ArrayFunc

Authors: Michael Griffin
Version: 0.9.5 for 2015-08-29

Language: Python 3.x

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 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.

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

Note: the 'q' and 'Q' array types and therefor limit attributes may not be present on all platforms.

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
```

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			

Notes

- The regular and floor division operators (af_div, af_div_r, af_floordiv, and af_floordiv_r) 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 (af_pow, af_pow_r) operators will not accept a negative exponent for integers, as the result would be a fractional number which is not compatible with an integer array.

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.

Platform Compiler Support

Amap and Amapi 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.

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.

Amap

opcode	b	B	h	H	i	I	l	L	q	Q	f	d
af_add	150	181	170	168	155	81	71	47	56	50	41	40
af_div	77	71	81	81	82	58	76	55	71	56	86	82
af_div_r	77	77	87	86	86	64	83	53	81	56	74	65
af_floordiv	39	33	26	40	37	28	40	27	34	27	49	50
af_floordiv_r	30	40	33	41	40	30	39	25	35	27	43	47
af_mod	33	37	27	39	39	28	42	26	33	27	25	27
af_mod_r	36	37	38	39	37	28	39	28	35	27	23	20
af_mult	108	150	100	155	102	89	71	49	62	48	46	41
af_neg	172		180		142		81		67		47	42
af_pow	75	74	67	64	49	43	28	24	26	24	14	13
af_pow_r	67	54	59	61	47	41	27	24	25	23	2.5	4.1
af_sub	173	178	168	166	123	93	88	51	62	66	42	39
af_sub_r	145	169	134	140	123	89	76	48	65	52	48	39
af_and	185	298	279	211	163	133	89	64	79	64		
af_or	174	280	270	193	170	124	89	63	70	60		

af_xor	198	309	294	208	158	129	96	55	75	60		
af_invert	235	244	365	364	219	222	116	88	114	99		
af_eq	210	244	191	186	152	115	74	58	73	63	126	72
af_gt	168	194	178	183	151	118	85	63	70	55	147	83
af_gte	165	244	180	183	153	111	85	58	72	62	148	90
af_lt	172	231	175	188	153	116	84	57	68	58	115	70
af_lte	168	197	182	181	163	113	51	55	73	57	152	87
af_ne	201	233	182	172	155	118	87	58	70	59	151	88
af_lshift	212	298	204	200	202	128	110	72	83	68		
af_lshift_r	210	278	205	192	188	130	83	64	81	65		
af_rshift	209	288	198	198	209	130	94	62	82	66		
af_rshift_r	217	265	204	197	215	142	92	63	80	65		
af_abs	128		136		125		82		61		121	75
math_acos											13	12
math_acosh											8.0	6.2
math_asin											17	14
math_asinh											7.5	7.4
math_atan											14	14
math_atan2											9.9	10
math_atan2_r											13	8.4
math_atanh											7.9	8.6
math_ceil											77	75
math_copysign											78	78
math_cos											20	10
math_cosh											11	8.6
math_degrees											59	49
math_erf											17	15
math_erfc											10	8.5
math_exp											15	12
math_expm1											8.0	8.2
math_fabs											81	75
math_factorial	85	93	82	108	86	73	80	66	80	69		
math_floor											76	74
math_fmod											12	13
math_fmod_r											12	12
math_gamma											1.3	1.5
math_hypot											19	14
math_hypot_r											21	15

math_isinf											59	51
math_isnan											66	52
math_ldexp											70	67
math_lgamma											10.0	6.9
math_log											17	9.9
math_log10											12	7.9
math_log1p											11	10
math_pow											24	23
math_pow_r											4.2	6.8
math_radians											61	50
math_sin											18	9.4
math_sinh											6.1	5.5
math_sqrt											59	48
math_tan											7.5	6.5
math_tanh											7.0	6.4
math_trunc											58	48
aops_subst_gt	199	199	222	233	195	174	108	75	88	70	181	91
aops_subst_gte	188	208	222	228	187	143	102	72	84	73	164	82
aops_subst_lt	216	244	234	220	171	162	101	75	87	74	149	87
aops_subst_lte	178	219	221	203	175	143	97	69	93	74	153	83

Stat	Value
Average:	95
Maximum:	365
Minimum:	1.3
Array size:	100000

Other Functions

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	9.9	13	14	12	10	12	5.9	7.3	6.5	7.9	14	7.5
aany	9.8	13	10	12	9.2	13	6.5	6.9	6.4	6.6	13	7.9
afilter	280	266	257	271	182	132	109	80	106	72	183	110
amax	24	34	22	32	21	22	13	14	14	14	33	23
amin	23	35	23	31	23	24	14	15	13	14	31	24
asum	9.6	12	9.6	11	9.7	13	8.8	9.6	8.2	6.9	3.6	3.9
compress	53	57	53	53	52	36	42	31	45	31	46	40
count	267	263	255	266	134	89	80	56	70	55	112	95
cycle	111	111	109	106	89	61	66	41	56	39	36	36

dropwhile	133	135	129	130	111	84	63	48	64	47	113	61
findindex	21	22	21	21	17	18	12	14	12	13	15	12
findindices	37	37	36	51	32	33	20	22	22	22	34	27
repeat	126	129	118	122	76	14	43	9.8	44	10	109	62
takewhile	231	296	248	225	186	132	97	80	101	72	160	102

Stat	Value
Average:	63
Maximum:	296
Minimum:	3.6
Array size:	1000000