array flattened to 1-dimension.

`? $a in @array`

Determine if a scalar with identity *\$a* is contained in *@array*.

`@ map(&closure, @|&)`

Applies the specified closure to each element of the second argument and returns an array of all return values.

`$ pop(@array)`

Removes the last element from *@array* and returns it.

`$ push(@array, $value, ...)`

Adds the specified values to the end of the specified array.

## Array Functions

`$ reduce(@|&, &closure)`

Applies `&closure` to the first two elements of the specified array or closure. The resulting value is then applied to the next value of the specified array or closure, so on and so forth. Returns one value.

`remove(@array, $scalar, ...)`

Removes all of the specified values from the array.

`remove(%hash, $scalar, ...)`

Removes all of the specified values from the hash.

`remove()`

This version of `&remove` should only be used within a `foreach` loop. This form removes the current active element of the `foreach` loop.

`@ removeAll(@a, @b)`

Removes all of the elements of `@b` from `@a`. This is the set difference operation on `@a` and `@b`.

`removeAt(@array, index, ...)`

removes the element located at `index` from `@array`.

`removeAt(%hash, "index", ...)`

removes the element associated with "index" from `%hash`.

`@ retainAll(@a, @b)`

Removes all of the elements of `@a` not present in `@b`. This is the set intersection operation on `@a` and `@b`.

`@ reverse(@|&)`

Returns a copy of the specified array/iterator in reverse order.

`$ search(@array, &closure, [index])`

Applies the specified `&closure` to each element of the specified array until a non-`$null` value is returned.

`$ shift(@array)`

Removes the first element from `@array` and returns it.

## Array Functions

`$ size(@array)`

return the number of elements in *@array*.

`$ size(%hash)`

return the number of elements in *%hash*.

`@ sort(&closure, @array)`

Sorts the specified array using the specified closure for comparisons.

`@ sorta(@array)`

Sorts the specified array alphabetically.

`@ sortd(@array)`

Sorts the specified array in numerical order as double values.

`@ sortn(@array)`

Sorts the specified array in numerical order as long values.

`@ splice(@array, @insert, [n], [m])`

removes *m* elements starting at position *n* from *@array* and splices in the contents of *@insert*.

`@ sublist(@array, start, [end])`

Extracts a subset of the specified array from the specified start index up to but not including the specified end index.

## Date/Time Functions

\$ formatDate([date], 'format')

formats the specified date as a string with the specified format.

\$ parseDate('format', "date string")

parses the specified date string into a scalar long.

\$ ticks()

obtain the current time in milliseconds

# File System

`-canread "file"`

A predicate to check if a file is readable

`-canwrite "file"`

A predicate to check if a file is writeable

`-exists "file"`

A predicate to check if a file exists

`-isDir "file"`

A predicate to check if a file is a directory

`-isFile "file"`

A predicate to check if a file is a file (i.e. not a directory)

`-isHidden "file"`

A predicate to check if a file is hidden

`$ chdir("directory")`

changes the current working directory to the specified directory.

## File System

```
$ createNewFile("file")
```

Creates an empty file at the specified file location.

```
$ cwd()
```

returns the current working directory.

```
$ deleteFile("file")
```

Deletes the specified file/directory.

```
$ getFileName("/path/file")
```

Extracts the file portion of the specified path

```
$ getFileParent("/path/path/file")
```

Extracts the parent path of the specified file/directory

```
$ getFileProper("path", "file", ...)
```

Concatenates all arguments into a single coherent path with appropriate separators.

```
$ lastModified("file")
```

obtain the last modified time of the specified file.

```
@ listRoots()
```

Lists all of the root directories.

```
$ lof("path/file")
```

Obtain the size of the specified file.

```
@ ls("path")
```

Lists all of the files/directories within the specified path.

```
$ mkdir("directory/subdirectory/...")
```

Creates the specified directory.

```
$ rename("old", "new")
```

Rename the specified file.

## File System

```
$ setLastModified("file", time)
```

set the last modified time of the specified file.

```
$ setReadOnly("file")
```

set the read only attribute of the specified file.

## Hash Functions

```
@ add(@array, $scalar, [position])
```

Inserts a scalar into an array at a certain position.

```
@ add(%hash, key => value, ...)
```

Adds any number of key/value pairs to the specified hash.

```
clear(@array)
```

Removes all of the contents from *@array*.

```
clear(%hash)
```

Removes all of the contents from *%hash*.

```
@ copy(@array)
```

Returns a shallow copy of the specified array.

```
$ copy($scalar)
```

Returns a shallow copy of the specified scalar.

```
% copy(%hash)
```

Returns a shallow copy of the specified hash.

## Hash Functions

```
@ keys(%hash)
```

Generates an array containing the keys within the specified hash.

```
% ohash(key => value, ...)
```

Creates an ordered Sleep hash. All keys are stored in insertion order.

```
% ohasha(key => value, ...)
```

Creates an ordered Sleep hash. All keys are stored in access order from least to most recently accessed.

```
% putAll(%hash, @|&keys, @|&values)
```

Populates the hash with data obtained from iterating over the key and value sources simultaneously.

```
% putAll(%hash, @|&source)
```

Populates the hash with data obtained from iterating over the specified source for key and value values.

```
remove(@array, $scalar, ...)
```

Removes all of the specified values from the array.

```
remove(%hash, $scalar, ...)
```

Removes all of the specified values from the hash.

```
remove()
```

This version of `&remove` should only be used within a `foreach` loop. This form removes the current active element of the `foreach` loop.

```
setMissPolicy(%ohash, &closure)
```

Sets the miss policy for an ordered hash. The miss policy is called when a key with no associated value is requested. The ordered hash is then populated with the value returned by the miss policy closure.

```
setRemovalPolicy(%ohash, &closure)
```

Sets the removal policy for an ordered hash. The removal policy is called when a new value is added to the hash. The return value of the policy determines whether the last element in the hash should be removed or not.

## Hash Functions

`$ size(@array)`

return the number of elements in *@array*.

`$ size(%hash)`

return the number of elements in *%hash*.

`@ values(%hash)`

Generates an array containing the values within the specified hash.

## Input/Output Functions

`-eof $handle`

A predicate to check if the reader portion of the handle is closed (end of file)

`$ allocate([initial size])`

allocates a writeable memory buffer. calling `closef` on the returned buffer turns it into a readable buffer. calling `closef` on a readable buffer frees the buffer.

`$ available([$handle])`

Obtain the number of bytes that can be read from handle without blocking.

`$ available($handle, "delim")`

Read ahead in the handle to see if the delimiter is present in the buffer or not.

`@ bread([$handle], 'format')`

reads data from `$handle`. Returned as a scalar array with types specified by the format string

`$ bwrite([$handle], 'format', $x, ...)`

writes data to `$handle`. each format character corresponds to one or more arguments.

`$ bwrite([$handle], 'format', @array)`

writes data to `$handle`. each format character corresponds to one or more array elements.

## Input/Output Functions

```
closef($handle)
```

Closes the read and write channels for the specified I/O handle

```
closef(port)
```

Locates the specified port in the server socket listen table and closes it

```
$ connect("host", port, [timeout], [&closure], [option => value, ...])
```

connects to the specified host:port and returns a \$handle. Check for issues connecting to a host with checkError(). If &closure is specified, this call will not block. &closure will be called when a connection is established.

```
$ exec("command", [%env], ["directory"])
```

executes the specified command and returns a \$handle.

```
$ exec(@command, [%env], ["directory"])
```

executes the first element of the command array and returns a \$handle. this form of exec is useful for passing arguments that have whitespace in them.

```
$ fork(&closure, [$key => $value, ...])
```

Creates a new thread, a new script environment, and executes the specified &closure. The returned \$handle acts as a pipe between the thread and the new script environment. Within the script environment \$source is available to act as an outward pipe.

```
$ getConsole()
```

returns the \$handle for stdin/stdout.

```
$ listen(port, [timeout], [$host], [&closure], [option => value, ...])
```

Instantiates a server socket to listen for TCP/IP connections on the specified port and accepts a connection.

```
mark([$handle], n)
```

marks the current point in this IO stream. a buffer is created allowing the mark to be &reset until n bytes has been reached.

```
$ openf("[>>|>]file")
```

opens the specified file for read or write

## Input/Output Functions

```
print([$handle], "text")
```

prints "text" to the specified handle (with no newline)

```
printAll([$handle], @array|&generator)
```

prints entire contents of passed in @array or &generator to \$handle. each element has a newline appended to it.

```
printEOF([$handle])
```

signals EOF (End of File) on the far end by shutting down output for \$handle

```
println([$handle], "text")
```

prints "text" to the specified handle (with a newline appended)

```
@ readAll([$handle])
```

reads all lines of text from the specified handle and places them into an array

```
$ readAsObject([$handle])
```

reads a serialized object from the specified handle

```
$ readb([$handle], n)
```

reads n bytes from \$handle. If 0 bytes are read \$null will be returned.

```
$ readb([$handle], -1, [est_size])
```

reads bytes from handle until none are left. Returns *\$null* when 0 bytes are read.

```
$ readc([$handle])
```

reads a single unicode character from the specified handle

```
$ readln([$handle])
```

reads a single line of text from the specified handle

```
$ readObject([$handle])
```

reads a serialized scalar back from the specified handle

```
reset([$handle])
```

resets this IO stream back to the last &mark

## Input/Output Functions

`setEncoding($handle, "charset name")`

sets the character set to encode/decode written/read characters with the specified handle.

`$ sizeof('format')`

calculates the size of the data structure specified by the format string.

`$ skip($handle, n, [buffer size])`

reads and discards up to n bytes from the specified handle. this is useful for causing data to be read and processed without the expensive conversion process to sleep strings (i.e. when one wants to `&digest` or `&checksum` a file)

`$ wait($handle, [timeout])`

Blocks and waits for the callback, process, or fork associated with \$handle to finish. if \$handle is a fork, the return value of the fork will be returned by `&wait`. if \$handle is a process, the return value of the process will be returned by `&wait`. If the specified timeout is reached \$null will be returned.

`writeAsObject([$handle], $object, ...)`

serializes and writes all the object representations of the scalar arguments out to the specified handle

`writeb([$handle], "string")`

writes the bytes contained in "string" to \$handle

`writeObject([$handle], $scalar, ...)`

serializes and writes all of the scalar arguments out to the specified handle

# Math Functions

`$a <=> $b`

performs a numerical comparison of `$a` and `$b`

`$ abs(n)`

Calculate the absolute value of the argument.

`$ acos(n)`

Calculate the arc cosine of the argument.

`$ asin(n)`

Calculate the arc sine of the argument. (answer in radians)

`$ atan(n)`

Calculate the arc tangent of the argument. (answer in radians)

`$ atan2(n, m)`

Calculate the arc tangent of angle `n / m`.

`$ ceil(n)`

Rounds the specified value up to the next integer value.

## Math Functions

`$ checksum("string", 'algorithm')`

Returns (as a scalar long) the checksum of the specified byte string using the specified algorithm.

`$ checksum($handle, '[>]algorithm')`

Sets up the specified handle so that all reads (or if the algorithm is prefixed with `>`, writes) are checksummed using the specified algorithm. Returns a `$checksum` object that can be used to obtain the final digest value.

`$ checksum($checksum)`

Returns (as a scalar long) the checksum of the handle associated with `$checksum`.

`$ cos(n)`

Calculate the cosine of the argument. (answer in radians)

`$ degrees(n)`

Converts the angle `n` measured in radians to an approximately equivalent angle in degrees.

`$ digest("string", 'algorithm')`

Returns (as a byte string) the digest of the specified byte string using the specified algorithm.

`$ digest($handle, '[>]algorithm')`

Sets up the specified handle so that all reads (or if the algorithm is prefixed with `>`, writes) are checksummed using the specified algorithm. Returns a `$digest` object that can be used to obtain the final digest value.

`$ digest($digest)`

Returns (as a byte string) the digest of the handle associated with `$digest`.

`$ double(n)`

Convert the specified value to a double scalar

`$ exp(n)`

Returns Euler's number raised to the power of `n`

`$ floor(n)`

Rounds the specified value down to the previous integer value.

## Math Functions

`$ formatNumber(number, [from], to)`

Parses the specified number string using the specified base system.

`$ int(n)`

Convert the specified value to an int scalar

`$ log(n, [base])`

Calculate the logarithm of the specified argument.

`$ long(n)`

Convert the specified value to a long scalar

`$ not(n)`

Calculate the logical not value of the argument.

`$ parseNumber("number", [base])`

Parses the specified number string using the specified base system.

`$ radians(n)`

Converts the angle n measured in degrees to an approximately equivalent angle in radians.

`$ rand([number])`

generates a random integer between 0 and number. If number is omitted the function generates a random double between 0 and 1

`$ rand(@array)`

returns a random element of @array

`$ round(n)`

Rounds the specified value to the nearest integer value.

`$ round(n, places)`

Rounds the specified value to the specified number of places.

`$ sin(n)`

Calculate the sine of the argument. (answer in radians)

## Math Functions

\$ `sqrt(n)`

Calculate the rounded positive square root value of the argument.

\$ `srand([number])`

seed the random number generator with the specified scalar (interpreted as a long)

\$ `tan(n)`

Calculate the tangent of the argument. (answer in radians)

\$ `uint(n)`

Interpret the specified value as an unsigned integer

## String Functions

```
$ asc("c")
```

Returns a scalar integer of the ascii value of the specified character

```
$ byteAt("string", n)
```

Returns the byte at the n'th position in the string

```
$ cast(@array, 't', ...)
```

Casts @array into an object scalar representing a native java array.

```
$ cast("string", 'b'|'c')
```

Casts the specified string of byte data into a 1-dimensional native java byte or character array

```
$ chr(n)
```

Returns a string containing the character that corresponds to the integer argument.

```
$ charAt("string", n)
```

Returns the character at the n'th position in the string

```
$a cmp $b
```

performs an alphabetical comparison of \$a and \$b

## String Functions

```
$ find("string", 'pattern', [start])
```

Returns the index of the first substring that matches 'pattern' starting from the specified index.

```
"string" hasmatch 'pattern'
```

Determine if the string contains a substring that matches the specified pattern. Subsequent calls to this predicate with the same string, pattern combination will check if there is another pattern beyond the first one.

```
$ indexOf("string", "substr", [start])
```

Returns the index of "substr" inside of "string" starting at the specified start index.

```
"string" ismatch 'pattern'
```

Determine if the string matches the specified pattern.

```
? '*filter*' iswm "string"
```

Determine if the specified wildcard pattern is a match to the string.

```
$ join("string", @array|&closure)
```

joins the elements of @array with "string"

```
$ lc("STRiNG")
```

Returns a lowercase version of the specified string

```
$ left("string", n)
```

Returns the left n characters of "string"

```
$ lindexOf("string", "substr", [start])
```

Returns the last index of "substr" inside of "string" counting backwards from the specified start index.

```
@ matched()
```

returns the matches from a "string" applied to a regex 'pattern' during an ismatch/hasmatch check

```
@ matches("string", 'pattern', [n], [m])
```

returns the matches from "string" applied to the regex 'pattern'. if n is specified this will return the grouped matches of the n'th substring matching the specified pattern. if n and m are specified, all of the grouped matches of the n-m substrings will be returned.

## String Functions

```
$ mid("string", start, [length])
```

Returns a substring of the specified "string" starting from the start index followed by the next n chars

```
$ pack('format', $x, ...)
```

packs data into a string of bytes. each format character corresponds to one or more arguments.

```
$ pack('format', @array)
```

packs data into a string of bytes. each format character corresponds to one or more array elements.

```
$ replace("string", 'pattern', "new", [n])
```

Replaces each substring of the specified string that matches the regular expression pattern with the specified new string.

```
$ replaceAt("string", "new", index, [n])
```

Replaces n characters starting at the specified index with the new string.

```
$ right("string", n)
```

Returns the right n characters of "string"

```
@ split('pattern', "string", [limit])
```

splits the specified string by the specified pattern

```
$ strlen("string")
```

Returns the length of the specified string.

```
$ strrep("string", "old", "new", ...)
```

Replaces occurrences of old with new in string. accepts multiple old, new parameters.

```
$ substr("string", start, [end])
```

Extracts a substring of the specified string from the specified start index up to but not including the specified end index.

```
$ tr("string", "matcher", "replacement", ['options'])
```

A character translation utility similar to the UNIX tr command. A transliteration consists of a pattern of characters to match and a pattern, typically of equal length, of characters to replace each match with. The tr utility loops through a specified string character by character. Each character is compared against the pattern of matching characters. If the character matches one of the pattern

## String Functions

characters it is either replaced with the replacement character mapped to that matcher or it is deleted. Which action is taken depends on what options are specified.

```
$ uc("string")
```

Returns a uppercase version of the specified string

```
@ unpack('format', "string")
```

unpacks data from the specified sleep string. data is returned as a sleep array with each scalar set to a type as specified in the format string

## Utility Functions

`acquire($semaphore)`

blocks the current thread of execution until the semaphore count is  $> 0$ , when that happens the semaphore count is decremented.

`$ casti($scalar, 't')`

casts an individual `$scalar` into an object scalar representing a Java value

`$ checkError([$error])`

Returns the last error message to occur.

`& compile_closure("code", ...)`

Creates a new Sleep closure from the specified string of Sleep code.

`@ copy(@array)`

Returns a shallow copy of the specified array.

`$ copy($scalar)`

Returns a shallow copy of the specified scalar.

`% copy(%hash)`

Returns a shallow copy of the specified hash.

## Utility Functions

```
$ debug([level])
```

query/set the debug level for the current script environment.

```
$ eval("code")
```

Parses and evaluates the specified sleep code returning the value of the code.

```
exit(["reason"])
```

Causes the currently executing script to stop executing.

```
$ expr("expr")
```

Parses and evaluates the specified sleep expression code returning the value of the expression.

```
& function('&name')
```

Obtains a reference to the specified function.

```
@ getStackTrace()
```

Within the context of a catch block, this function will return a trace of the Sleep call stack that caused the caught exception condition to occur. Returns an empty array otherwise.

```
global('$x $y')
```

Parses the specified string and declares all variables in the string as global variables.

```
$ iff(comparison, [$iftrue], [$iffalse])
```

Takes a comparison as the first parameter and returns its second parameter if and only if the condition is true. The third parameter is returned if and only if the condition is false.

```
use(['/path/to/file.jar'], 'script.sl')
```

Compiles and executes the specified script in the current script context.

```
inline(&closure)
```

Dynamically invokes the specified closure as if it was an inline function occurring within the local scope.

```
$ invoke(&closure, @args, ["message"], [key => value, ...])
```

Dynamically invokes the specified closure. Allows programmatic specification of arguments, key/value pairs, and the message parameter.

## Utility Functions

? \$a is \$b

Determine if *\$a* references the same data as *\$b*

? \$a isa ^Class

Determine if object value of *\$a* is an instance of the specified ^Class

& lambda(&closure, [\$key => "value", ...])

Copies *&closure* into a new closure. The new closure environment is initialized with all of the specified key/value pair arguments.

& let(&closure, \$key => "value", ...)

Updates the specified closure's environment with all of the key/value pair arguments. Returns the specified closure.

local('\$x \$y')

Parses the specified string and declares all variables in the string as local variables.

\$ newInstance(^Class|@array, &closure)

Creates an instance of the specified Java interface (or interfaces if an array is used) backed by the specified closure.

popl([\$var => value, ...])

removes current local scope restoring previous scope.

@ profile()

Returns the profiler statistics for the current script environment. Profiler statistics will only be collected if `DEBUG_TRACE_CALLS` (8) or `DEBUG_TRACE_PROFILE_ONLY` (24) are enabled.

pushl([\$var => value, ...])

creates an additional local scope.

release(\$semaphore)

increments the count value of the specified semaphore. notifies other threads waiting on this semaphore

\$ scalar(\$object)

Runs the specified object through the Java type to Sleep scalar conversion process. This same process used by HOES.

## Utility Functions

```
$ semaphore(initial_count)
```

Creates a counting semaphore suitable for use with `acquire` and `release`. A semaphore is a synchronization primitive used to protect critical sections of code.

```
setf('&function', &closure)
```

Binds a closure to the specified function name.

```
setField(^Class|$object, field => value, ...)
```

Sets any number of public/protected fields of the specified class or instance of `$object` to their corresponding values.

```
sleep(n)
```

Blocking call that forces the current executing thread to sleep for `n` milliseconds

```
% systemProperties()
```

Returns a hash of the available system properties.

```
$ taint($scalar)
```

Taints the specified scalar

```
this('$x $y')
```

Parses the specified string and declares all variables in the string as variables specific to the scope of the current closure.

```
^ typeOf($scalar)
```

Returns the Java class of the container referenced by *\$scalar*

```
$ untaint($scalar)
```

Untaints the specified scalar

```
use(^Class)
```

Installs the specified class (which is a *sleep.interfaces.Loadable*) into the current Sleep environment. This is a way of extending the Sleep language at runtime.

```
use(['/path/to/file.jar'], 'Loadable')
```

Dynamically loads a specified Sleep bridge and installs it into the current Sleep environment. This is a way of extending the Sleep language at runtime with Sleep bridges.

## Utility Functions

```
warn("text")
```

Prints "text" to the registered runtime warning watcher. Provides an application neutral way to print messages to the Sleep console.

```
watch('$var @ar')
```

Declares all ovariables in the string as "watch" variables. Any attempt to set a value in a watched container will print out a warning. The warning does not prevent the setting of the variable. The value will change as normal.