ArrayFunc

Authors: Michael Griffin
Version: 1.0.0 for 2016-03-23

Language: Python 3.4 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.

Functions

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
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.
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.

Operating on Arrays

Function	Description
amap	Apply an operator to each element of an array, together with an optional second parameter (for operators taking two parameters). The results are written to a second array.

amapi	Like amap, but the results are written in place to the input array.
starmap	Like amap, but where a second array acts as the second parameter. The results are written to an output array.
starmapi	Like starmap, but the results are written in place to the first input array.
asum	Calculate the arithmetic sum of an array.
acalc	Calculate arbitrary equations over 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.

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.

Details

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)

- 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.
- 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)

- 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.
- 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)

- 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.
- 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)

- 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.
- 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)

- 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.
- 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
```

amap

Apply an operator to each element of an array, together with an optional second parameter (for operators taking two parameters). The results are written to a second array.

amap(op, inparray, outparray, rparam)

amap(op, inparray, outparray, rparam, disovfl=True)

amap(op, inparray, outparray, rparam, disovfl=True, maxlen=500)

- op - The arithmetic comparison operation.
- inparray - The input data array to be examined.
- outparray - The output array.
- rparam - The 'y' parameter to be applied to 'op'. This is an optional parameter.
- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
outparray = array.array('i', [0]*6)
arrayfunc.amap(arrayfunc.aops.af_add, inparray, outparray, 5)
==> ('i', [6, 7, 10, 38, 59, -1])
arrayfunc.amap(arrayfunc.aops.af_add, inparray, outparray, 5, disovfl=True)
==> ('i', [6, 7, 10, 38, 59, -1])
arrayfunc.amap(arrayfunc.aops.af_add, inparray, outparray, 5, disovfl=False)
==> ('i', [6, 7, 10, 38, 59, -1])
inparray = array.array('i', [1, 2, 3, 4, 5, 6])
arrayfunc.amap(arrayfunc.aops.math_factorial, inparray, outparray)
==> ('i', [1, 2, 6, 24, 120, 720])
outparray = array.array('i', [0]*6)
arrayfunc.amap(arrayfunc.aops.math_factorial, inparray, outparray, maxlen=5)
==> array('i', [1, 2, 6, 24, 120, 0])
```

amapi

Like amap, but the results are written in place to the input array.

amapi(op, inparray, rparam)

amapi(op, inparray, rparam, disovfl=True)

amapi(op, inparray, rparam, disovfl=True, maxlen=500)

- op - The arithmetic comparison operation.
- inparray - The input data array to be examined.
- rparam - The 'y' parameter to be applied to 'op'. This is an optional parameter.
- 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.

example:

```
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
arrayfunc.amapi(arrayfunc.aops.af_add, inparray, 5)
==> ('i', [6, 7, 10, 38, 59, -1])
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
arrayfunc.amapi(arrayfunc.aops.af_add, inparray, 5, disovfl=True)
==> ('i', [6, 7, 10, 38, 59, -1])
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
arrayfunc.amapi(arrayfunc.aops.af_add, inparray, 5, disovfl=False)
==> ('i', [6, 7, 10, 38, 59, -1])
inparray = array.array('i', [1, 2, 3, 4, 5, 6])
arrayfunc.amapi(arrayfunc.aops.math_factorial, inparray)
==> ('i', [1, 2, 6, 24, 120, 720])
inparray = array.array('i', [1, 2, 5, 33, 54, -6])
arrayfunc.amapi(arrayfunc.aops.af_add, inparray, 5, disovfl=False, maxlen=5)
==> array('i', [6, 7, 10, 38, 59, -6])
```

starmap

Like amap, but where a second array acts as the second parameter. The results are written to an output array. All valid operators and math functions must take a second parameter (for single parameter operators or math functions, use amap).

starmap(op, inparray1, inparray2, outparray)

starmap(op, inparray1, inparray2, outparray, disovfl=True)

starmap(op, inparray1, inparray2, outparray, disovfl=True, maxlen=500)

- op - The arithmetic comparison operation.
- inparray1 - The first input data array to be examined.
- inparray2 - The second input data array to be examined.
- outparray - The output array.
- 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.

example:

```
inparray1 = array.array('i', [1, 2, 5, 33, 54, 6])
inparray2 = array.array('i', [1, 2, 5, -88, -5, 2])
```

```
outparray = array.array('i', [0]*6)
arrayfunc.starmap(arrayfunc.aops.af_add, inparray1, inparray2, outparray)
==> array('i', [2, 4, 10, -55, 49, 8])
arrayfunc.starmap(arrayfunc.aops.af_add, inparray1, inparray2, outparray, disovfl=True)
==> array('i', [2, 4, 10, -55, 49, 8])
outparray = array.array('i', [0]*6)
arrayfunc.starmap(arrayfunc.aops.af_add, inparray1, inparray2, outparray, maxlen=5)
==> array('i', [2, 4, 10, -55, 49, 0])
```

starmapi

Like starmap, but the results are written in place to the first input array.

starmapi(op, inparray1, inparray2)

starmapi(op, inparray1, inparray2, disovfl=True)

starmapi(op, inparray1, inparray2, disovfl=True, maxlen=500)

- op - The arithmetic comparison operation.
- inparray1 - The first input data array to be examined.
- inparray2 - The second input data array to be examined.
- 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.

example:

```
inparray1 = array.array('i', [1, 2, 5, 33, 54, 6])
inparray2 = array.array('i', [1, 2, 5, -88, -5, 2])
arrayfunc.starmapi(arrayfunc.aops.af_add, inparray1, inparray2)
==> array('i', [2, 4, 10, -55, 49, 8])
inparray1 = array.array('i', [1, 2, 5, 33, 54, 6])
arrayfunc.starmapi(arrayfunc.aops.af_add, inparray1, inparray2, disovfl=True)
==> array('i', [2, 4, 10, -55, 49, 8])
inparray1 = array.array('i', [1, 2, 5, 33, 54, 6])
arrayfunc.starmapi(arrayfunc.aops.af_add, inparray1, inparray2, disovfl=True, maxlen=5)
==> array('i', [2, 4, 10, -55, 49, 6])
```

asum

Calculate the arithmetic sum of an array.

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

asum(inparray)

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

- 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.

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])
```

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
bytes	Python bytes type	bytes_min	bytes_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
```

ACalc

Description

Calculate arbitrary equations over an array.

ACalc solves complex equations (expressions) over an array. It accepts a valid Python mathematical expression as a string, compiles it, and executes it. The expression can include constants, variables, and the same functions as defined in the "math" module.

ACalc consists of a class "calc" with two methods, "comp" (compile) and "execute".

For simple calculations, amap will normally be much, much faster than acalc. However, acalc is useful for equations requiring multiple terms, as it can solve them in a single operation whereas amap (or amapi) would require multiple function calls (once for each term).

Initialisation

The "calc" class is initialised with the input and output arrays. The input and output arrays must be of the same array type. The array type determines the data type of the calculation. That is, an integer array will result in integer math, and a floating point array will result in floating point math.

The first parameter is the input array, and the second parameter is the output array. These arrays remain associated with the equation object.

example:

```
data = array.array('b', [0,1,2,3,4,5,6,7,8,9])
dataout = array.array('b', [0]*len(data))
eqnd = acalc.calc(data, dataout)
```

Compiling

The compile method accepts three positional parameters. These are:

- Equation - This is the equation as a string.
- Array variable - This defines which variable in the equation represents the current array index value. This must be a string which follows the same rules as valid Python variable names.
- Other variables - This is a sequence of strings, with each element corresponding to a variable in the equation. The sequence can be a list or a tuple.

example:

```
eqnd.comp('x + y - z + 5', 'x', ['y', 'z'])
```

example:

```
eqnd.comp('-x', 'x', [])
```

example:

```
eqnd.comp('abs(x) + y - (z << 2)', 'x', ('y', 'z'))
```

Executing

Once an equation is compiled, it can be executed. A compiled equation can be executed multiple times with different parameter values without recompiling it.

The execute method accepts one positional parameter which represents the additional variables and two keyword parameters which are used to control the execution of the equation.

- Variable values - This is a list or tuple of of numeric values which corresponds to the additional (non-array) variables in the equation. The order and number of elements must match the sequence of additional variables defined in the compile step.
- disovfl - If this keyword parameter is True, 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.

example:

```
eqnd.execute([-25, 3])
```

example:

```
eqnd.execute([-25, 3], disovfl=True)
```

example:

```
eqnd.execute([-25, 3], disovfl=False, maxlen=500)
```

Complete Example

example:

```
import array
from arraycalc import acalc
data = array.array('b', [0,1,2,3,4,5,6,7,8,9])
dataout = array.array('b', [0]*len(data))
eqnd = acalc.calc(data, dataout)
eqnd.comp('x + y - z + 5', 'x', ['y', 'z'])
eqnd.execute([-25, 3])
print(dataout)
array('b', [-23, -22, -21, -20, -19, -18, -17, -16, -15, -14])
```

Option Flags

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.

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

Bytes Type

The 'bytes' array type is also supported, and is treated the same as an unsigned char (array type 'B'). To conduct operations on a Python 'bytes' string, simply pass the bytes string in place of an array. Any integer operations which are valid for an unsigned char array will be valid for a bytes string.

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.

Operators

The following lists the operators available, together with the types of arrays they are compamtible with.

Some operators are checked for integer overflow or underflow. These are indicated by the "OV" column. An overflow or underflow will generate an error.

In the following, the values in the input data array are represented by 'x'. The second input array or numerical parameter is represented by 'y'. Some operators come in two forms, where the second allows the 'x' and 'y' parameters to be exchanged in cases where this may produce a different result.

The operator categories are used to indicate which functions support which operators.

Python Equivalent Operators and Functions

The following operators and functions are equivalent to ones found in the Python standard library. For explanations of the math functions, see the Python standard documentation for the standard math library.

Name	Equivalent to	b h i l	B H I L	f d	OV	Compare Ops	Win
af_add	x + y	X	X	X	X		X
af_div	x / y	X	X	X	X		X
af_div_r	y / x	X	X	X	X		X
af_floordiv	x // y	X	X	X	X		X
af_floordiv_r	y // x	X	X	X	X		X
af_mod	x % y	X	X	X	X		X
af_mod_r	y % x	X	X	X	X		X
af_mult	x * y	X	X	X	X		X
af_neg	-x	X		X	X		X
af_pow	x**y	X	X	X	X		X
af_pow_r	y**x	X	X	X	X		X
af_sub	x - y	X	X	X	X		X
af_sub_r	y - x	X	X	X	X		X
af_and	x & y	X	X				X
af_or	x | y	X	X				X
af_xor	x ^ y	X	X				X
af_invert	~x	X	X				X
af_eq	x == y	X	X	X		X	X
af_gt	x > y	X	X	X		X	X
af_gte	x >= y	X	X	X		X	X
af_lt	x < y	X	X	X		X	X
af_lte	x <= y	X	X	X		X	X
af_ne	x != y	X	X	X		X	X
af_lshift	x << y	X	X				X
af_lshift_r	y << x	X	X				X

af_rshift	x >> y	X	X				X
af_rshift_r	y >> x	X	X				X
af_abs	abs(x)	X		X	X		X
math_acos	math.acos(x)			X			X
math_acosh	math.acosh(x)			X			
math_asin	math.asin(x)			X			X
math_asinh	math.asinh(x)			X			
math_atan	math.atan(x)			X			X
math_atan2	math.atan2(x, y)			X			X
math_atan2_r	math.atan2(y, x)			X			X
math_atanh	math.atanh(x)			X			
math_ceil	math.ceil(x)			X			X
math_copysign	math.copysign(x, y)			X			X
math_cos	math.cos(x)			X			X
math_cosh	math.cosh(x)			X			X
math_degrees	math.degrees(x)			X			X
math_erf	math.erf(x)			X			
math_erfc	math.erfc(x)			X			
math_exp	math.exp(x)			X			X
math_expm1	math.expm1(x)			X			
math_fabs	math.fabs(x)			X			X
math_factorial	math.factorial(x)	X	X		X		X
math_floor	math.floor(x)			X			X
math_fmod	math.fmod(x, y)			X			X
math_fmod_r	math.fmod(y, x)			X			X
math_gamma	math.gamma(x)			X			
math_hypot	math.hypot(x, y)			X			X
math_hypot_r	math.hypot(y, x)			X			X
math_isinf	math.isinf(x)			X			
math_isnan	math.isnan(x)			X			
math_ldexp	math.ldexp(x, y)			X			X
math_lgamma	math.lgamma(x)			X			
math_log	math.log(x)			X			X
math_log10	math.log10(x)			X			X
math_log1p	math.log1p(x)			X			
math_pow	math.pow(x, y)			X			X
math_pow_r	math.pow(y, x)			X			X
math_radians	math.radians(x)			X			X

math_sin	math.sin(x)			X			X
math_sinh	math.sinh(x)			X			X
math_sqrt	math.sqrt(x)			X			X
math_tan	math.tan(x)			X			X
math_tanh	math.tanh(x)			X			X
math_trunc	math.trunc(x)			X			

Additional Operators

The arrayfuncs module includes operators which are not found in the Python standard library. These are the "substitute" operators. Substitute operators compare the contents of each array element to the parameter (which must be included in the call). If the comparison evaluates to true, the array contents at that index are replaced by (substituted with) the parameter. If the comparison fails, the contents of the input array are used.

Name	Equivalent to	b h i l	B H I L	f d	OV	Compare Ops	Win
aops_subst_gt	x > y	X	X	X			X
aops_subst_gte	x >= y	X	X	X			X
aops_subst_lt	x < y	X	X	X			X
aops_subst_lte	x <= y	X	X	X			X

For example, and array [1, 2, 3, 4, -2] is evaluated using the "aops_subst_gt" and a parameter of 3. The resulting output is [1, 2, 3, 3, -2]. The effect has been to limit the maximum value to no more than 3.

ACalc Operators and Functions

The following operators and functions are equivalent to ones found in the Python standard library. ACalc uses the representation in the "equivalent to" column to actually specify the equations. The "name" column is only for reference purposes.

For explanations of the math functions, see the Python standard documentation for the standard math library.

Name	Equivalent to	b h i l	B H I L	f d	OV	Win
add	x + y	X	X	X	X	X
sub	x - y	X	X	X	X	X
mult	x * y	X	X	X	X	X
div	x / y	X	X	X	X	X
floordiv	x // y	X	X	X	X	X
mod	x % y	X	X	X	X	X
uadd	+x	X	X	X		X
usub	-x	X	X	X	X	X
pow	x**y	X	X	X	X	X
bitand	x & y	X	X			X
bitor	x | y	X	X			X

bitxor	x ^ y	X	X			X
invert	~x	X	X			X
lshift	x << y	X	X			X
rshift	x >> y	X	X			X
abs	abs(x)	X	X	X	X	X
math.acos	math.acos(x)			X		X
math.acosh	math.acosh(x)			X		
math.asin	math.asin(x)			X		X
math.asinh	math.asinh(x)			X		
math.atan	math.atan(x)			X		X
math.atan2	math.atan2(x, y)			X		X
math.atanh	math.atanh(x)			X		
math.ceil	math.ceil(x)			X		X
math.copysign	math.copysign(x, y)			X		X
math.cos	math.cos(x)			X		X
math.cosh	math.cosh(x)			X		X
math.degrees	math.degrees(x)			X		X
math.erf	math.erf(x)			X		
math.erfc	math.erfc(x)			X		
math.exp	math.exp(x)			X		X
math.expm1	math.expm1(x)			X		
math.fabs	math.fabs(x)			X		X
math.factorial	math.factorial(x)	X	X		X	X
math.floor	math.floor(x)			X		X
math.fmod	math.fmod(x, y)			X		X
math.gamma	math.gamma(x)			X		
math.hypot	math.hypot(x, y)			X		X
math.ldexp	math.ldexp(x, y)			X		X
math.lgamma	math.lgamma(x)			X		
math.log	math.log(x)			X		X
math.log10	math.log10(x)			X		X
math.log1p	math.log1p(x)			X		
math.pow	math.pow(x, y)			X		X
math.radians	math.radians(x)			X		X
math.sin	math.sin(x)			X		X
math.sinh	math.sinh(x)			X		X
math.sqrt	math.sqrt(x)			X		X
math.tan	math.tan(x)			X		X

math.tanh	math.tanh(x)			X		X
math.trunc	math.trunc(x)			X		

Notes on Operators and Functions

- The regular and floor division operators (/, //) all perform division using the native division instructions. That is, integer division always results in an integer result, and floating point division always results in a floating point result.
- The math.gamma function (and the Python math.gamma) functions are equivalent to the C library tgamma function. The C library gamma and lgamma functions are equivalent to each other.
- The raise to power (x**y) operator will not accept a negative exponent for integers, as the result would be a fractional number which is not compatible with an integer array.
- Some mathematical operations are not supported by the Microsoft compiler. This This is indicated by the *Win* column.

Platform Compiler Support

Amap, Amapi, and ACalc Functions

The Microsoft Visual Studio 2010 C compiler is built to an older C standard (C89) than GCC and does not have some functions in its standard library. The Microsoft compiler is used for the MS Windows versions of Python.

Since Arrayfunc depends on the standard C libraries to implement the underlying math functions, this means that the MS Windows version of Arrayfunc does not implement some math functions. These are indicated above by the "Win" column in the above tables.

The "math" library in Python implements it's own versions of these functions to paper over the missing functions for the MS Windows version. Arrayfunc however relies on the C libraries.

Long Long Integer ('Q' and 'q') Array Types

Not all platforms support long long array types. The presence of these arrays can be tested for by examining the array module array codes.

Example:

```
if 'q' in array.typecodes:
        print('Long long integer arrays are present')
```

Using Unsigned Long Long Arrays with Convert on Microsoft Windows

The Microsoft VC 2010 compiler appears to not convert floating point numbers to unsigned long long integers correctly under some circumstances. Due to this problem, converting float or double to unsigned long long is disabled when the library is compiled with the Microsoft VC compiler. Attempts to perform this operation will result in an exception.

Integer Overflow Checking

Overflow checking in integer operators is conducted as follows:

Overflow 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.

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 or bytes expected.	A non-array parameter was found where an array (or bytes) 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.

Exceptions - ACalc

ACalc has additional exceptions which are defined here. In addition to these, some of the general exceptions also apply.

Initialisation

This are the exceptions which can occurr during class initialisation.

Exception type	Text	Description
TypeError	first parameter must be an array or bytes in ACalc init.	The first parameter is of an incorrect type.

TypeError	second parameter must be an array or bytes in ACalc init.	The first parameter is of an incorrect type.
TypeError	unknown array type in ACalc init.	The type of one of the parameters is not recognised.
TypeError	data array type mismatch error in ACalc init.	The parameters are not of the same array type.

Compile

These are the exceptions which can occur during the compile phase.

Exception type	Text	Description
ValueError	unknown call name in ACalc compile.	A function call name is not recognised.
OverflowError	equation constant 'x' is out of range for the selected array type in ACalc compile.	The specified constant is not valid for the array type selected.
ValueError	Invalid operations in ACalc compile: 'x'.	The specified operators are invalid.
ValueError	Unsupported operations in ACalc compile: 'x'	The specified operators are not supported on the current platform. Some platforms do not support all features.
ValueError	array name used in additional parameters in ACalc compile.	The variable which specifies the array element was repeated in the additional parameters list.
ValueError	undefined variables in ACalc compile: 'x'.	A variable was used in the equation which was not defined in the parameter list.
ValueError	unused variables in ACalc compile: 'x'.	A variable was defined in the parameter list but was not used in the equation.
ValueError	duplicate parameter names in ACalc compile.	One or more variable names were repeated in the parameter list.
ValueError	unbalanced parentheses in ACalc compile.	The left and right parentheses "(", ")", do not match.
ValueError	invalid tokens in ACalc compile: 'x'.	An invalid symbol was present in the equation.
SyntaxError	invalid syntax in equation in ACalc compile in position 'x' 'y'.	A syntax error was found in the equation.
ValueError	unsupported element in equation in ACalc compile.	The equation contains one or more elements which are likely valid Python, but are not supported in ACalc.
ValueError	unsupported function call in equation in ACalc compile.	An unsupported function call was made.
SyntaxError	parsing error in ACalc compile: 'x'	An unspecified parsing error occured.
ValueError	unknown compile error in ACalc compile.	An unspecified compile error occured.

ValueError	stack overflow or underflow in ACalc compile.	The equation was checked before execution, and a stack overflow was detected. The equation may be too complex.

Run Time

These are the exceptions which can occur during the execution phase. All errors except for the arithmetic overflow errors should have been detected during the compile phase. These run-time checks are in addition to the compile checks.

Exception type	Text	Description
ValueError	ACalc vm stack overflow or underflow.	A stack overflow was detected.
ValueError	ACalc vm uknown op code.	An unknown opcde was detected.
ValueError	ACalc vm variable array overflow.	The variable array index overflowed.
ValueError	ACalc vm operator is invalid for array type.	An operator used was invalid for the array type.

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 or opcode.
- The data type of the array.
- Function options. Turning overflow 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. Overflow 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.

Amap

opcode	b	B	h	H	i	I	l	L	q	Q	f	d
af_add	122	130	125	136	95	76	60	61	61	53	41	39
af_div	58	55	61	58	58	54	59	50	62	47	78	69

af_div_r	56	62	63	63	68	53	59	44	66	44	71	58
af_floordiv	34	30	26	36	35	32	34	28	42	28	54	47
af_floordiv_r	26	35	29	38	35	30	34	26	35	27	51	40
af_mod	32	34	23	40	38	29	33	26	35	28	27	27
af_mod_r	33	30	31	37	30	27	32	26	30	28	20	18
af_mult	92	136	84	130	87	106	60	61	57	51	47	39
af_neg	109		132		115		67		63		39	35
af_pow	52	49	47	45	34	30	19	16	18	16	15	14
af_pow_r	47	41	43	40	33	30	19	18	18	17	2.6	4.0
af_sub	136	135	124	124	108	91	63	57	70	50	39	40
af_sub_r	131	142	104	108	108	86	61	44	61	48	44	39
af_and	155	238	235	161	150	122	72	71	79	66		
af_or	151	234	238	161	147	124	78	73	75	70		
af_xor	150	235	227	162	161	129	89	76	82	72		
af_invert	180	190	282	300	210	193	102	96	114	107		
af_eq	159	182	143	142	133	99	72	59	75	58	127	83
af_gt	151	154	147	146	139	105	70	58	79	62	157	84
af_gte	147	201	146	147	147	105	70	60	76	57	158	104
af_lt	137	188	160	145	137	108	73	60	75	60	170	97
af_lte	139	155	133	158	138	117	74	62	77	64	175	107
af_ne	161	194	151	172	134	128	76	68	76	63	163	115
af_lshift	177	240	183	164	192	118	108	83	100	91		
af_lshift_r	181	254	197	175	185	141	102	77	94	84		
af_rshift	170	238	159	150	191	124	92	72	95	77		
af_rshift_r	170	217	157	187	194	129	88	70	92	84		
af_abs	101		100		94		70		72		139	76
math_acos											12	12
math_acosh											6.7	5.2
math_asin											13	11
math_asinh											6.7	6.8
math_atan											12	12
math_atan2											8.4	8.4
math_atan2_r											11	7.2
math_atanh											6.6	7.4
math_ceil											69	67
math_copysign											73	65
math_cos											16	8.4
math_cosh											11	7.2

math_degrees											58	47
math_erf											15	13
math_erfc											8.4	7.6
math_exp											12	8.9
math_expm1											7.1	6.9
math_fabs											64	65
math_factorial	73	41	74	93	75	62	65	59	77	56		
math_floor											60	63
math_fmod											12	11
math_fmod_r											31	30
math_gamma											1.1	1.3
math_hypot											19	14
math_hypot_r											21	13
math_isinf											53	54
math_isnan											57	54
math_ldexp											58	54
math_lgamma											8.8	6.1
math_log											15	8.9
math_log10											9.8	7.0
math_log1p											9.0	8.6
math_pow											21	20
math_pow_r											3.7	6.0
math_radians											55	47
math_sin											15	8.4
math_sinh											5.0	5.3
math_sqrt											48	41
math_tan											7.0	5.6
math_tanh											6.1	5.6
math_trunc											49	42
aops_subst_gt	160	185	193	161	193	139	99	79	98	90	212	89
aops_subst_gte	147	181	177	178	150	137	82	66	74	61	143	73
aops_subst_lt	180	200	200	180	176	149	68	72	63	62	165	67
aops_subst_lte	174	174	172	183	160	145	66	58	65	60	141	61

Stat	Value
Average:	84
Maximum:	300
Minimum:	1.1

Array size:	100000

ACalc

opcode	b	B	h	H	i	I	l	L	q	Q	f	d
add	20	21	21	22	21	18	20	16	21	18	18	21
sub	22	26	23	25	24	21	25	20	24	21	17	20
mult	12	14	7.2	13	5.2	7.3	3.3	4.8	3.2	4.9	20	19
div	25	36	37	38	37	28	25	21	24	23	32	36
floordiv	19	21	20	21	21	17	18	13	17	14	26	27
mod	17	22	13	20	19	19	20	15	21	16	16	15
uadd	44	57	54	59	51	45	57	35	48	37	17	19
usub	33		29		33		30		31		20	20
pow	33	34	30	30	25	22	16	14	16	15	11	11
bitand	27	31	31	36	31	26	28	27	29	23		
bitor	27	31	30	28	31	25	28	22	29	23		
bitxor	29	32	38	33	29	29	33	29	28	23		
invert	56	58	59	59	62	44	57	45	63	53		
lshift	30	32	29	34	33	27	32	30	31	25		
rshift	30	35	31	31	30	23	30	23	30	24		
abs	39	65	38	53	38	44	35	51	34	55	40	35
math_acos											9.6	9.6
math_acosh											6.3	5.0
math_asin											11	10
math_asinh											5.6	6.6
math_atan											10	9.7
math_atan2											8.1	7.4
math_atanh											6.4	7.0
math_ceil											40	43
math_copysign											33	36
math_cos											12	7.9
math_cosh											8.9	6.6
math_degrees											30	29
math_erf											13	12
math_erfc											7.6	7.0
math_exp											10	7.9
math_expm1											6.5	6.5
math_fabs											66	56

math_factorial	35	40	36	40	37	27	34	31	39	26		
math_floor											39	39
math_fmod											9.3	11
math_gamma											1.1	1.3
math_hypot											15	11
math_ldexp											32	32
math_lgamma											7.1	5.6
math_log											12	8.2
math_log10											8.5	6.8
math_log1p											7.6	8.3
math_pow											16	16
math_radians											27	27
math_sin											12	7.8
math_sinh											4.6	4.9
math_sqrt											31	27
math_tan											6.4	5.6
math_tanh											5.2	4.8
math_trunc											31	33

Stat	Value
Average:	25
Maximum:	66
Minimum:	1.1
Array size:	100000

Other Functions

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	11	8.8	8.7	8.8	8.2	8.7	6.5	7.8	6.7	7.9	15	8.2
aany	9.8	7.2	5.9	7.2	5.7	7.4	6.0	6.3	5.9	6.2	11	6.5
afilter	224	222	215	212	143	99	87	60	86	59	157	88
amax	21	28	22	24	19	20	12	13	13	13	30	23
amin	20	29	20	29	20	18	12	12	12	12	29	23
asum	6.1	8.5	6.6	8.1	7.1	8.7	5.7	6.4	5.7	6.3	2.8	2.8
compress	35	38	35	36	36	18	31	16	30	16	33	30
count	221	202	207	207	111	81	64	46	64	47	105	85
cycle	94	97	92	96	81	57	54	37	54	38	35	35
dropwhile	88	85	87	86	85	61	53	38	53	39	87	52
findindex	15	15	15	14	18	18	10	12	10	13	15	12

findindices	21	21	21	21	20	21	19	20	19	20	33	28
repeat	131	129	120	117	79	22	47	13	47	13	107	62
takewhile	239	179	173	139	157	85	90	61	90	61	123	89

Stat	Value
Average:	51
Maximum:	239
Minimum:	2.8
Array size:	1000000