ArrayFunc

Authors:	Michael Griffin
Version:	4.0.0 for 2018-05-14
Copyright:	2014 - 2018
License:	This document may be distributed under the Apache License V2.0.
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 These have all been replaced by individual functions which perform the same calculations but in a more direct way.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)
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.

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(arrayfunc.aops.af_gt, inparray, outparray, 10)
==> array('i', [33, 54, 0, 0, 0, 0])
==> x equals 2
x = arrayfunc.afilter(arrayfunc.aops.af_gt, 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(arrayfunc.aops.af_lt, inparray, outparray, 10)
==> array('i', [33, 54, 0, 0, 0, 0])
==> x equals 3
x = arrayfunc.dropwhile(arrayfunc.aops.af_lt, 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(arrayfunc.aops.af_lt, inparray, outparray, 10)
==> array('i', [1, 2, 5, 0, 0, 0])
==> x equals 3
x = arrayfunc.takewhile(arrayfunc.aops.af_lt, 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(arrayfunc.aops.af_eq, inparray, 5)
==> x equals True
x = arrayfunc.aany(arrayfunc.aops.af_eq, inparray, 54, maxlen=5)
==> x equals True
x = arrayfunc.aany(arrayfunc.aops.af_eq, 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(arrayfunc.aops.af_lt, inparray, 66)
==> x equals True
x = arrayfunc.aall(arrayfunc.aops.af_lt, inparray, 66, maxlen=5)
==> x equals True
inparray = array.array('i', [1, 2, 5, 33, 54, 66])
x = arrayfunc.aall(arrayfunc.aops.af_lt, inparray, 66)
==> x equals False
x = arrayfunc.aall(arrayfunc.aops.af_lt, 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(arrayfunc.aops.af_eq, inparray, 54)
==> x equals 4
x = arrayfunc.findindex(arrayfunc.aops.af_eq, 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(arrayfunc.aops.af_lt, inparray, outparray, 5)
==> ('i', [0, 1, 5, 0, 0, 0])
==> x equals 3
x = arrayfunc.findindices(arrayfunc.aops.af_lt, 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 extremely large arrays are used (and may be impossible due to limits on array indices in the array module).

asum(inparray)

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

> • inparray - The array to be summed.

> • disovfl - If this keyword parameter is True, integer 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, disovfl=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])
```

TODO: Write new section on operators for findindex, takewhile, etc.

Since the old operators have been eliminated along with amap, starmap, etc. this section needs a new description of the flags for gt, lt, eq, etc.

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 platforms 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 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.

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 "disovfl" parameter. Setting "disovfl" to true will *disable* overflow checking, while setting it to false will *enable* overflow checking. Checking is enabled by default, including when the "disovfl" 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 -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.

ZeroDivision Error	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

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 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. Error checking was on in all tests.

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 Arrayfunc functions in the "other functions" table 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.

Arithmetical Functions

function	b	B	h	H	i	I	l	L	q	Q	f	d
add	113	151	90	150	97	130	68	63	70	54	156	73
truediv	95	91	91	88	85	79	82	74	78	73	191	106
floordiv	53	48	52	50	45	45	42	44	42	40	90	84
mod	33	41	31	43	42	37	44	30	42	31	46	51
mul	19	41	19	28	11	15	6.9	9.2	6.9	9.0	156	69
neg	160		180		162		103		101		144	83
pow	63	66	56	57	52	45	28	26	28	25	24	20
sub	101	202	110	195	97	140	68	58	76	65	155	81
and_	293	332	316	318	263	235	78	69	80	64		
or_	230	232	243	225	210	205	89	70	84	68		
xor	305	325	323	325	271	217	74	71	67	59		
invert	384	281	400	319	337	269	249	205	220	256		
eq	125	112	122	121	124	129	124	121	88	116	131	106
gt	144	152	97	111	171	153	97	109	109	96	142	109
ge	201	209	136	158	181	178	92	99	92	143	140	104
lt	155	142	140	126	147	141	99	113	138	85	168	109
le	226	203	166	102	196	183	144	125	110	118	145	117
ne	196	201	170	107	192	186	108	100	117	107	134	107
lshift	235	304	226	290	209	239	88	66	90	72		
rshift	253	242	255	234	304	189	84	73	84	71		
abs_	144		140		119		100		97		166	122
acos											16	14
acosh											7.8	7.3
asin											17	15

asinh											8.0	8.2
atan											15	15
atan2											13	11
atanh											8.4	9.1
ceil											127	110
copysign											285	119
cos											21	10
cosh											12	9.6
degrees											189	128
erf											16	15
erfc											9.7	8.3
exp											17	11
expm1											8.3	8.2
fabs											222	155
factorial	75	92	93	88	84	77	89	74	95	80		
floor											121	104
fmod											12	14
gamma											1.3	1.5
hypot											33	22
isinf											144	127
isnan											179	155
ldexp											32	34
lgamma											8.8	6.8
log											16	12
log10											11	8.7
log1p											9.0	11
log2											14	12
radians											181	139
sin											22	11
sinh											6.2	5.9
sqrt											31	27
tan											8.4	6.8
tanh											6.8	7.4
trunc											94	77

Stat	Value
Average:	108
Maximum:	400

Minimum:	1.3
Array size:	100000

Other Functions

Asumov in the following indicates asum with overflow checking turned off. This is required to enable SIMD features.

Arrayfunc faster than Python factor.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	10	9.8	8.9	7.7	10	9.9	6.6	5.9	6.1	6.3	13	9.4
aany	7.7	10.0	10	6.0	7.5	7.4	6.2	6.3	6.2	6.3	11	8.7
afilter	265	206	242	239	165	121	106	75	104	77	199	109
amax	32	29	21	23	20	20	14	14	14	14	36	27
amin	24	24	30	34	21	21	14	14	14	14	48	25
asum	6.6	11	6.9	10	7.5	10	6.7	7.1	7.3	7.1	11	10
asumov	13	18	14	16	12	14	7.5	8.6	7.9	8.0	11	11
compress	42	42	41	35	41	21	34	16	30	17	28	32
count	241	243	236	299	138	87	69	50	75	51	114	105
cycle	112	118	101	106	89	61	60	44	72	43	39	40
dropwhile	107	102	104	103	100	70	65	46	66	47	104	66
findindex	13	16	16	12	18	15	11	11	11	11	27	20
findindices	39	31	38	36	35	29	26	27	26	26	41	36
repeat	128	146	122	141	88	24	52	14	52	15	135	80
takewhile	222	223	213	224	205	129	109	81	109	84	205	110

Stat	Value
Average:	57
Maximum:	299
Minimum:	5.9
Array size:	1000000

Arrayfunc with SIMD faster than Python factor.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	89		32		14						21	11
aany	120		60		20						29	10
afilter												
amax	527	501	117	114	40	40					66	33
amin	335	345	96	96	34	35					57	30
asum												
asumov											32	14

function	b	B	h	H	i	I	l	L	q	Q	f	d
compress												
count												
cycle												
dropwhile												
findindex	246		74		28						54	27
findindices												
repeat												
takewhile												

Stat	Value
Average:	101
Maximum:	527
Minimum:	10.4
Array size:	1000000

Arrayfunc with SIMD faster than Arrayfunc without SIMD factor. SIMD is not supported for all array types, so some types will not show a speed up.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	8.8		3.7		1.4						1.5	1.2
aany	16		5.9		2.6						2.6	1.2
afilter												
amax	16	17	5.5	5.1	2.0	2.0					1.8	1.2
amin	14	15	3.2	2.8	1.7	1.7					1.2	1.2
asum												
asumov												2.8	1.3
compress												
count												
cycle												
dropwhile												
findindex	19		4.7		1.5						2.0	1.3
findindices												
repeat												
takewhile												

Stat	Value
Average:	5
Maximum:	19
Minimum:	1.2
Array size:	1000000