ArrayFunc

Authors: Michael Griffin
Version: 4.1.0 for 2018-11-16

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.

Important Note for Upgrading to Version 4

Version 4 drops support for the amap, amapi, starmap, starmapi, and acalc functions. These have all been replaced by individual functions which perform the same calculations but in a more direct way.

The reason for this change is that it was not possible to support these functions while also providing a simple and consistent call interface. Now each function has a call interface tailored specifically for how that function works. This also provides for a more natural mix of array and numeric parameters.

This change will now allow more mathematical functions to be added in future without trying to force-fit them into a single call interface.

Version 4 also changes the parameter used to select the type of comparison operation for dropwhile, takewhile, aany, aall, findindex, and findindices. This change has been necessitated by the removal of amap and related functions. These functions however should still work in a compatible manner.

Finally, support for the "bytes" type has been dropped.

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 proceding 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
add	x + y
truediv	x / y
floordiv	x // y
mod	x % y
mul	x * y
neg	-x
pow	x**y or math.pow(x, y)
sub	x - y
abs_	abs(x)

Comparison operator functions

Function	Equivalent to
eq	x == y
gt	x > y
ge	x >= y
lt	x < y
le	x <= y
ne	x != y

Bitwise operator functions

Function	Equivalent to

and_	x & y
or_	x | y
xor	x ^ y
invert	~x
lshift	x << y
rshift	x >> y

Power and logarithmic functions

Function	Equivalent to
exp	math.exp(x)
expm1	math.expm1(x)
log	math.log(x)
log10	math.log10(x)
log1p	math.log1p(x)
log2	math.log2(x)
sqrt	math.sqrt(x)

Hyperbolic functions

Function	Equivalent to
acosh	math.acosh(x)
asinh	math.asinh(x)
atanh	math.atanh(x)
cosh	math.cosh(x)
sinh	math.sinh(x)
tanh	math.tanh(x)

Trigonometric functions

Function	Equivalent to
acos	math.acos(x)
asin	math.asin(x)
atan	math.atan(x)
atan2	math.atan2(x, y)
cos	math.cos(x)
hypot	math.hypot(x, y)
sin	math.sin(x)
tan	math.tan(x)

Angular conversion

Function	Equivalent to
degrees	math.degrees(x)
radians	math.radians(x)

Number-theoretic and representation functions

Function	Equivalent to
ceil	math.ceil(x)
copysign	math.copysign(x, y)
fabs	math.fabs(x)
factorial	math.factorial(x)
floor	math.floor(x)
fmod	math.fmod(x, y)
isfinite	math.isfinite(x)
isinf	math.isinf(x)
isnan	math.isnan(x)
ldexp	math.ldexp(x, y)
trunc	math.trunc(x)

Special functions

Function	Equivalent to
erf	math.erf(x)
erfc	math.erfc(x)
gamma	math.gamma(x)
lgamma	math.lgamma(x)

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

count

Fill an array with evenly spaced values using a start and step values. The function continues until the end of the array. The function does not check for integer overflow.

count(dataarray, start, step)

- dataarray - 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 negative step value will cause the function to count down.

example:

```
dataarray = array.array('i', [0]*10)
arrayfunc.count(dataarray, 0, 5)
==> array('i', [0, 5, 10, 15, 20, 25, 30, 35, 40, 45])
arrayfunc.count(dataarray, 99)
==> array('i', [99, 100, 101, 102, 103, 104, 105, 106, 107, 108])
arrayfunc.count(dataarray, 29, -8)
==> array('i', [29, 21, 13, 5, -3, -11, -19, -27, -35, -43])
dataarray = array.array('b', [0]*10)
arrayfunc.count(dataarray, 52, 10)
==> array('b', [52, 62, 72, 82, 92, 102, 112, 122, -124, -114])
```

cycle

Fill an array with evenly spaced values using a start, stop, and step values, and repeat until the array is filled.

cycle(dataarray, start, stop, step)

- dataarray - 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 sign is ignored and the absolute value used when incrementing.

example:

```
dataarray = array.array('i', [0]*100)
arrayfunc.cycle(dataarray, 0, 25, 5)
==> array('i', [0, 5, 10, 15, 20, 25, 0, 5, ... , 10, 15])
arrayfunc.cycle(dataarray, 5, 30)
==> array('i', [5, 6, 7, 8, 9, 10, ... 28, 29, 30, 5, ... , 24, 25, 26])
dataarray = array.array('i', [0]*10)
arrayfunc.cycle(dataarray, 10, 5, 1)
==> array('i', [10, 9, 8, 7, 6, 5, 10, 9, 8, 7])
arrayfunc.cycle(dataarray, -2, 3, 1)
==> array('i', [-2, -1, 0, 1, 2, 3, -2, -1, 0, 1])
```

repeat

Fill an array with a specified value.

repeat(dataarray, value)

- dataarray - The output array.
- value - The value to use to fill the array.

example:

```
dataarray = array.array('i', [0]*100)
arrayfunc.repeat(dataarray, 99)
==> array('i', [99, 99, 99, 99, ... , 99, 99])
```

afilter

Select values from an array based on a boolean criteria.

x = afilter(op, inparray, outparray, rparam)

x = afilter(op, inparray, outparray, rparam, maxlen=500)

- op - The arithmetic comparison operation.
- inparray - The input data array to be filtered.
- outparray - The output array.
- rparam - The 'y' parameter to be applied to 'op'.
- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('i', [0]*6)
x = arrayfunc.afilter('>', inparray, outparray, 10)
==> array('i', [33, 54, 0, 0, 0, 0])
==> x equals 2
x = arrayfunc.afilter('>', inparray, outparray, 10, maxlen=4)
==> array('i', [33, 0, 0, 0, 0, 0])
==> x equals 1
```

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 the output array, while a value of *0* causes the value to be ignored.

The input, selector, and output arrays need not be of the same length. The copy operation will be terminated when the end of the input or output array is reached. The selector array will be cycled through repeatedly as many times as necessary until the end of the input or output array is reached.

x = compress(inparray, outparray, selectorarray)

x = compress(inparray, outparray, selectorarray, maxlen=500)

- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('i', [0]*6)
selectorarray = array.array('i', [0, 1, 0, 1])
x = arrayfunc.compress(inparray, outparray, selectorarray)
==> array('i', [2, 33, -6, 0, 0, 0])
==> x equals 3
x = arrayfunc.compress(inparray, outparray, selectorarray, maxlen=4)
==> array('i', [2, 33, 0, 0, 0, 0])
==> x equals 2
```

dropwhile

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

x = dropwhile(op, inparray, outparray, rparam)

x = dropwhile(op, inparray, outparray, rparam, maxlen=500)

- op - The arithmetic comparison operation.
- inparray - The input data array to be filtered.
- outparray - The output array.
- rparam - The 'y' parameter to be applied to 'op'.
- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('i', [0]*6)
x = arrayfunc.dropwhile('<', inparray, outparray, 10)
```

```
==> array('i', [33, 54, 0, 0, 0, 0])
==> x equals 3
x = arrayfunc.dropwhile('<', inparray, outparray, 10, maxlen=5)
==> array('i', [33, 54, 0, 0, 0, 0])
==> x equals 2
```

takewhile

Like dropwhile, but starts from the beginning and stops when the criteria fails.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('i', [0]*6)
x = arrayfunc.takewhile('<', inparray, outparray, 10)
==> array('i', [1, 2, 5, 0, 0, 0])
==> x equals 3
x = arrayfunc.takewhile('<', inparray, outparray, 10, maxlen=2)
==> array('i', [1, 2, 0, 0, 0, 0])
==> x equals 2
```

aany

Returns True if any element in an array meets the selected criteria.

x = aany(op, inparray, rparam)

x = aany(op, inparray, rparam, maxlen=500, nosimd=True)

- op - The arithmetic comparison operation.
- inparray - The input data array to be examined.
- rparam - The 'y' parameter to be applied to 'op'.
- 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, use of SIMD is disabled.
- x - The boolean result.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
x = arrayfunc.aany('==', inparray, 5)
==> x equals True
x = arrayfunc.aany('==', inparray, 54, maxlen=5)
==> x equals True
x = arrayfunc.aany('==', inparray, -6, maxlen=5)
==> x equals False
```

aall

Returns True if all elements in an array meet the selected criteria.

x = aall(op, inparray, rparam)

x = aall(op, inparray, rparam, maxlen=500, nosimd=True)

- op - The arithmetic comparison operation.

- inparray - The input data array to be examined.
- rparam - The 'y' parameter to be applied to 'op'.
- 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, use of SIMD is disabled.
- x - The boolean result.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
x = arrayfunc.aall('<', inparray, 66)
==> x equals True
x = arrayfunc.aall('<', inparray, 66, maxlen=5)
==> x equals True
inparray = array.array('i', [1, 2, 5, 33, 54, 66])
x = arrayfunc.aall('<', inparray, 66)
==> x equals False
x = arrayfunc.aall('<', inparray, 66, maxlen=5)
==> x equals True
```

amax

Returns the maximum value in the array.

x = amax(inparray)

x = amax(inparray, maxlen=500)

x = amax(inparray, maxlen=500, nosimd=True)

- inparray - 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, use of SIMD is disabled.
- x - The maximum value.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
x = arrayfunc.amax(inparray)
==> x equals 54
x = arrayfunc.amax(inparray, maxlen=3)
==> x equals 5
```

amin

Returns the minimum value in the array.

x = amin(inparray)

x = amin(inparray, maxlen=500)

x = amin(inparray, maxlen=500, nosimd=True)

- inparray - 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, use of SIMD is disabled.
- x - The minimum value.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
x = arrayfunc.amin(inparray)
==> x equals -6
x = arrayfunc.amin(inparray, maxlen=3)
==> x equals 1
```

findindex

Returns the index of the first value in an array to meet the specified criteria.

x = findindex(op, inparray, rparam)

x = findindex(op, inparray, rparam, maxlen=500, nosimd=True)

- op - The arithmetic comparison operation.
- inparray - The input data array to be examined.
- rparam - The 'y' parameter to be applied to 'op'.
- 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, use of SIMD is disabled.
- x - The resulting index. This will be negative if no match was found.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
x = arrayfunc.findindex('==', inparray, 54)
==> x equals 4
x = arrayfunc.findindex('==', inparray, 54, maxlen=4)
==> x equals -1  (not 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.

x = findindices(op, inparray, outparray, rparam)

x = findindices(op, inparray, outparray, rparam, maxlen=500)

- op - The arithmetic comparison operation.
- inparray - The input data array to be examined.
- outparray - The output array. This must be an integer array of array type 'q' (signed long long).
- rparam - The 'y' parameter to be applied to 'op'.

- 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 indicating the number of matches found.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('q', [0]*6)
x = arrayfunc.findindices('<', inparray, outparray, 5)
==> ('i', [0, 1, 5, 0, 0, 0])
==> x equals 3
x = arrayfunc.findindices('<', inparray, outparray, 5, maxlen=4)
==> array('q', [0, 1, 0, 0, 0, 0])
==> x equals 2
```

asum

Calculate the arithmetic sum of an array.

For integer arrays, the intermediate sum is accumulated in the largest corresponding integer size. Signed integers are accumulated in the equivalent to an 'l' array type, and unsigned integers are accumulated in the equivalent to an 'L' array type. This means that integer arrays using smaller integer word sizes cannot overflow unless extremenly large arrays are used (and may be impossible due to limits on array indices in the array module).

asum(inparray)

asum(inparray, matherrors=True, maxlen=5, nosimd=True)

- inparray - The array to be summed.
- matherrors - If this keyword parameter is True, numeric overflow checking will be disabled. This is an optional 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, use of SIMD is disabled. SIMD will only be enabled if overflow checking is also disabled.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, 6])
arrayfunc.asum(inparray)
==> 101
inparray = array.array('i', [1, 2, 5, -88, -5, 2])
arrayfunc.asum(inparray, matherrors=True)
==> -83
inparray = array.array('i', [1, 2, 5, -88, -5, 2])
arrayfunc.asum(inparray, maxlen=5)
==> -85
```

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.

convert(inparray, outparray)

convert(inparray, outparray, maxlen=500)

- inparray - The input data array to be examined.
- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('d', [0.0]*6)
arrayfunc.convert(inparray, outparray)
==> ('d', [1.0, 2.0, 5.0, 33.0, 54.0, -6.0])
inparray = array.array('d', [5.7654]*10)
outparray = array.array('h', [0]*10)
arrayfunc.convert(inparray, outparray)
==> array('h', [5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
inparray = array.array('d', [5.7654]*10)
outparray = array.array('h', [0]*10)
arrayfunc.convert(inparray, outparray, maxlen=5)
==> array('h', [5, 5, 5, 5, 5, 0, 0, 0, 0, 0])
```

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 do so 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)
```

Supressing 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 supressed, 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).

Mathematical operator functions

add

Calculate add over the values in an array.

Equivalent to:	x + y
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)
```

- 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.

truediv

Calculate truediv over the values in an array.

Equivalent to:	x / y
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.

floordiv

Calculate floordiv over the values in an array.

Equivalent to:	x // y
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 % y
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 * y
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)
```

- 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.

neg

Calculate neg over the values in an array.

Equivalent to:	-x
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))
```

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

pow

Calculate pow over the values in an array.

Equivalent to:	x**y or math.pow(x, y)
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 - y
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)
```

- 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.

*abs_*

Calculate abs_ over the values in an array.

Equivalent to:	abs(x)
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))
```

- 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.
- 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:	x == y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.
- 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:	x > y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.
- 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:	x >= y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.

- 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:	x < y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.
- 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:	x <= y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.
- 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:	x != y
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	

Call formats:

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

- 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.
- 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 & y
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)
```

- 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.

*or_*

Calculate or_ over the values in an array.

Equivalent to:	x | y
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)
```

- 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.

xor

Calculate xor over the values in an array.

Equivalent to:	x ^ y
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)
```

- 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.

invert

Calculate invert over the values in an array.

Equivalent to:	~x
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)
```

- 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.

lshift

Calculate lshift over the values in an array.

Equivalent to:	x << y
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)
```

- 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.

rshift

Calculate rshift over the values in an array.

Equivalent to:	x >> y
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)
```

- 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.

Power and logarithmic functions

exp

Calculate exp over the values in an array.

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

Call formats:

```
sqrt(array1)
sqrt(array1, outparray)
sqrt(array1, maxlen=y)
sqrt(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.

Hyperbolic functions

acosh

Calculate acosh over the values in an array.

Equivalent to:	math.acosh(x)
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)
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)
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)
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)
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)
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)
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)
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)
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:	math.atan2(x, y)
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)

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

Call formats:

```
degrees(array1)
degrees(array1, outparray)
degrees(array1, maxlen=y)
degrees(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.

radians

Calculate radians over the values in an array.

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

Call formats:

```
radians(array1)
radians(array1, outparray)
radians(array1, maxlen=y)
radians(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.

Number-theoretic and representation functions

ceil

Calculate ceil over the values in an array.

Equivalent to:	math.ceil(x)
Array types supported:	f, d

Exceptions raised:	ArithmeticError

Call formats:

```
ceil(array1)
ceil(array1, outparray)
ceil(array1, maxlen=y)
ceil(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.

copysign

Calculate copysign over the values in an array.

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

Call formats:

```
floor(array1)
floor(array1, outparray)
floor(array1, maxlen=y)
floor(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.

fmod

Calculate fmod over the values in an array.

Equivalent to:	math.fmod(x, y)
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:	math.isfinite(x)
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 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.

isinf

Calculate isinf over the values in an array.

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

Call formats:

```
trunc(array1)
trunc(array1, outparray)
trunc(array1, maxlen=y)
trunc(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.

Special functions

erf

Calculate erf over the values in an array.

Equivalent to:	math.erf(x)
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)
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)
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)
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.

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 intializer. 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 symetrical 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

Divison 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 commmon 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 occured 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 ocurred 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 occured 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.

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 64 bit x86 (i.e. AMD64) using the GCC compiler. 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.

Data Type Support

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

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X		X		X						X	X
aany	X		X		X						X	X
amax	X	X	X	X	X	X					X	X
amin	X	X	X	X	X	X					X	X
asum											X	X
findindex	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

Non-Optmised Performance

In this set of tests, all error checking was turned on (the default state) and SIMD acceleration was disabled (not the default).

Relative Performance - Python Time / Arrayfunc Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	8.5	6.9	8.5	11	8.7	11	11	11	6.6	10.0	17	13
aany	5.7	3.7	4.8	4.5	3.7	5.8	5.3	5.6	5.1	5.5	7.0	6.3
afilter	116	116	114	110	116	115	115	114	116	116	116	114
amax	30	34	32	27	27	29	14	16	13	15	32	21
amin	16	16	20	18	26	27	14	13	14	16	21	47
asum	7.9	10	8.8	10	8.8	11	8.2	11	8.1	8.7	5.7	6.2
compress	37	35	37	24	33	20	39	23	42	26	33	32
count	213	212	201	213	160	106	156	115	153	111	75	76
cycle	85	84	83	79	80	58	81	63	81	62	61	59
dropwhile	146	132	149	144	176	179	169	173	171	167	155	136
findindex	14	14	16	20	19	18	16	17	16	17	18	16
findindices	26	26	26	26	26	27	26	27	25	27	23	23
repeat	132	131	126	126	109	45	118	41	124	38	110	114
takewhile	233	210	249	267	250	192	177	123	157	121	317	148
add	101	126	91	130	83	122	66	57	68	52	115	65
truediv	74	68	75	74	75	65	71	72	63	67	150	85
floordiv	34	38	35	41	36	35	34	38	34	31	129	91
mod	27	31	22	33	34	28	32	24	37	22	75	61

mul	19	30	15	24	9.5	86	5.8	47	5.8	46	133	65
neg	135		157		138		93		95		117	89
pow	56	62	52	56	45	64	26	71	25	56	9.5	16
sub	79	173	99	166	100	149	55	54	56	57	137	67
and_	313	304	257	294	218	198	71	59	66	65		
or_	311	314	277	323	229	197	75	64	70	65		
xor	200	190	210	228	196	164	75	72	67	61		
invert	362	256	352	249	293	222	195	185	227	213		
eq	125	117	130	123	123	115	101	91	110	88	119	90
gt	161	103	103	106	185	110	84	100	88	110	111	98
ge	117	132	109	120	103	125	86	116	94	134	148	107
lt	175	96	103	101	173	113	85	119	92	117	127	99
le	107	125	122	119	118	131	90	98	81	101	138	86
ne	155	101	95	101	180	199	87	110	97	119	128	102
lshift	203	193	185	195	224	140	74	63	75	65		
rshift	195	290	200	256	219	216	73	68	82	67		
abs_	135		102		114		102		91		159	111
acos											14	12
acosh											9.5	6.3
asin											14	13
asinh											6.9	7.0
atan											12	11
atan2											10	10
atanh											7.4	7.9
ceil											234	178
copysign											217	99
cos											15	8.1
cosh											13	8.6
degrees											147	129
erf											16	13
erfc											10	7.6
exp											22	10
expm1											6.7	6.7
fabs											166	149
factorial	214	209	200	203	213	181	145	121	134	134		
floor											233	175
fmod											13	12
gamma											1.5	1.3

hypot											24	17
isfinite											140	122
isinf											151	130
isnan											159	144
ldexp											31	33
lgamma											9.1	5.8
log											26	8.8
log10											15	7.5
log1p											8.5	8.8
log2											22	11
radians											151	131
sin											14	8.3
sinh											5.2	5.6
sqrt											23	19
tan											6.8	6.2
tanh											5.5	5.6
trunc											251	200

Stat	Value
Average:	89
Maximum:	362
Minimum:	1.3
Array size:	100000

Optmised Performance

In this set of tests, all arithmatic error checking was disabled (not the default state) and SIMD acceleration was enabled (the normal default).

Relative Performance with Optimisations - Python Time / Arrayfunc Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	105	6.3	59	7.4	27	13	11	11	11	10	42	23
aany	53	4.1	31	4.5	14	5.8	5.1	5.5	5.2	5.6	23	11
afilter	117	116	114	109	116	115	115	115	116	116	116	114
amax	79	86	40	40	78	130	14	16	14	15	113	62
amin	74	73	37	37	115	77	14	16	14	16	121	59
asum	13	16	14	15	14	18	11	19	14	15	22	13
compress	37	34	37	24	32	17	40	23	43	26	33	32
count	203	189	200	213	159	116	156	122	153	102	73	76
cycle	85	84	80	78	67	58	85	63	82	62	61	59
dropwhile	149	144	149	145	178	176	168	173	171	162	159	145

findindex	188	14	104	20	50	17	16	17	15	15	62	33
findindices	26	26	26	26	26	27	26	27	25	27	23	23
repeat	128	131	134	127	123	44	121	41	124	42	120	97
takewhile	251	234	257	272	250	194	167	137	164	124	317	153
add	191	165	170	267	250	204	135	115	137	108	149	100
truediv	73	63	66	71	79	61	80	75	68	68	204	184
floordiv	38	34	40	43	43	33	41	40	42	33	189	177
mod	28	32	24	33	36	29	35	32	40	30	114	103
mul	174	169	176	153	180	166	107	89	107	88	160	131
neg	165		202		127		95		103		185	107
pow	45	49	44	48	30	66	18	62	17	51	8.5	19
sub	165	265	259	251	177	142	113	109	119	106	177	117
and_	213	205	198	201	185	161	130	115	135	129		
or_	211	197	178	174	195	160	136	115	135	126		
xor	290	302	314	340	280	258	150	147	145	127		
invert	361	206	394	266	336	211	216	197	212	225		
eq	133	114	98	93	94	157	109	93	137	141	130	112
gt	95	85	92	108	99	101	96	111	103	164	150	141
ge	118	127	90	99	88	127	89	97	153	199	131	122
lt	134	142	138	157	153	126	158	147	105	180	140	134
le	120	136	150	140	148	175	144	149	129	170	137	126
ne	148	151	150	161	149	108	144	178	108	167	138	131
lshift	177	199	173	197	247	162	152	123	155	122		
rshift	231	328	241	305	212	232	139	193	159	177		
abs_	143		140		156		113		106		176	109
acos											15	12
acosh											11	6.4
asin											16	14
asinh											7.0	7.3
atan											14	12
atan2											9.4	8.6
atanh											7.6	8.3
ceil											256	215
copysign											277	167
cos											16	8.2
cosh											15	8.7
degrees											273	203
erf											17	14

erfc											10	7.9
exp											23	10
expm1											6.9	7.3
fabs											163	160
factorial	184	196	185	142	183	190	130	132	144	122		
floor											257	216
fmod											14	14
gamma											1.5	1.3
hypot											26	15
isfinite											140	122
isinf											151	131
isnan											159	146
ldexp											33	35
lgamma											9.2	6.1
log											31	9.0
log10											16	7.7
log1p											9.1	9.9
log2											25	12
radians											186	140
sin											17	8.4
sinh											5.3	5.5
sqrt											28	22
tan											7.1	6.3
tanh											5.5	5.8
trunc											282	282

Stat	Value
Average:	106
Maximum:	394
Minimum:	1.3
Array size:	100000

SIMD Optimisations

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.

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

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	12		6.9		3.1						2.5	1.8

aany	9.2		6.5		3.8						3.3	1.7
amax	2.7	2.5	1.3	1.5	2.9	4.5					3.5	3.0
amin	4.5	4.5	1.9	2.0	4.5	2.9					5.7	1.3
asum											3.9	2.0
findindex	13		6.5		2.7						3.4	2.0

Array Size Versus Performance

Benchmark the effects of array size on a selected arrayfunc function.

Add two arrays - times faster than Python, unoptimised.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	1.8	1.7	1.6	1.6	1.6	1.3	1.5	1.2	1.4	1.2	1.4	1.4
100	13	13	13	12	13	9.5	11	9.1	11	8.9	11	10
1000	57	73	62	70	62	56	59	52	54	51	67	60
10000	84	150	103	140	116	121	106	107	90	119	141	128
100000	87	158	110	152	127	129	80	63	71	59	150	80
1000000	87	131	102	121	86	79	59	48	58	50	116	65
10000000	90	145	96	131	101	87	61	47	61	49	114	65

Add constant to array - times faster than Python, optimised.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	1.2	1.1	1.0	1.1	1.1	0.8	1.1	0.8	1.0	0.8	1.0	1.0
100	9.0	8.6	9.1	8.3	8.8	6.4	8.4	6.6	8.0	6.7	8.4	8.0
1000	65	59	68	59	58	47	58	46	58	44	65	62
10000	166	176	217	165	166	117	174	136	159	132	235	197
100000	199	208	277	208	216	137	151	134	133	108	312	158
1000000	202	193	231	171	136	107	79	63	79	67	165	84
10000000	198	192	247	174	146	111	84	58	79	62	158	82

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 require exact application requires

Benchmark your application on the specific platform

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 18.10	64 bit	GCC	3.6
Debian 9	32 bit	GCC	3.5
Debian 9	64 bit	GCC	3.5
OpenSuse 15	64 bit	GCC	3.6
FreeBSD 11	64 bit	LLVM	3.5
MS Windows 10	64 bit	MS Visual Studio C 2015	3.7
Raspbian (RPi 3)	32 bit	GCC	3.5

The Raspbian (RPi 3) tests were conducted on a Raspberry Pi ARM CPU. All others were conducted using VMs running on x86 hardware.