

Eidos Reference Sheet

14 September 2022

Types (in promotion order):

NULL: no explicit value
logical: true/false values
integer: whole numbers
float: real numbers
string: characters
object: Eidos objects,
such as SLiM objects

Constants:

E: e (2.7182...) (**float**)
PI: π (3.1415...) (**float**)
F: false (**logical**)
T: true (**logical**)
INF: infinity (**float**)
NAN: not a number (**float**)
NULL: a **NULL**-type value

Operators (precedence order):

[], (), . subset, call, member
^ exponentiation
+, -, ! unary plus/minus, logical not
: sequence construction
***, /, %** multiplication, division, modulo
+, - addition and subtraction
<, >, <=, >= less-than, greater-than, etc.
==, != equality and inequality
& logical (Boolean) and
| logical (Boolean) or
?else ternary conditional
= assignment

Empty statement: ;
Compound statement: { ... }
Single-line comment: // ...
Block comment: /* ... */

if (condition) statement [**else** statement]
while (condition) statement
do statement **while** (condition)
for (identifier **in** vector) statement
next / **break**
return [return-value]
function (return)name(params) { ... }

conditional statement with optional alternative statement
loop while **T**, with a condition test at the loop top
loop while **T**, with a condition test at the loop bottom
iterate through the values in a vector, executing statement
skip the rest of this iteration, or exit a loop entirely
exit a script block, returning a value if one is given
create a user-defined function (only at the top level)

Math:

(**numeric**)**abs**(**numeric** x): absolute value of x
(**float**)**acos**(**numeric** x): arc cosine of x
(**float**)**asin**(**numeric** x): arc sine of x
(**float**)**atan**(**numeric** x): arc tangent of x
(**float**)**atan2**(**numeric** x, **numeric** y): arc tangent of y/x, inferring the correct quadrant
(**float**)**ceil**(**float** x): ceiling (rounding toward $+\infty$) of x
(**float**)**cos**(**numeric** x): cosine of x
(**numeric**)**cumProduct**(**numeric** x): cumulative product along x
(**numeric**)**cumSum**(**numeric** x): cumulative summation along x
(**float**)**exp**(**numeric** x): base-e exponential of x, e^x
(**float**)**floor**(**float** x): floor (rounding toward $-\infty$) of x
(**integer**)**integerDiv**(**integer** x, **integer** y): integer division of x by y
(**integer**)**integerMod**(**integer** x, **integer** y): integer modulo of x by y (the remainder after integer division)
(**logical**)**isFinite**(**float** x): T or F for each element of x; "finite" means not **INF**, **-INF**, or **NAN**
(**logical**)**isInfinite**(**float** x): T or F for each element of x; "infinite" means **INF** and **-INF** only
(**logical**)**isNAN**(**float** x): T or F for each element of x; "infinite" means **NAN** only
(**float**)**log**(**numeric** x): base-e logarithm of x
(**float**)**log10**(**numeric** x): base-10 logarithm of x
(**float**)**log2**(**numeric** x): base-2 logarithm of x
(**numeric**\$)**product**(**numeric** x): product of the elements of x, $\prod x$
(**float**)**round**(**float** x): round x to the nearest values; half-way cases round away from 0
(***setDifference**(* x, * y): set-theoretic difference, $x \setminus y$
(***setIntersection**(* x, * y): set-theoretic intersection, $x \cap y$
(***setSymmetricDifference**(* x, * y): set-theoretic symmetric difference $x \Delta y$
(***setUnion**(* x, * y): set-theoretic union, $x \cup y$
(**float**)**sin**(**numeric** x): sine of x
(**float**)**sqrt**(**numeric** x): square root of x
(**numeric**\$)**sum**(**lif** x): summation of the elements of x, $\sum x$
(**float**\$)**sumExact**(**float** x): exact summation of x without roundoff error, to the limit of floating-point precision
(**float**)**tan**(**numeric** x): tangent of x
(**float**)**trunc**(**float** x): truncation (rounding toward 0) of x

Statistics:

(float\$)cor(numeric x, numeric y): sample Pearson's correlation coefficient between x and y
(float\$)cov(numeric x, numeric y): corrected sample covariance between x and y
(+\$)max(+ x, ...): largest value within x and the additional optional arguments
(float\$)mean(lif x): arithmetic mean of x
(+\$)min(+ x, ...): smallest value within x and the additional optional arguments
(+)pmax(+ x, + y): parallel maximum of x and y (the element-wise maximum for each corresponding pair)
(+)pmin(+ x, + y): parallel minimum of x and y (the element-wise maximum for each corresponding pair)
(float)quantile(numeric x, [Nf probs = NULL]): quantiles of x
(numeric)range(numeric x, ...): range (min/max) of x and the additional optional arguments
(float\$)sd(numeric x): corrected sample standard deviation of x
(float\$)ttest(float x, [Nf y = NULL], [Nf\$ mu = NULL]): run a one-sample or two-sample *t*-test
(float\$)var(numeric x): corrected sample variance of x

Vector construction:

(*)c(...): concatenate the given vectors to make a single vector of uniform type
(float)float(integer\$ length): construct a float vector of length, initialized with 0.0
(integer)integer(integer\$ length, [integer\$ fill1 = 0], [integer\$ fill2 = 1], [Ni fill2indices = NULL]): construct an integer vector of length, initialized with the given fill values
(logical)logical(integer\$ length): construct a logical vector of length, initialized with F
(object<Object>)object(void): construct an empty object vector
(*)rep(* x, integer\$ count): repeat x a given number of times
(*)repEach(* x, integer count): repeat each element of x a given number of times
(*)sample(* x, integer\$ size, [logical\$ replace = F], [Nif weights = NULL]): sample from x
(numeric)seq(n\$ from, n\$ to, [Nif\$ by = NULL], [Ni\$ length = NULL]): construct a sequence
(integer)seqAlong(* x): construct a sequence along the indices of x
(integer)seqLen(integer\$ length): construct a sequence with length elements, counting upward from 0
(string)string(integer\$ length): construct a string vector of length, initialized with ""

Value inspection / manipulation:

(logical\$)all(logical x, ...): T if *all* values supplied are T, otherwise F
(logical\$)any(logical x, ...): T if *any* values supplied are T, otherwise F
(void)cat(* x, [s\$ sep = " "], [l\$ error = F]): concatenate output
(void)catn([* x = ""], [s\$ sep = " "], [l\$ error = F]): concatenate output with trailing newline
(string)format(string\$ format, numeric x): format the elements of x as strings
(logical\$)identical(* x, * y): T if x and y are identical in all respects, otherwise F
(*)ifelse(logical test, * trueValues, * falseValues): vector conditional
(integer\$)length(* x): count elements in x (synonymous with size())
(integer)match(* x, * table): positions of matches for x within table
(integer)order(+ x, [logical\$ ascending = T]): indexes of x that would produce sorted order
(string\$)paste(..., [string\$ sep = " "]): paste together a string with separators
(string\$)paste0(...): paste together a string with no separators
(void)print(* x, [l\$ error = F]): print x to the output stream
(*)rev(* x): reverse the order of the elements in x
(integer\$)size(* x): count elements in x (synonymous with length())
(+)sort(+ x, [logical\$ ascending = T]): sort non-object vector x
(object)sortBy(object x, string\$ property, [l\$ ascending = T]): sort object vector x by a property
(void)str(* x, [l\$ error = F]): print the external structure of a value
(integer)tabulate(integer bin, [Ni\$ maxbin = NULL]): tabulate occurrence counts of values in bin
(*)unique(* x, [logical\$ preserveOrder = T]): unique values in x (preserveOrder = F is faster)
(integer)which(logical x): indices in x which are T
(integer\$)whichMax(+ x): first index in x with the maximum value
(integer\$)whichMin(+ x): first index in x with the minimum value

Distribution drawing / density:

(float)dmvnorm(float x, numeric mu, numeric sigma): multivariate normal density function values
(float)dbeta(float x, numeric alpha, numeric beta): beta distribution density function values
(float)dexp(float x, [numeric mu = 1]): exponential distribution density function values
(float)dgamma(float x, numeric mean, numeric shape): gamma distribution density function values
(float)dnorm(float x, [numeric mean = 0], [numeric sd = 1]): normal density function values
(integer)findInterval(numeric x, numeric vec, [logical\$ rightmostClosed = F],
[logical\$ allInside = F]): find interval indices
(float)pnorm(float q, [numeric mean = 0], [numeric sd = 1]): normal distribution CDF values
(float)qnorm(float p, [numeric mean = 0], [numeric sd = 1]): normal distribution quantile values
(float)rbeta(integer \$n, numeric alpha, numeric beta): beta distribution draws
(integer)rbinom(integer \$n, integer size, float prob): binomial distribution draws
(float)rcauchy(integer \$n, [numeric location = 0], [numeric scale = 1]): Cauchy distribution draws
(integer)rdunif(integer \$n, [integer min = 0], [integer max = 1]): discrete uniform distribution draws
(float)rexp(integer \$n, [numeric mu = 1]): exponential distribution draws
(float)rf(integer \$n, numeric d1, numeric d1): *F*-distribution draws
(float)rgamma(integer \$n, numeric mean, numeric shape): gamma distribution draws
(integer)rgeom(integer \$n, float p): geometric distribution draws
(float)rlnorm(integer \$n, [numeric meanlog = 0], [numeric sdlog = 1]): lognormal distribution draws
(float)rmvnorm(integer \$n, numeric mu, numeric sigma): multivariate normal distribution draws
(integer)rnbinom(integer \$n, integer size, float prob): negative binomial distribution draws
(float)rnorm(integer \$n, [numeric mean = 0], [numeric sd = 1]): normal distribution draws
(integer)rpois(integer \$n, numeric lambda): Poisson distribution draws
(float)runif(integer \$n, [numeric min = 0], [numeric max = 1]): uniform distribution draws
(float)rweibull(integer \$n, numeric lambda, numeric k): Weibull distribution draws

Type testing / coercion:

(float)asFloat(+ x): convert x to type float
(integer)asInteger(+ x): convert x to type integer
(logical)asLogical(+ x): convert x to type logical
(string)asString(+ x): convert x to type string
(string)\$elementType(* x): element type of x; for object x, this is the class of the object-elements
(logical)\$isFloat(* x): T if x is of type float, F otherwise
(logical)\$isInteger(* x): T if x is of type integer, F otherwise
(logical)\$isLogical(* x): T if x is of type logical, F otherwise
(logical)\$isNULL(* x): T if x is of type NULL, F otherwise
(logical)\$isObject(* x): T if x is of type object, F otherwise
(logical)\$isString(* x): T if x is of type string, F otherwise
(string)\$type(* x): type of vector x; this is NULL, logical, integer, float, string, or object

String manipulation:

(lis)grep(string\$ pattern, string x, [logical\$ ignoreCase = F],
[string\$ grammar = "ECMAScript"], [string\$ value = "indices"], [logical\$ fixed = F],
[logical\$ invert = F]): regular expression substring matching
(integer)nchar(string x): character counts for the string values in x
(logical)strcontains(string x, string\$ s, [i\$ pos = 0]): check for occurrence of s in x from pos
(integer)strfind(string x, string\$ s, [i\$ pos = 0]): find first occurrences of s in x from pos
(logical)strprefix(string x, string\$ s): check for prefix s in x
(string)strsplit(string\$ x, [string\$ sep = " "]): split string x into substrings by separator sep
(logical)strsuffix(string x, string\$ s): check for suffix s in x
(string)substr(string x, integer first, [Ni last = NULL]): get substrings from x

Color manipulation:

(string)colors(numeric x, string\$ name): generate color strings from the named color palette
(float)color2rgb(string color): convert color string(s) to RGB values
(float)hsv2rgb(float hsv): convert HSV color(s) to RGB values
(string)rainbow(integer\$ n, [float\$ s = 1], [float\$ v = 1], [float\$ start = 0], [Nf\$ end = NULL], [logical\$ ccw = T]): generate colors in a "rainbow" color palette
(string)rgb2color(float rgb): convert RGB color(s) to color string(s)
(float)rgb2hsv(float rgb): convert RGB color(s) to HSV values

Filesystem access:

(logical\$)createDirectory(string\$ path): create a new filesystem directory at path
(logical\$)deleteFile(string\$ filePath): delete file at filePath
(logical\$)fileExists(string\$ filePath): check for the existence of a file (or directory) at filePath
(string)filesAtPath(string\$ path, [logical\$ fullPaths = F]): get the names of the files in a directory
(logical\$)flushFile(string\$ filePath): flush any buffered content for the file at filePath
(string\$)getwd(void): get the current filesystem working directory
(object<DataFrame>)\$readCSV(string\$ filePath, [ls colNames = T], [Ns\$ colTypes = NULL], [string\$ sep = ",", [string\$ quote = "'"], [string\$ dec = "."], [string\$ comment = ""]): read tabular data from a CSV/TSV file to create a new DataFrame
(string)readFile(string\$ filePath): read lines from the file at filePath as a string vector
(string\$)setwd(string\$ path): set the filesystem working directory
(string\$)tempdir(void): get the path for a directory suitable for temporary files
(logical\$)writeFile(string\$ filePath, string contents, [logical\$ append = F], [logical\$ compress = F]): write to a file
(string\$)writeTempFile(string\$ prefix, string\$ suffix, string contents, [logical\$ compress = F]): write to a temporary file

Miscellaneous:

(void)assert(logical assertions, [Ns\$ message = NULL]): assert that condition(s) are true; if not, stop
(void)beep([Ns\$ soundName = NULL]): play a sound or beep
(void)citation(void): print the reference citation for Eidos and the current Context
(float\$)clock([string\$ type = "cpu"]): get the current CPU usage clock, for timing of code blocks
(string\$)date(void): get the current date as a formatted string
(string\$)debugIndent(void): get the current indentation string for debugging output
(void)defineConstant(string\$ symbol, * value): define a new constant with a given value
(void)defineGlobal(string\$ symbol, * value): define a new global variable with a given value
(*)doCall(string\$ functionName, ...): call the named function with the given arguments
(*)executeLambda(string\$ lambdaSource, [ls\$ timed = F]): execute a string as code
(logical)exists(string symbol): T for defined symbols, F otherwise
(void)functionSignature([Ns\$ functionName = NULL]): print the call signature(s) for function(s)
(void)functionSource(string\$ functionName): print the Eidos source code (if any) for a function
(integer\$)getSeed(void): get the last random number generator seed set
(void)license(void): print license information for Eidos and the current Context
(void)ls([logical\$ showSymbolTables = F]): list all variables currently defined
(void)rm([Ns\$ variableNames = NULL]): remove (undefine) variables
(*)sapply(* x, string\$ lambdaSource, [string\$ simplify = "vector"]): apply code across elements of x
(void)setSeed(integer\$ seed): set the random number generator seed
(void)source(string\$ filePath, [logical\$ chdir = F]): execute a source file as code
(void)stop([Ns\$ message = NULL]): stop execution and print the given error message
(logical\$)suppressWarnings(logical\$ suppress): suppress (or stop suppressing) warning messages
(*)sysinfo(string\$ key): get information about the system – operating system, hardware, etc.
(string)system(string\$ command, [string args = ""], [string input = ""], [logical\$ stderr = F], [logical\$ wait = T]): run a Un*x command with the given arguments and input
(string\$)time(void): get the current time as a formatted string
(float\$)usage([logical\$ peak = F]): get the current or peak memory usage of the process
(float)version([logical\$ print = T]): get the Eidos and Context version numbers

Matrix and array functions:

(*)`apply(* x, integer margin, string$ lambdaSource)`: apply code across margins of matrix/array x
(*)`array(* data, integer dim)`: create an array from data, with dimensionality dim
(*)`cbind(...)`: combine vectors and/or matrices by column
(integer)`dim(* x)`: dimensions of matrix or array x
(*)`drop(* x)`: drop redundant dimensions from matrix or array x
(*)`matrix(* data, [Ni$ nrow = NULL], [Ni$ ncol = NULL], [logical$ byrow = F])`: create a matrix
(numeric)`matrixMult(numeric x, numeric y)`: matrix multiplication of conformable matrices x and y
(integer\$)`ncol(* x)`: number of columns in matrix or array x
(integer\$)`nrow(* x)`: number of rows in matrix or array x
(*)`rbind(...)`: combine vectors and/or matrices by row
(*)`t(* x)`: transpose of x

Class Object:

Superclass: none

+ (integer\$)`length(void)`: count elements in the target object vector (synonymous with `size()`)
+ (void)`methodSignature([Ns$ methodName])`: print the signature for `methodName`, or for all methods
+ (void)`propertySignature([Ns$ propertyName])`: print the signature for `propertyName`, or for all properties
+ (integer\$)`size(void)`: count elements in the target object vector (synonymous with `length()`)
- (void)`str(void)`: print the internal structure (properties, types, values) for an object vector
- (string\$)`stringRepresentation(void)`: returns the string representation that `print()` would print

Class Dictionary:

Superclass: *Object*

(object<Dictionary>)\$`Dictionary(...)`: creates a new Dictionary, with several variants:
(object<Dictionary>)\$`Dictionary()`: new empty Dictionary
(object<Dictionary>)\$`Dictionary(string$ key, *$ value, ...)`: new Dictionary with key-value pairs
(object<Dictionary>)\$`Dictionary(object<Dictionary>$ d)`: new Dictionary that is a copy of d
(object<Dictionary>)\$`Dictionary(string$ json)`: new Dictionary from a JSON serialization

`allKeys => (string)`: a vector of all keys that have been assigned a value

- (void)`addKeysAndValuesFrom(object$ source)`: adds key-value pairs from source
- (void)`appendKeysAndValuesFrom(object source)`: appends key-value pairs from source
- (void)`clearKeysAndValues(void)`: removes all key-value pairs
- (*)`getValue(string$ key)`: fetch the value assigned to key (or NULL if no value is assigned)
- (object<Dictionary>)\$`getRowValues(li index, [logical$ drop = F])`: returns selected "rows"
- (logical\$)`identicalContents(object$ x)`: returns T if the target contains identical keys and values to x
- (string)`serialize(string$ format)`: returns a string representation ('slim' / 'json' / 'csv' / 'tsv')
- (void)`setValue(string$ key, * value)`: sets a key-value pair

Class DataFrame:

Superclass: *Dictionary*

(object<Dictionary>)\$`DataFrame(...)`: creates a new DataFrame, with the same variants as `Dictionary()`
see also the `readCSV()` function, which creates a DataFrame from a CSV/TSV file

`colNames => (string)`: a vector of the names of all columns

`dim => (integer)`: a vector of dimensions (rows, columns)

`ncol => (integer$)`: the number of columns

`nrow => (integer$)`: the number of rows

- (*)`asMatrix(void)`: returns a matrix representation of the target DataFrame
- (void)`cbind(object source, ...)`: adds columns from source and ..., increasing the target's width
- (void)`rbind(object source, ...)`: adds rows from source and ..., increasing the target's height
- (*)`subset([Nli rows = NULL], [Nlis cols = NULL])`: returns selected elements, as a DataFrame or vector
- (object<DataFrame>)\$`subsetColumns(lis index)`: returns selected columns
- (object<DataFrame>)\$`subsetRows(li index, [logical$ drop = F])`: returns selected rows

Class Image:

Superclass: *Dictionary*

`(object<Image>)` `Image(string$ filePath)`: creates a new `Image` object from the PNG file at `filePath`

`width => (integer$)`: the width of the image, in pixels

`height => (integer$)`: the height of the image, in pixels

`isGrayscale => (logical$)`: T if the image is grayscale, F if it is RGB

`bitsPerChannel => (integer$)`: the number of bits used to represent one channel of the image (R/G/B/K)

`integerR => (integer)`: the red (R) channel of the image, represented as a 2D integer matrix

`integerG => (integer)`: the green (G) channel of the image, represented as a 2D integer matrix

`integerB => (integer)`: the blue (B) channel of the image, represented as a 2D integer matrix

`integerK => (integer)`: the black (K) channel of the image, represented as a 2D integer matrix

`floatR => (float)`: the red (R) channel of the image, represented as a 2D float matrix

`floatG => (float)`: the green (G) channel of the image, represented as a 2D float matrix

`floatB => (float)`: the blue (B) channel of the image, represented as a 2D float matrix

`floatK => (float)`: the black (K) channel of the image, represented as a 2D float matrix

- `(void)write(string$ filePath)`: write PNG data for the image to `filePath`