ArrayFunc

Authors: Michael Griffin
Version: 6.0.0 for 2020-03-27

Language: Python 3.5 or later

Table of Contents

Introduction

The ArrayFunc module provides high speed array processing functions for use with the standard Python array module. These functions are patterned after the functions in the standard Python Itertools module together with some additional ones from other sources.

The purpose of these functions is to perform mathematical calculations on arrays significantly faster than using native Python.

Function Summary

The functions fall into several categories.

Filling Arrays

Function	Description
count	Fill an array with evenly spaced values using a start and step values.
cycle	Fill an array with evenly spaced values using a start, stop, and step values, and repeat until the array is filled.
repeat	Fill an array with a specified value.

Filtering Arrays

Function	Description
afilter	Select values from an array based on a boolean criteria.
compress	Select values from an array based on another array of boolean values.
dropwhile	Select values from an array starting from where a selected criteria fails and proceeding to the end.
takewhile	Like dropwhile, but starts from the beginning and stops when the criteria fails.

Examining and Searching Arrays

Function	Description
findindex	Returns the index of the first value in an array to meet the specified criteria.
findindices	Searches an array for the array indices which meet the specified criteria and writes the results to a second array. Also returns the number of matches found.

Summarising Arrays

Function	Description
aany	Returns True if any element in an array meets the selected criteria.
aall	Returns True if all element in an array meet the selected criteria.

amax	Returns the maximum value in the array.
amin	Returns the minimum value in the array.
asum	Calculate the arithmetic sum of an array.

Data Conversion

Function	Description
convert	Convert arrays between data types. The data will be converted into the form required by the output array.

Mathematical operator functions

Function	Equivalent to
abs_	[abs(x) for x in array1]
add	[x + param for x in array1]
floordiv	[x // param for x in array1]
mod	[x % param for x in array1]
mul	[x * param for x in array1]
neg	[-x for x in array1]
pow	[x ** param for x in array1]
sub	[x - param for x in array1]
truediv	[x / param for x in array1]

Comparison operator functions

Function	Equivalent to
eq	all([x == param for x in array1])
ge	all([x >= param for x in array1])
gt	all([x > param for x in array1])
le	all([x <= param for x in array1])
lt	all([x < param for x in array1])
ne	all([x != param for x in array1])

Bitwise operator functions

Function	Equivalent to
and_	[x x & y param for x in array1]
invert	[~x for x in array1]
lshift	[x x << y param for x in array1]
or_	[x x | y param for x in array1]
rshift	[x x >> y param for x in array1]

xor	[x x ^ y param for x in array1]

Power and logarithmic functions

Function	Equivalent to
exp	[math.exp(x) for x in array1]
expm1	[math.expm1(x) for x in array1]
log	[math.log(x) for x in array1]
log10	[math.log10(x) for x in array1]
log1p	[math.log1p(x) for x in array1]
log2	[math.log2(x) for x in array1]
sqrt	[math.sqrt(x) for x in array1]

Hyperbolic functions

Function	Equivalent to
acosh	[math.acosh(x) for x in array1]
asinh	[math.asinh(x) for x in array1]
atanh	[math.atanh(x) for x in array1]
cosh	[math.cosh(x) for x in array1]
sinh	[math.sinh(x) for x in array1]
tanh	[math.tanh(x) for x in array1]

Trigonometric functions

Function	Equivalent to
acos	[math.acos(x) for x in array1]
asin	[math.asin(x) for x in array1]
atan	[math.atan(x) for x in array1]
atan2	[atan2(x, param) for x in array1]
cos	[math.cos(x) for x in array1]
hypot	[hypot(x, param) for x in array1]
sin	[math.sin(x) for x in array1]
tan	[math.tan(x) for x in array1]

Angular conversion

Function	Equivalent to
degrees	[math.degrees(x) for x in array1]
radians	[math.radians(x) for x in array1]

Number-theoretic and representation functions

Function	Equivalent to
ceil	[math.ceil(x) for x in array1]
copysign	[copysign(x, param) for x in array1]
fabs	[math.fabs(x) for x in array1]
factorial	[math.factorial(x) for x in array1]
floor	[math.floor(x) for x in array1]
fmod	[fmod(x, param) for x in array1]
isfinite	all([isfinite(x) for x in array1])
isinf	any([isinf(x) for x in array1])
isnan	any([isnan(x) for x in array1])
ldexp	math.ldexp(x, y)
trunc	[math.trunc(x) for x in array1]

Special functions

Function	Equivalent to
erf	[math.erf(x) for x in array1]
erfc	[math.erfc(x) for x in array1]
gamma	[math.gamma(x) for x in array1]
lgamma	[math.lgamma(x) for x in array1]

Additional functions

Function	Equivalent to
fma	[(x * param2 + param3) for x in array1]

Array Limit Attributes

In addition to functions, a set of attributes are provided representing the platform specific maximum and minimum numerical values for each array type. These attributes are part of the "arraylimits" module.

Searching and Summarising Arrays.

Comparison Operators

Some functions use comparison operators. These are unicode strings containing the Python compare operators and include following:

Operator	Description
'<'	Less than.

'<='	Less than or equal to.
'>'	Greater than.
'>='	Greater than or equal to.
'=='	Equal to.
'!='	Not equal to.

All comparison operators must contain only the above characters and may not include any leading or trailing spaces or other characters.

Description

aall

Calculate aall over the values in an array.

Equivalent to:	all([(x > param) for x in array])
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = aall(opstr, array, param)
result = aall(opstr, array, param, maxlen=y)
result = aall(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If any comparison operations result in true, the return value will be true. If all of them result in false, the return value will be false.

aany

Calculate aany over the values in an array.

Equivalent to:	any([(x > param) for x in array])
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = aany(opstr, array, param)
result = aany(opstr, array, param, maxlen=y)
result = aany(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

afilter

Select values from an array based on a boolean criteria.

Equivalent to:	filter(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = afilter(opstr, array, outparray, param)
result = afilter(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

amax

Calculate amax over the values in an array.

Equivalent to:	max(x)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = amax(array)
result = amax(array, maxlen=y)
result = amax(array, nosimd=False)
```

- array - The input data array to be examined.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The maximum of all the values in the array.

amin

Calculate amin over the values in an array.

Equivalent to:	min(x)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = amin(array)
result = amin(array, maxlen=y)
result = amin(array, nosimd=False)
```

- array - The input data array to be examined.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The minimum of all the values in the array.

asum

Calculate the arithmetic sum of an array.

Equivalent to:	sum()
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = asum(array)
result = asum(array, maxlen=y)
result = asum(array, nosimd=False)
result = asum(array, matherrors=False)
```

- array - The input data array to be examined.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- matherrors - If True, checks for numerical errors including integer overflow are ignored.
- result - The sum of the array.

compress

Select values from an array based on another array of integers values. The selector array is interpreted as a set of boolean values, where any value other than *0* causes the value in the input array to be selected and copied to theoutput array, while a value of *0* causes the value to be ignored.

Equivalent to:	itertools.compress(inparray, selectorarray)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
x = compress(inparray, outparray, selectorarray)
x = compress(inparray, outparray, selectorarray, maxlen=y)
```

- inparray - The input data array to be filtered.
- outparray - The output array.
- selectorarray - The selector array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- x - An integer count of the number of items filtered into outparray.

convert

Convert arrays between data types. The data will be converted into the form required by the output array. If any values in the input array are outside the range of the output array type, an exception will be raised. When floating point values are converted to integers, the value will be truncated.

Equivalent to:	[x for x in inputarray]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
convert(inparray, outparray)
convert(inparray, outparray, maxlen=y)
```

- inparray - The input data array to be filtered.
- outparray - The output array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

count

Fill an array with evenly spaced values using a start and step values.

Equivalent to:	itertools.count(start, len(array))
or	itertools.count(start, len(array), step)

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
count(array, start, step).
```

- array - The output array.
- start - The numeric value to start from.
- step - The value to increment by when creating each element. This parameter is optional. If it is omitted, a value of 1 is assumed. A

cycle

Fill an array with a series of values, repeating as necessary.

Equivalent to:	itertools.cycle(itertools.count(start, len(array)))
or	itertools.cycle(itertools.count(start, len(array), step))

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
cycle(array, start, stop, step)
```

- array - The output array.
- start - The numeric value to start from.
- stop - The value at which to stop incrementing. If stop is less than start, cycle will count down.
- step - The value to increment by when creating each element. This parameter is optional. If it is omitted, a value of 1 is assumed. The

dropwhile

Select values from an array starting from where a selected criteria fails and proceeding to the end.

Equivalent to:	itertools.dropwhile(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = dropwhile(opstr, array, outparray, param)
result = dropwhile(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**

  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

findindex

Calculate findindex over the values in an array.

Equivalent to:	[x for x,y in enumerate(array) if y > param][0]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = findindex(opstr, array, param)
result = findindex(opstr, array, param, maxlen=y)
result = findindex(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - The resulting index. This will be negative if no match was found.

findindices

Searches an array for the array indices which meet the specified criteria and writes the results to a second array. Also returns the number of matches found.

Equivalent to:	[x for x,y in enumerate(inparray) if y == param]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = findindices(opstr, array, arrayout, param)
result = findindices(opstr, array, arrayout, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- arrayout - The output array. This must be an integer array of array type 'q' (signed long long).
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer indicating the number of matches found.

repeat

Fill an array with a specified value.

Equivalent to:	itertools.repeat(value)

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
repeat(array, value)
```

- array - The output array.

takewhile

Select values from an array starting from the beginning and stopping when the criteria fails.

Equivalent to:	itertools.takewhile(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = takewhile(opstr, array, outparray, param)
result = takewhile(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

arraylimits attributes

A set of attributes are provided representing the platform specific maximum and minimum numerical values for each array type. These attributes are part of the "arraylimits" module.

Array integer sizes may differ on 32 versus 64 bit versions, plus other platform characteristics may also produce differences.

Array Type Code	Description	Min Value	Max Value
b	signed char	b_min	b_max
B	unsigned char	B_min	B_max
h	signed short	h_min	h_max
H	unsigned short	H_min	H_max
i	signed int	i_min	i_max
I	unsigned int	I_min	I_max
l	signed long	l_min	l_max
L	unsigned long	L_min	L_max

q	signed long long	q_min	q_max
Q	unsigned long long	Q_min	Q_max
f	float	f_min	f_max
d	double	d_min	d_max

example:

```
import arrayfunc
from arrayfunc import arraylimits

arrayfunc.arraylimits.b_min
==> -128
arrayfunc.arraylimits.b_max
==> 127
arrayfunc.arraylimits.f_min
==> -3.4028234663852886e+38
arrayfunc.arraylimits.f_max
==> 3.4028234663852886e+38
```

Mathematical Functions

Description

Mathematical functions provide similar functionality to the functions of the same name in the standard library "math" and "operator" modules, but operate over whole arrays instead of on a single value.

Mathematical functions can accept a variety of different combinations of array and numerical parameters. Each function will automatically detect the category of parameter and adjust its behaviour accordingly.

Output can be either into a separate output array, or in-place (into the original array) if no output array is provided.

Parameter Forms

This example will subtract 10 from each element of array 'x', replacing the original data.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.sub(x, 10)
```

This example will do the same, but place the results into array 'z', leaving the original array unchanged.:

```
x = array.array('b', [20,21,22,23,24,25])
z = array.array('b', [0] * len(x))
arrayfunc.sub(x, 10, z)
```

This is similar to the first one, but performs the calculation of '10 - x' instead of 'x - 10'.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.sub(10, x)
```

This example takes each element of array 'x', adds the corresponding element of array 'y', and puts the result in array 'z'.:

```
x = array.array('b', [20,21,22,23,24,25])
y = array.array('b', [10,5,55,42,42,0])
z = array.array('b', [0] * len(x))
arrayfunc.add(x, y, z)
```

Parameter Type Consistency

Unless otherwise noted, all array and numeric parameters must be of the same type when calling a mathematical function. That is, you may not mix integer and floating point, or different integer sizes in the same calculation. Failing to use consistent parameters will result in an exception being raised.

Using Less than the Entire Array

If the size of the array is larger than the desired length of the calculation, it may be limited to the first part of the array by using the 'maxlen' parameter. In the following example only the first 3 array elements will be operated on, with the following ones left unchanged.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.add(x, 10, maxlen=3)
```

Suppressing or Ignoring Math Errors

Functions can be made to ignore some mathematical errors (e.g. integer overflow) by setting the 'matherrors' keyword parameter to True.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.add(x, 235, matherrors=True)
```

However, not all math errors can be suppressed, only those which would not otherwise cause a fatal error (e.g. division by zero).

Ignoring errors may be desirable if the side effect (e.g. the result of an integer overflow) is the intended effect, or for reasons of a minor performance improvement in some cases. Note that any such performance improvement will vary greatly depending upon the specific function and array type. Benchmark your calculation before deciding if this is worth while.

Differences with Native Python

In many cases the Python 'math' module functions are thin wrappers around the underlying C library, as is 'arrayfunc'.

However, in some cases 'arrayfunc' will not produce exactly the same result as Python. There are several reasons for this, the primary one being that arrayfunc operates on different underlying data types. Specifically, arrayfunc uses the platform's native integer and floating point types as exposed by the array module. For example, Python integers are of arbitrary size and can never overflow (Python simply expands the word size indefinitely), while arrayfunc integers will overflow the same as they would with programs written in C.

Think of arrayfunc as exposing C style semantics in a form convenient to use in Python. Some convenience which Python provides (e.g. no limit to the size of integers) is traded off for large performance increases.

However, Arrayfunc does implement the mod or '%' operator in a manner which is compatible with Python, not 'C'. The C method will produce mathematically incorrect answers under some ranges of values (as will many other programming languages as well as some popular spreadsheets which use the C compiler

without correction). Python implements this in a mathematically correct manner in all cases, and Arrayfunc follows suit.

Arrayfunc diverges from Python in the following areas:

- The handling of non-finite floating point values such as 'NaN' (not-a-number) and +/-Inf in calculations may not always be compatible.
- The 'floor' function will return a floating point value when floating point arrays are used, rather than an integer. This is necessary to maintain compatibility with the array parameters.
- Floordiv does not behave the same as '//' when working with infinity. When dividing positive or negative infinity by any number, the arrayfunc version of floordiv will return +/- infinity, while the Python '//' operator will return 'NaN' (not-a-number) in each case.
- Binary operations such as shift and invert will operate according to their native array data types, which may differ from Python's own integer implementation. This is necessary because the array integer is of fixed size (Python integers can be infinitely large) and has both signed and unsigned types (Python integers are signed only).
- "Mod" does not behave exactly as "%" does for floating point. X % inf and x % -inf will return nan rather than +/- inf.
- The type of exception raised when an error is encountered in Python versus arrayfunc may not be the same in all cases.

Other Notes

- Ldexp only accepts an integer number as the second parameter, not an array.
- Math.pow is not implemented because it duplicates the operator pow (and the names would collide in arrayfunc).
- Fma is not part of the Python standard library, but has been offered here as an additional feature.

Mathematical operator functions

*abs_*

Calculate abs_ over the values in an array.

Equivalent to:	[abs(x) for x in array1]

Array types supported:	b, h, i, l, q, f, d
Exceptions raised:	OverflowError

Call formats:

```
abs_(array1)
abs_(array1, outparray)
abs_(array1, maxlen=y)
abs_(array1, matherrors=False))
abs_(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

add

Calculate add over the values in an array.

Equivalent to:	[x + param for x in array1]
or	[param + y for y in array2]
or	[x + y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
add(array1, param)
add(array1, param, outparray)
add(param, array1)
add(param, array1, outparray)
add(array1, array2)
add(array1, array2, outparray)
add(array1, param, maxlen=y)
add(array1, param, matherrors=False)
add(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

floordiv

Calculate floordiv over the values in an array.

Equivalent to:	[x // param for x in array1]
or	[param // y for y in array2]
or	[x // y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
floordiv(array1, param)
floordiv(array1, param, outparray)
floordiv(param, array1)
floordiv(param, array1, outparray)
floordiv(array1, array2)
floordiv(array1, array2, outparray)
floordiv(array1, param, maxlen=y)
floordiv(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

mod

Calculate mod over the values in an array.

Equivalent to:	[x % param for x in array1]
or	[param % y for y in array2]
or	[x % y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
mod(array1, param)
mod(array1, param, outparray)
mod(param, array1)
mod(param, array1, outparray)
mod(array1, array2)
mod(array1, array2, outparray)
mod(array1, param, maxlen=y)
mod(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.

- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

mul

Calculate mul over the values in an array.

Equivalent to:	[x * param for x in array1]
or	[param * y for y in array2]
or	[x * y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
mul(array1, param)
mul(array1, param, outparray)
mul(param, array1)
mul(param, array1, outparray)
mul(array1, array2)
mul(array1, array2, outparray)
mul(array1, param, maxlen=y)
mul(array1, param, matherrors=False)
mul(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

neg

Calculate neg over the values in an array.

Equivalent to:	[-x for x in array1]

Array types supported:	b, h, i, l, q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
neg(array1)
neg(array1, outparray)
neg(array1, maxlen=y)
neg(array1, matherrors=False))
neg(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

pow

Calculate pow over the values in an array.

Equivalent to:	[x ** param for x in array1]
or	[param ** y for y in array2]
or	[x ** y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
pow(array1, param)
pow(array1, param, outparray)
pow(param, array1)
pow(param, array1, outparray)
pow(array1, array2)
pow(array1, array2, outparray)
pow(array1, param, maxlen=y)
pow(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sub

Calculate sub over the values in an array.

Equivalent to:	[x - param for x in array1]
or	[param - y for y in array2]
or	[x - y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
sub(array1, param)
sub(array1, param, outparray)
sub(param, array1)
sub(param, array1, outparray)
sub(array1, array2)
sub(array1, array2, outparray)
sub(array1, param, maxlen=y)
sub(array1, param, matherrors=False)
sub(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

truediv

Calculate truediv over the values in an array.

Equivalent to:	[x / param for x in array1]
or	[param / y for y in array2]
or	[x / y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
truediv(array1, param)
truediv(array1, param, outparray)
truediv(param, array1)
```

```
truediv(param, array1, outparray)
truediv(array1, array2)
truediv(array1, array2, outparray)
truediv(array1, param, maxlen=y)
truediv(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Comparison operator functions

eq

Calculate eq over the values in an array.

Equivalent to:	all([x == param for x in array1])
or	all([param == x for x in array1])
or	all([x == y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = eq(array1, param)
result = eq(param, array1)
result = eq(array1, array2)
result = eq(array1, param, maxlen=y)
result = eq(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).

- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

ge

Calculate ge over the values in an array.

Equivalent to:	all([x >= param for x in array1])
or	all([param >= x for x in array1])
or	all([x >= y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = ge(array1, param)
result = ge(param, array1)
result = ge(array1, array2)
result = ge(array1, param, maxlen=y)
result = ge(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

gt

Calculate gt over the values in an array.

Equivalent to:	all([x > param for x in array1])
or	all([param > x for x in array1])
or	all([x > y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = gt(array1, param)
result = gt(param, array1)
result = gt(array1, array2)
```

```
result = gt(array1, param, maxlen=y)
result = gt(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

le

Calculate le over the values in an array.

Equivalent to:	all([x <= param for x in array1])
or	all([param <= x for x in array1])
or	all([x <= y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = le(array1, param)
result = le(param, array1)
result = le(array1, array2)
result = le(array1, param, maxlen=y)
result = le(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

lt

Calculate lt over the values in an array.

Equivalent to:	all([x < param for x in array1])
or	all([param < x for x in array1])
or	all([x < y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = lt(array1, param)
result = lt(param, array1)
result = lt(array1, array2)
result = lt(array1, param, maxlen=y)
result = lt(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

ne

Calculate ne over the values in an array.

Equivalent to:	all([x != param for x in array1])
or	all([param != x for x in array1])
or	all([x != y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = ne(array1, param)
result = ne(param, array1)
result = ne(array1, array2)
result = ne(array1, param, maxlen=y)
result = ne(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.

- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

Bitwise operator functions

*and_*

Calculate and_ over the values in an array.

Equivalent to:	[x x & y param for x in array1]
or	[param x & y x for x in array1]
or	[x x & y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
and_(array1, param)
and_(array1, param, outparray)
and_(param, array1)
and_(param, array1, outparray)
and_(array1, array2)
and_(array1, array2, outparray)
and_(array1, param, maxlen=y)
and_(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

invert

Calculate invert over the values in an array.

Equivalent to:	[~x for x in array1]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
invert(array1)
invert(array1, outparray)
invert(array1, maxlen=y)
invert(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

lshift

Calculate lshift over the values in an array.

Equivalent to:	[x x << y param for x in array1]
or	[param x << y x for x in array1]
or	[x x << y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
lshift(array1, param)
lshift(array1, param, outparray)
lshift(param, array1)
lshift(param, array1, outparray)
lshift(array1, array2)
lshift(array1, array2, outparray)
lshift(array1, param, maxlen=y)
lshift(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

*or_*

Calculate or_ over the values in an array.

Equivalent to:	[x x | y param for x in array1]
or	[param x | y x for x in array1]
or	[x x | y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
or_(array1, param)
or_(array1, param, outparray)
or_(param, array1)
or_(param, array1, outparray)
or_(array1, array2)
or_(array1, array2, outparray)
or_(array1, param, maxlen=y)
or_(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

rshift

Calculate rshift over the values in an array.

Equivalent to:	[x x >> y param for x in array1]
or	[param x >> y x for x in array1]
or	[x x >> y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
rshift(array1, param)
rshift(array1, param, outparray)
rshift(param, array1)
rshift(param, array1, outparray)
rshift(array1, array2)
rshift(array1, array2, outparray)
rshift(array1, param, maxlen=y)
rshift(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

xor

Calculate xor over the values in an array.

Equivalent to:	[x x ^ y param for x in array1]
or	[param x ^ y x for x in array1]
or	[x x ^ y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
xor(array1, param)
xor(array1, param, outparray)
xor(param, array1)
xor(param, array1, outparray)
xor(array1, array2)
xor(array1, array2, outparray)
xor(array1, param, maxlen=y)
xor(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Power and logarithmic functions

exp

Calculate exp over the values in an array.

Equivalent to:	[math.exp(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
exp(array1)
exp(array1, outparray)
exp(array1, maxlen=y)
exp(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

expm1

Calculate expm1 over the values in an array.

Equivalent to:	[math.expm1(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
expm1(array1)
expm1(array1, outparray)
expm1(array1, maxlen=y)
expm1(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- matherrors - If true, arithmetic error checking is disabled. The default is false.

log

Calculate log over the values in an array.

Equivalent to:	[math.log(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log(array1)
log(array1, outparray)
log(array1, maxlen=y)
log(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log10

Calculate log10 over the values in an array.

Equivalent to:	[math.log10(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log10(array1)
log10(array1, outparray)
log10(array1, maxlen=y)
log10(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log1p

Calculate log1p over the values in an array.

Equivalent to:	[math.log1p(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log1p(array1)
log1p(array1, outparray)
log1p(array1, maxlen=y)
log1p(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log2

Calculate log2 over the values in an array.

Equivalent to:	[math.log2(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log2(array1)
log2(array1, outparray)
log2(array1, maxlen=y)
log2(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sqrt

Calculate sqrt over the values in an array.

Equivalent to:	[math.sqrt(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sqrt(array1)
sqrt(array1, outparray)
sqrt(array1, maxlen=y)
sqrt(array1, matherrors=False))
sqrt(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Hyperbolic functions

acosh

Calculate acosh over the values in an array.

Equivalent to:	[math.acosh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
acosh(array1)
acosh(array1, outparray)
acosh(array1, maxlen=y)
acosh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

asinh

Calculate asinh over the values in an array.

Equivalent to:	[math.asinh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
asinh(array1)
asinh(array1, outparray)
asinh(array1, maxlen=y)
asinh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atanh

Calculate atanh over the values in an array.

Equivalent to:	[math.atanh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atanh(array1)
atanh(array1, outparray)
atanh(array1, maxlen=y)
atanh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

cosh

Calculate cosh over the values in an array.

Equivalent to:	[math.cosh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
cosh(array1)
cosh(array1, outparray)
```

```
cosh(array1, maxlen=y)
cosh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sinh

Calculate sinh over the values in an array.

Equivalent to:	[math.sinh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sinh(array1)
sinh(array1, outparray)
sinh(array1, maxlen=y)
sinh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

tanh

Calculate tanh over the values in an array.

Equivalent to:	[math.tanh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
tanh(array1)
tanh(array1, outparray)
tanh(array1, maxlen=y)
tanh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Trigonometric functions

acos

Calculate acos over the values in an array.

Equivalent to:	[math.acos(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
acos(array1)
acos(array1, outparray)
acos(array1, maxlen=y)
acos(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

asin

Calculate asin over the values in an array.

Equivalent to:	[math.asin(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
asin(array1)
asin(array1, outparray)
asin(array1, maxlen=y)
asin(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.

- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atan

Calculate atan over the values in an array.

Equivalent to:	[math.atan(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atan(array1)
atan(array1, outparray)
atan(array1, maxlen=y)
atan(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atan2

Calculate atan2 over the values in an array.

Equivalent to:	[atan2(x, param) for x in array1]
or	[atan2(param, x) for x in array1]
or	[atan2(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atan2(array1, param)
atan2(array1, param, outparray)
atan2(param, array1)
atan2(param, array1, outparray)
atan2(array1, array2)
atan2(array1, array2, outparray)
atan2(array1, param, maxlen=y)
atan2(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

cos

Calculate cos over the values in an array.

Equivalent to:	[math.cos(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
cos(array1)
cos(array1, outparray)
cos(array1, maxlen=y)
cos(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

hypot

Calculate hypot over the values in an array.

Equivalent to:	[hypot(x, param) for x in array1]
or	[hypot(param, x) for x in array1]
or	[hypot(x, y) for x, y in zip(array1, array2)]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
hypot(array1, param)
hypot(array1, param, outparray)
hypot(param, array1)
hypot(param, array1, outparray)
```

```
hypot(array1, array2)
hypot(array1, array2, outparray)
hypot(array1, param, maxlen=y)
hypot(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sin

Calculate sin over the values in an array.

Equivalent to:	[math.sin(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sin(array1)
sin(array1, outparray)
sin(array1, maxlen=y)
sin(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

tan

Calculate tan over the values in an array.

Equivalent to:	[math.tan(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
tan(array1)
tan(array1, outparray)
tan(array1, maxlen=y)
tan(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Angular conversion

degrees

Calculate degrees over the values in an array.

Equivalent to:	[math.degrees(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
degrees(array1)
degrees(array1, outparray)
degrees(array1, maxlen=y)
degrees(array1, matherrors=False))
degrees(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

radians

Calculate radians over the values in an array.

Equivalent to:	[math.radians(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
radians(array1)
radians(array1, outparray)
radians(array1, maxlen=y)
radians(array1, matherrors=False))
radians(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Number-theoretic and representation functions

ceil

Calculate ceil over the values in an array.

Equivalent to:	[math.ceil(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
ceil(array1)
ceil(array1, outparray)
ceil(array1, maxlen=y)
ceil(array1, matherrors=False))
ceil(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

copysign

Calculate copysign over the values in an array.

Equivalent to:	[copysign(x, param) for x in array1]
or	[copysign(param, x) for x in array1]
or	[copysign(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
copysign(array1, param)
copysign(array1, param, outparray)
copysign(param, array1)
copysign(param, array1, outparray)
copysign(array1, array2)
copysign(array1, array2, outparray)
copysign(array1, param, maxlen=y)
copysign(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

fabs

Calculate fabs over the values in an array.

Equivalent to:	[math.fabs(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fabs(array1)
fabs(array1, outparray)
fabs(array1, maxlen=y)
fabs(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

factorial

Calculate factorial over the values in an array.

Equivalent to:	[math.factorial(x) for x in array1]
Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	OverflowError

Call formats:

```
factorial(array1)
factorial(array1, outparray)
factorial(array1, maxlen=y)
factorial(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

floor

Calculate floor over the values in an array.

Equivalent to:	[math.floor(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
floor(array1)
floor(array1, outparray)
floor(array1, maxlen=y)
floor(array1, matherrors=False))
floor(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

fmod

Calculate fmod over the values in an array.

Equivalent to:	[fmod(x, param) for x in array1]
or	[fmod(param, x) for x in array1]
or	[fmod(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fmod(array1, param)
fmod(array1, param, outparray)
fmod(param, array1)
fmod(param, array1, outparray)
fmod(array1, array2)
fmod(array1, array2, outparray)
fmod(array1, param, maxlen=y)
fmod(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

isfinite

Calculate isfinite over the values in an array.

Equivalent to:	all([isfinite(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isfinite(array1)
result = isfinite(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- result - A boolean value corresponding to the result of all the comparison operations. If all of the comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

isinf

Calculate isinf over the values in an array.

Equivalent to:	any([isinf(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isinf(array1)
result = isinf(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - A boolean value corresponding to the result of all the comparison operations. If at least one comparison operation results in true, the return value will be true. If none of them result in true, the return value will be false.

isnan

Calculate isnan over the values in an array.

Equivalent to:	any([isnan(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isnan(array1)
result = isnan(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - A boolean value corresponding to the result of all the comparison operations. If at least one comparison operation results in true, the return value will be true. If none of them result in true, the return value will be false.

ldexp

Calculate ldexp over the values in an array.

Equivalent to:	math.ldexp(x, y)
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
ldexp(array1, exp)
ldexp(array1, exp, outparray)
ldexp(array1, exp, maxlen=y)
ldexp(array1, exp, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- exp - The exponent to apply to the input array. This must be an integer.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

trunc

Calculate trunc over the values in an array.

Equivalent to:	[math.trunc(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
trunc(array1)
trunc(array1, outparray)
trunc(array1, maxlen=y)
trunc(array1, matherrors=False))
trunc(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Special functions

erf

Calculate erf over the values in an array.

Equivalent to:	[math.erf(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
erf(array1)
erf(array1, outparray)
erf(array1, maxlen=y)
erf(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

erfc

Calculate erfc over the values in an array.

Equivalent to:	[math.erfc(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
erfc(array1)
erfc(array1, outparray)
erfc(array1, maxlen=y)
erfc(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

gamma

Calculate gamma over the values in an array.

Equivalent to:	[math.gamma(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
gamma(array1)
gamma(array1, outparray)
gamma(array1, maxlen=y)
gamma(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

lgamma

Calculate lgamma over the values in an array.

Equivalent to:	[math.lgamma(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
lgamma(array1)
lgamma(array1, outparray)
lgamma(array1, maxlen=y)
lgamma(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Additional functions

fma

Calculate fma over the values in an array.

Equivalent to:	[(x * param2 + param3) for x in array1]
or	[(x * y + param3) for x,y in zip(array1, array2)]

or	[(x * param2 + z) for x,z in zip(array1, array3)]
or	[(x * y + z) for x,y,z in zip(array1, array2, array3)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fma(array1, array2, array3)
fma(array1, array2, array3, outparray)
fma(array1, array2, param3)
fma(array1, array2, param3, outparray)
fma(array1, param2, array3)
fma(array1, param2, array3, outparray)
fma(array1, param2, param3)
fma(array1, param2, param3, outparray)
fma(array1, array2, array3, maxlen=y)
fma(array1, array2, array3, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- **array2 - A second input data array. Each element in this array is**
  applied to the corresponding element in the first array.
- **param2 - A non-array numeric parameter which may be used in place**
  of array2.
- array3 - A third input data array. Each element in this array is applied to the corresponding element in the first array.
- **param3 - A non-array numeric parameter which may be used in place**
  of array3.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Option Flags and Parameters

Arithmetic Overflow Control

Many functions allow integer overflow detection to be turned off if desired. See the list of operators for which operators this applies to.

Integer overflow is when a number becomes too large to fit within the specified word size for that array data type. For example, an unsigned char has a range of 0 to 255. When a calculation overflows, it "wraps around" one or more times and produces an arithmetically invalid result.

If it is known in advance that overflow cannot occur (due to the size of the numbers), or if overflow is a desired side effect, then overflow checking may be disabled via the "matherrors" parameter. Setting

"matherrors" to true will *disable* overflow checking, while setting it to false will *enable* overflow checking. Checking is enabled by default, including when the "matherrors" parameter is not specified.

Disabling overflow checking can significantly increase the speed of calculation, with the amount of improvement depending on the type of calculation being performed and the data type used.

Using Only Part of an Array

The array math functions only use existing arrays that the user provides and do not create new arrays or resize existing ones. The reason for this is that when very large arrays are being used, continually allocating and de-allocating arrays can take too much time, plus this may result in problems controlling how much memory is used.

Since the filter functions (or other data sources) may not use all of an output array, and the result may vary depending on the data, most functions provide an optional keyword parameter which limits the functions to part of the array. The "maxlen" parameter specifies the maximum number of array elements to use, starting from the beginning of the array.

For example, specifying a "maxlen" of 10 for a 20 element array will limit a function to using only the first 10 array elements and ignoring the rest of the array.

If the array length limit value is zero, negative, or greater than the actual size of the array, the length limit will be ignored and the entire array used. The default is to use the entire array.

SIMD Control

SIMD (Single Instruction Multiple Data) is a set of CPU features which allow multiple operations to take place in parallel. Some, but not all, functions will make use of these instructions to speed up execution.

Those functions which do support SIMD features will automatically make use of them by default unless this feature is disabled. There is normally no reason to disable SIMD, but should there be hardware related problems the function can be forced to fall back to conventional execution mode.

If the optional parameter "nosimd" is set to true ("nosimd=True"), SIMD execution will be disabled. The default is "False".

To repeat, there is normally no reason to wish to disable SIMD.

See the documentation section on SIMD support has more detail.

Data Types

Array Types

The following array types from the Python standard library are supported.

Array Type Code	Description
b	signed char
B	unsigned char
h	signed short
H	unsigned short
i	signed int
I	unsigned int
l	signed long

L	unsigned long
q	signed long long
Q	unsigned long long
f	float
d	double

Numeric Parameter Types

Python Type	Description
integer	Integral values such as 0, 1, 100, -99, etc.
floating point	Real numbers such as 0.0, 1.93, 3.1417, -5693.0, etc.

The numeric type must be compatible with the array type code.

The 'L' and 'Q' type parameters cannot be checked for integer overflow due to a mismatch between Python and 'C' language numeric limits.

Maximum Array Size

Arrays are limited to no more than the number of elements defined by the Python C API constant Py_ssize_t. The size of this will depend on your platform characteristics. However, it will normally allow for arrays larger than can be contained in memory for most computers.

When creating very large arrays, it is recommended to consider using itertools.repeat as an initializer or to use array.extend or array.append to add to an array rather than using a list as an initializer. Lists use much more memory than arrays (even for the same data type), and it is easy to run out of memory if you are not careful when creating very large arrays from lists.

Platform Compiler Support

Beginning with version 2.0 of ArrayFunc, versions compiled with the Microsoft MSVS compiler now has feature parity with the GCC version. This change is due to the Microsoft C compiler now supporting a new enough version of the 'C' standard.

Integer Error Checking

Error checking in integer operators is conducted as follows:

Error Categories

Operation	Result out of range	Divide by zero	Negate max. negative signed int	Parameter is negative
Addition (+)	X			
Subtraction (-)	X			
Modulus (%)		X	X	
Multiplication (*)	X			
Division (/, //)		X	X	
Negation (-)			X	
Absolute Value			X	

Factorial	X			X
Power (**)	X			X

- Negation of the maximum negative signed in (the most negative integer for that array type) can be caused by negation, absolute value, division, and modulus operations. Since signed integers do not have a symmetrical range (e.g. -128 to 127 for 8 bit sizes) anything which attempts to convert (in this example) -128 to +128 would cause an overflow back to -128.
- The factorial of negative numbers is undefined.
- Powers are not calculated for integers raised to negative powers, as integer arrays cannot contain fractional results.

Disabling Integer Division by Zero Checks

Division by zero cannot be disabled for integer division or modulus operations. Division by zero could cause seg faults (crashes), so this option is ignored for these functions.

Floating Point NaN and Infinity

Floating point numbers include three special values, NaN (Not a Number), and negative and positive infinity. Arrayfunc uses the platform C compiler to create executable code. Some compilers may produce different results than other compilers under certain conditions when operating on NaN and infinity values. In addition, the Arrayfunc results may differ from those in native Python on some platforms when using NaN and infinity as inputs.

However, since using NaN and infinity as numeric inputs is not a common operation, this is unlikely to be a serious problem when writing cross platform code in most cases.

Exceptions

Exceptions - General

The following exceptions apply to most functions.

Exception type	Text	Description
ArithmeticError	arithmetic error in calculation.	An arithmetic error occurred in a calculation.
ZeroDivisionError	zero division error in calculation.	A calculation attempted to divide by zero.
IndexError	array length error.	One or more arrays has an invalid length (e.g a length of zero).
IndexError	input array length error.	The input array has an invalid length.
IndexError	output length error.	The output array has an invalid length.
IndexError	array length mismatch.	Two or more arrays which are expected to be of equal length are not.
OverflowError	arithmetic overflow in calculation.	An arithmetic integer overflow occurred in a calculation.
OverflowError	arithmetic overflow in parameter.	The size or range of a non-array parameter was not compatible with the array parameters.

TypeError	array and parameter type mismatch.	A non-array parameter data type was not compatible with the array parameters.
TypeError	array type mismatch.	An array parameter is not compatible with another array parameter. For most functions, both arrays must be of the same type.
TypeError	unknown array type.	The array type is unknown.
TypeError	array.array expected.	A non-array parameter was found where an array parameter was expected.
ValueError	operator not valid for this function.	An operator parameter used was not valid for this function.
ValueError	operator not valid for this platform.	The operator used is not supported on this platform.
TypeError	parameter error.	An unspecified error occurred when parsing the parameters.
TypeError	parameter missing.	An expected parameter was missing.
ValueError	parameter not valid for this operation.	A value is not valid for this operation. E.g. attempting to perform a factorial on a negative number.
IndexError	selector length error.	The selector array length is incorrect.
ValueError	conversion not valid for this type.	The conversion attempted was invalid.
ValueError	cannot convert float NaN to integer.	Cannot convert NaN (Not A Number) floating point value in the input array to integer.
TypeError	output array type invalid.	The output array type is invalid.

Platform Oddities

As most operators are implemented using native behaviour, details of some operations may depend on the CPU architecture.

Lshift and rshift will exhibit a behaviour that depends on the CPU type whether it is 32 or 64 bit, and array size.

For 32 bit x86 systems, if the array word size is 32 bits or less, the shift is masked to 5 bits. That is, shift amounts greater than 32 will "roll over", repeating smaller shifts.

On 64 bit systems, this behaviour will vary depending on whether SIMD is used or not. This, arrays which are not even multiples of SIMD register sizes may exibit different behaviour at different array indexes (depending on whether SIMD or non-SIMD instructions were used for those parts of the array).

ARM does not display this roll-over behaviour, and so may give different results than x86. However, negative shift values may result in the shift operation being conducted in the opposite direction (e.g. right shift instead of left shift).

The conclusion is that bit shift operations which use a shift amount which is not in the range of 0 to "maximum number" may produce undefined results. So valid bit shift amounts should be 0 to 7, 0 to 15, 0 to 31 and 0 to 63, depending on the array type.

SIMD Support

General

SIMD (Single Instruction Multiple Data) is a set of CPU features which allow multiple operations to take place in parallel. Some, but not all, functions will make use of these instructions to speed up execution.

Those functions which do support SIMD features will automatically make use of them by default unless this feature is disabled. There is normally no reason to disable SIMD, but should there be hardware related problems the function can be forced to fall back to conventional execution mode.

Platform Support

SIMD instructions are presently supported only on the following:

- 64 bit x86 (i.e. AMD64) using GCC.
- 32 bit ARMv7 using GCC (tested on Raspberry Pi 3).
- 64 bit ARMv8 AARCH64 using GCC (tested on Raspberry Pi 3).

Other compilers or platforms will still run the same functions and should produce the same results, but they will not benefit from SIMD acceleration.

However, non-SIMD functions will still be much faster standard Python code. See the performance benchmarks to see what the relative speed differences are. With wider data types (e.g. double precision floating point) SIMD provides only marginal speed ups anyway.

Raspberry Pi 32 versus 64 bit

The Raspberry Pi uses an ARM CPU. This can operate in 32 or 64 bit mode. When in 32 bit mode, the Raspberry Pi 3 operates in ARMv7 mode. This has 64 bit ARM NEON SIMD vectors.

When in 64 bit mode, it acts as an ARMv8, with AARCH64 128 bit ARM NEON SIMD vectors.

The Raspbian Linux OS is 32 bit mode only. Other distros such as Ubuntu offer 64 bit versions.

The "setup.py" file uses platform detection code to determine which ARM CPU and mode it is running on. Due to the availability of hardware for testing, this code is tailored to the Raspberry Pi 3 and the operating systems listed. This code then selects the appropriate compiler arguments to pass to the setup routines to tell the compiler what mode to compile for.

If other ARM platforms are used which have different platform signatures or which require different compiler arguments, the "setup.py" file may need to be modified in order to use SIMD acceleration.

However, the straight 'C' code should still compile and run, and still provide performance many times faster than when using native Python.

Data Type Support

x86-64

The following table shows which array data types are supported by x86-64 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X	X	X					X	X
aany	X	X	X	X	X	X					X	X
abs_	X		X		X							
add	X		X		X						X	X

amax	X	X	X	X	X	X					X	X
amin	X	X	X	X	X	X					X	X
and_	X	X	X	X	X	X						
asum											X	X
ceil											X	X
degrees											X	X
eq	X	X	X	X	X	X					X	X
findindex	X	X	X	X	X	X					X	X
floor											X	X
ge	X	X	X	X	X	X					X	X
gt	X	X	X	X	X	X					X	X
invert	X	X	X	X	X	X						
le	X	X	X	X	X	X					X	X
lshift			X	X	X	X						
lt	X	X	X	X	X	X					X	X
ne	X	X	X	X	X	X					X	X
neg	X		X		X							
or_	X	X	X	X	X	X						
radians											X	X
rshift				X		X						
sqrt											X	X
sub	X		X		X						X	X
trunc											X	X
xor	X	X	X	X	X	X						

ARMv7

The following table shows which array data types are supported by ARMv7 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X								
aany	X	X	X	X								
abs_	X		X		X							
add	X	X	X	X								
amax	X	X	X	X	X	X					X	
amin	X	X	X	X	X	X					X	
and_	X	X	X	X	X	X						
degrees											X	
eq	X	X	X	X								
findindex	X	X	X	X								

ge	X	X	X	X								
gt	X	X	X	X								
invert	X	X	X	X	X	X						
le	X	X	X	X								
lshift	X	X	X	X	X	X						
lt	X	X	X	X								
mul	X	X	X	X								
ne	X	X	X	X								
neg	X		X		X							
or_	X	X	X	X	X	X						
radians											X	
rshift	X	X	X	X	X	X						
sub	X	X	X	X								
xor	X	X	X	X	X	X						

ARMv8 AARCH64

The following table shows which array data types are supported by ARMv8 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X	X	X					X	
aany	X	X	X	X	X	X					X	
abs_	X		X		X						X	
add	X	X	X	X	X	X					X	
amax	X	X	X	X	X	X					X	
amin	X	X	X	X	X	X					X	
and_	X	X	X	X	X	X						
degrees											X	
eq	X	X	X	X	X	X					X	
findindex	X	X	X	X	X	X					X	
ge	X	X	X	X	X	X					X	
gt	X	X	X	X	X	X					X	
invert	X	X	X	X	X	X						
le	X	X	X	X	X	X					X	
lshift	X	X	X	X	X	X						
lt	X	X	X	X	X	X					X	
mul	X	X	X	X	X	X					X	
ne	X	X	X	X	X	X					X	
neg	X		X		X						X	
or_	X	X	X	X	X	X						

radians											X	
rshift	X	X	X	X	X	X						
sub	X	X	X	X	X	X					X	
xor	X	X	X	X	X	X						

SIMD Support Attributes

There is an attribute which can be tested to detect if ArrayFunc is compiled with SIMD support and if the current hardware supports the required SIMD level.

arrayfunc.simdsupport.hassimd

The attribute "hassimd" will be True if the module supports SIMD.

example:

```
import arrayfunc
arrayfunc.simdsupport.hassimd
==> True
```

Performance

Variables affecting Performance

The purpose of the Arrayfunc module is to execute common operations faster than native Python. The relative speed will depend upon a number of factors:

- The function.
- The data type of the array.
- Function options. Turning checking off will result in faster performance.
- The data in the arrays and the parameters.
- The size of the array.
- The platform, including CPU type (e.g. x86 or ARM), operating system, and compiler.

The speeds listed below should be used as rough guidelines only. More exact results will require application specific testing. The numbers shown are the execution time of each function relative to native Python. For example, a value of '50' means that the corresponding Arrayfunc operation ran 50 times faster than the closest native Python equivalent.

Both relative performance (the speed-up as compared to Python) and absolute performance (the actual execution speed of Python and ArrayFunc) will vary significantly depending upon the compiler (which is OS platform dependent) and whether compiled to 32 or 64 bit. If your precise actual benchmark performance results matter, be sure to conduct your testing using the actual OS and compiler your final program will be deployed on. The values listed below were measured on x86-64 Linux compiled with GCC.

Note: Some more complex Arrayfunc functions do not work exactly the same way as the built-in or "itertools" Python equivalents. This means that the benchmark results should be taken as general guidelines rather than precise comparisons.

Typical Performance Readings

Default Performance

In this set of tests, all error checking was turned on and SIMD acceleration was enabled where this did not conflict with the preceding (the defaults in each case).

Relative Performance - Python Time / Arrayfunc Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	113	64	52	35	26	20	6.6	9.9	6.4	9.5	50	26
aany	47	40	25	25	13	15	5.0	4.6	4.6	4.6	23	13
afilter	116	118	116	117	117	94	118	119	116	119	118	118
amax	76	75	39	38	119	75	15	11	14	14	153	35
amin	71	69	36	38	72	109	13	14	12	14	116	59
asum	4.9	9.2	3.9	7.6	7.7	7.2	4.7	8.9	3.9	8.1	6.9	6.5
compress	34	35	37	22	30	19	38	26	38	23	31	29
count	186	206	137	180	146	116	111	111	104	99	73	74
cycle	82	72	79	78	73	52	73	50	75	51	57	59
dropwhile	228	229	116	115	116	184	160	177	172	157	182	157
findindex	206	219	86	87	56	57	17	24	17	24	68	36
findindices	21	26	21	26	21	27	22	28	21	27	28	28
repeat	124	89	126	116	112	31	102	38	101	38	90	91
takewhile	221	234	191	276	241	163	194	139	199	146	221	187
add	130	137	147	135	131	103	92	79	97	68	103	76
truediv	69	60	62	64	70	60	65	57	73	57	185	152
floordiv	31	27	32	29	33	26	31	26	32	26	156	132
mod	21	25	17	25	26	22	27	22	27	22	72	61
mul	83	96	81	121	77	60	71	38	70	38	103	82
neg	111		123		130		79		82		123	76
pow	50	49	47	45	32	54	17	50	18	49	6.3	14
sub	136	138	124	100	126	109	91	74	93	69	98	78
and_	1388	1417	887	885	410	340	107	76	101	91		
or_	1915	1782	806	866	398	334	106	89	118	100		
xor	1884	1878	820	781	409	354	113	93	121	96		
invert	2173	2339	1300	1430	603	681	147	178	192	172		
eq	886	1093	435	450	208	246	58	59	90	101	230	128
gt	921	616	482	324	266	175	58	71	90	100	162	120

ge	819	795	442	479	257	243	60	63	89	106	252	123
lt	898	613	378	318	194	170	96	99	97	95	259	143
le	996	887	476	454	276	263	100	99	93	90	247	134
ne	1004	900	488	497	287	306	93	93	86	99	244	147
lshift	190	236	916	804	411	448	91	113	121	82		
rshift	162	159	156	787	184	384	101	89	112	83		
abs_	121		116		114		81		92		227	106
acos											13	11
acosh											8.6	6.2
asin											14	13
asinh											7.3	6.8
atan											14	12
atan2											7.5	6.8
atanh											7.3	8.3
ceil											278	186
copysign											228	147
cos											15	8.2
cosh											12	7.7
degrees											164	120
erf											17	14
erfc											10	7.5
exp											19	9.3
expm1											6.7	7.1
fabs											203	148
factorial	203	251	204	245	196	211	139	127	125	129		
floor											261	197
fma											114	97
fmod											12	12
gamma											1.6	1.3
hypot											21	13
isfinite											125	120
isinf											143	131
isnan											144	136
ldexp											28	29
lgamma											10.0	6.2
log											26	9.3
log10											15	7.7

log1p											8.4	8.9
log2											22	12
radians											164	127
sin											14	8.4
sinh											5.7	5.7
sqrt											23	20
tan											5.9	5.6
tanh											6.3	6.1
trunc											254	172

Stat	Value
Average:	171
Maximum:	2339
Minimum:	1.3
Array size:	100000

Optimised Performance (with SIMD)

In this set of tests, all arithmetic error checking was disabled (not the default state) and SIMD acceleration was enabled (the normal default). Note that there may be unexpected slight differences as compared to the previous data table due to variations in test timing.

This data may be of some use when estimating if any useful performance gains can be made in your specific application by disabling error checking in order to enable SIMD operations. It is not recommended to disable math error checking without good reason.

Relative Performance with SIMD Optimisations - Python Time / Arrayfunc Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	114	64	53	35	25	20	6.7	10	5.8	9.5	48	25
aany	46	40	25	25	12	15	4.5	4.5	4.6	4.8	23	13
amax	76	75	39	38	117	76	15	16	15	14	154	35
amin	71	69	36	38	73	104	13	12	13	14	114	59
asum	8.0	13	6.5	10	11	12	7.7	15	6.4	13	27	13
findindex	206	218	86	88	55	57	16	24	17	24	68	36
add	1468	132	751	134	340	151	90	88	91	74	345	101
neg	1115		631		339		80		95		182	93
sub	1452	203	659	131	303	108	89	72	88	68	433	103
and_	1394	1420	887	888	401	339	107	76	102	89		
or_	1908	1784	806	862	402	337	106	89	117	100		

xor	1845	1815	819	808	409	358	111	92	123	96		
invert	2174	2373	1268	1428	603	680	147	176	192	172		
eq	881	1084	433	440	211	254	59	58	90	104	228	130
gt	878	617	488	326	264	177	61	71	82	102	154	119
ge	821	799	452	479	265	242	60	62	83	107	247	123
lt	906	612	379	318	194	170	89	102	92	94	235	143
le	966	875	484	459	272	263	99	98	93	92	250	133
ne	1021	916	495	461	286	306	96	90	84	101	265	152
lshift	190	236	917	805	411	448	91	113	121	82		
rshift	163	159	157	783	189	387	102	90	112	83		
abs_	1758		844		556		95		103		259	128
ceil											765	233
degrees											564	194
floor											983	276
radians											567	201
sqrt											211	91
trunc											953	224

Stat	Value
Average:	308
Maximum:	2373
Minimum:	4.5
Array size:	100000

SIMD Optimisation Effects

This set of tests shows what the effect of SIMD optimisations are for those functions which support it. SIMD optimisations are enabled by default except in a few cases where they conflict with math error checking (in which case error checking must be disabled to use them). This information may be useful in deciding which platform you wish to use to run your application. This data is primarily of interest in judging expected benchmark performance on different platforms.

Relative Performance with and without SIMD Optimisations - Optimised / SIMD Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	17	10	6.2	3.3	2.3	2.3					2.6	1.6
aany	9.3	14	8.1	5.2	2.7	3.9					2.9	1.3
amax	3.6	3.8	1.6	1.9	4.9	3.9					3.8	1.3
amin	3.1	3.7	1.8	2.1	4.2	5.1					5.5	2.9
asum											3.9	2.0

findindex	9.8	10	3.9	3.9	3.2	3.3					5.1	2.0
add	7.5		5.3		2.6						2.6	1.3
neg	9.1		3.5		1.9							
sub	7.5		5.3		2.4						3.4	1.3
and_	9.1	9.3	6.4	4.0	2.5	2.7						
or_	8.2	7.8	3.6	5.3	1.9	1.8						
xor	9.9	7.9	3.8	5.2	1.9	1.8						
invert	9.6	6.4	5.1	3.5	2.7	1.9						
eq	16	18	7.1	7.9	3.6	4.4					2.9	1.8
gt	11	10	6.1	4.1	4.1	1.7					1.9	1.1
ge	10	13	5.6	5.7	4.4	2.4					3.2	1.2
lt	8.1	7.7	4.0	3.4	2.0	2.7					2.7	1.7
le	11	11	5.1	5.0	2.8	4.1					2.7	1.6
ne	13	11	6.4	5.9	3.1	3.0					3.2	1.9
lshift			5.9	5.2	2.7	2.6						
rshift				3.8		2.7						
abs_	12		6.5		3.8							
ceil											1.8	1.3
degrees											2.4	1.6
floor											3.4	1.1
radians											2.2	1.6
sqrt											7.5	4.0
trunc											3.4	1.1

Array Size Versus Performance

The following shoes the effects of array size on a selected arrayfunc function benchmark.

As array size increases, function call overhead decreases as a proportion of total run time.

Declines in performance when the array exceeds a certain size may be related to hardware cache effects. Arrayfunc functions together with their data may be able to reside entirely in cache, but larger arrays may require repeated cache reloads. This threshold will depend upon the particular hardware being used.

Add constant to array - times faster than Python, default settings.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	2.2	1.6	1.6	1.5	1.5	1.2	1.4	1.1	1.5	1.1	1.6	1.3
100	12	11	11	11	11	8.3	10	9.4	10	9.8	11	9.3
1000	70	69	66	62	59	49	53	42	52	42	52	47
10000	123	121	127	115	120	91	108	94	110	88	97	87
100000	137	127	143	128	131	102	82	83	91	82	102	75
1000000	133	135	126	120	90	79	51	44	49	43	85	52
10000000	134	134	124	123	98	72	50	38	47	38	75	49

Xor an array by a constant - times faster than Python, default settings.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	2.2	1.8	2.0	1.8	1.8	1.4	1.8	1.5	1.7	1.3		
100	15	14	15	14	15	11	14	11	12	9.7		
1000	156	154	133	130	87	77	67	59	70	54		
10000	852	807	529	506	307	262	145	120	151	118		
100000	1877	1841	831	798	415	350	116	100	134	97		
1000000	818	763	244	241	121	102	56	49	59	48		
10000000	504	517	246	238	116	98	57	47	56	47		

Platform Effects

The platform, including CPU, OS, compiler, and compiler version can affect performance, and this influence can change significantly for different functions.

If your application requires exact performance data, then benchmark your application in the specific platform (hardware, OS, and compiler) that you will be using.

Platform support

Arrayfunc is written in 'C' and uses the standard C libraries to implement the underlying math functions. Arrayfunc has been tested on the following platforms.

OS	Bits	Compiler	Python Version Tested
Ubuntu 18.04 LTS	64 bit	GCC	3.6
Ubuntu 19.10	64 bit	GCC	3.7
Ubuntu 20.04 beta	64 bit	GCC	3.8
Debian 10	32 bit	GCC	3.7
Debian 10	64 bit	GCC	3.7
OpenSuse 15	64 bit	GCC	3.6
Centos 8	64 bit	GCC	3.6
FreeBSD 12	64 bit	LLVM	3.7
OpenBSD 6.5	64 bit	LLVM	3.6
MS Windows 10	64 bit	MS Visual Studio C 2015	3.8
Raspbian (RPi 3)	32 bit	GCC	3.7
Ubuntu 19.10 ARM	64 bit	GCC	3.7

- The Raspbian (RPi 3) tests were conducted on a Raspberry Pi 3 ARM CPU running in 32 bit mode.
- The Ubuntu ARM tests were conducted on a Raspberry Pi 3 ARM CPU running in 64 bit mode.
- All others were conducted using VMs running on x86 hardware.