BytesFunc

Authors: Michael Griffin
Version: 3.3.0 for 2022-07-11

Language: Python 3.6 or later

Table of Contents

Introduction **3**

Function Summary **3**

Brief Description 3

Python Equivalent 4

Description **5**

Parameters 5

Parameter Formats 5

Function Documentation Details 5

and_ 5

ball 6

bany 6

bmax 7

bmin 7

bsum 8

eq 8

findindex 9

ge 9

gt 10

invert 11

le 11

lshift 12

lt 12

ne 13

or_ 14

rshift 14

xor 15

Parameter Details 15

Comparison Operators 15

Numeric Parameters 16

Using Less than the Entire Sequence 16

Suppressing or Ignoring Math Errors 16

Differences with Native Python 16

SIMD Support **16**

General 16

Disabling SIMD 17

Platform Support 17

Raspberry Pi 32 versus 64 bit 17
SIMD Function Support 17
SIMD Support Attributes 18
Performance **18**
Variables affecting Performance 18
Typical Performance Readings 19
x86-64 Benchmarks 19
ARMv7 Benchmarks 20
ARMv8 Benchmarks 21
Platform Effects 21
Platform support **22**
List of tested Operation Systems, Compilers, and CPU Architectures 22
Platform Oddities 22

Introduction

The BytesFunc module provides high speed array processing functions for use with Python 'bytes' and 'bytearray' objects. These functions are patterned after the functions in the standard Python "operator" module together with some additional ones from other sources.

The purpose of these functions is to perform mathematical calculations on "bytes" and "bytearray" objects significantly faster than using native Python.

Function Summary

The compare operators used for 'ball', 'bany', and 'findindex' are examples only, and other compare operations are available. Many functions will accept other parameter combinations of sequences and numeric parameters. See the details for each function for what parameter combinations are valid.

Brief Description

Function	Equivalent to
and_	Perform a bitwise AND across the sequence.
ball	True if all elements of the sequence meet the match criteria.
bany	True if any elements of the sequence meet the match criteria.
bmax	Return the maximum value in the sequence.
bmin	Return the minimum value in the sequence.
bsum	Return the sum of the sequence.
eq	True if all elements of the sequence equal the compare value.
findindex	Returns the index of the first value in an array to meet the specified criteria.

ge	True if all elements of the sequence are greater than or equal to the compare value.
gt	True if all elements of the sequence are greater than the compare value.
invert	Perform a bitwise invert across the sequence.
le	True if all elements of the sequence are less than or equal to the compare value.
lshift	Perform a bitwise left shift across the sequence.
lt	True if all elements of the sequence are less than the compare value.
ne	True if all elements of the sequence are not equal the compare value.
or_	Perform a bitwise OR across the sequence.
rshift	Perform a bitwise right shift across the sequence.
xor	Perform a bitwise XOR across the sequence.

Python Equivalent

Function	Equivalent to
and_	[x & param for x in sequence1]
ball	all([(x > param) for x in array])
bany	any([(x > param) for x in array])
bmax	max(sequence)
bmin	min(sequence)
bsum	sum(sequence)
eq	all([x == param for x in sequence])
findindex	[x for x,y in enumerate(array) if y > param][0]
ge	all([x >= param for x in sequence])
gt	all([x > param for x in sequence])
invert	[~x for x in sequence1]
le	all([x <= param for x in sequence])
lshift	[x << param for x in sequence1]
lt	all([x < param for x in sequence])
ne	all([x != param for x in sequence])
or_	[x | param for x in sequence1]
rshift	[x >> param for x in sequence1]
xor	[x ^ param for x in sequence1]

Description

Parameters

Parameter Formats

Parameters come in several forms.

- Sequences. Sequences are either "bytes" or "bytearray" objects. Bytes sequences are immutable and must not be used for output destinations. Bytearray sequences are mutable, and may be used for inputs or outputs.
- Numeric parameters. Numeric input parameters are individual integers and must be in the range of 0 to 255.
- Comparison operators. Comparison operators are unicode strings in the form used by Python for compare operations. These must be quoted strings, and not bare Python symbols. See the section below for a list of these.
- Sequence length control. Sequence length control allows only part of a sequence to be used as an input. See the section below for details.
- Overflow detection disable. Overflow detection control is used for disable integer overflow. See the section below for details.

Example:

```
sequence = bytes([1, 2, 5, 99, 8])
# Find the maximum value and return it. The answer should be 99.
result = bytesfunc.bmax(sequence)
```

Example:

```
sequence1 = bytes([1, 2, 5, 99, 8])
sequence2 = bytearray([0, 0, 0, 0, 0])
# Xor each element in sequence1 with '7', and write the output to
# sequence2. Sequence2 should be bytearray(b'\x06\x05\x02d\x0f').
bytesfunc.xor(sequence1, 7, sequence2)
```

Example:

```
sequence1 = bytes([1, 2, 5, 99, 8, 101])
# Find the first index of sequence1 which is greater than or equal to 99.
# The answer should be 3.
result = bytesfunc.findindex('>=', sequence, 99)
```

Function Documentation Details

*and_*

Calculate and_ over the values in a bytes or bytearray object.

Equivalent to:	[x & param for x in sequence1]
or	[param & x for x in sequence1]
or	[x & y for x,y in zip(sequence1, sequence2)]

Call formats:

```
and_(sequence1, param)
and_(sequence1, param, outpsequence)
and_(param, sequence1)
and_(param, sequence1, outpsequence)
and_(sequence1, sequence2)
and_(sequence1, sequence2, outpsequence)
and_(sequence1, param, maxlen=y)
and_(sequence1, param, nosimd=False)
```

- sequence1 - The first input data bytes or bytearray sequence to be examined. If no output sequence is provided the results will overwrite the input data.
- param - A non-sequence numeric parameter.
- sequence2 - A second input data sequence. Each element in this sequence is applied to the corresponding element in the first sequence.
- outpsequence - The output sequence. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

ball

Calculate ball over the values a bytes or bytearray object.

Equivalent to:	all([(x > param) for x in array])

Call formats:

```
result = ball(opstr, sequence, param)
result = ball(opstr, sequence, param, maxlen=y)
result = ball(opstr, sequence, param, nosimd=False)
```

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

  These are: '==', '>', '>=', '<', '<=', '!='.
- sequence - An input bytes or bytearray to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If any comparison operations result in true, the return value will be true. If all of them result in false, the return value will be false.

bany

Calculate bany over the values a bytes or bytearray object.

Equivalent to:	any([(x > param) for x in array])

Call formats:

```
result = bany(opstr, sequence, param)
result = bany(opstr, sequence, param, maxlen=y)
result = bany(opstr, sequence, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- sequence - An input bytes or bytearray to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

bmax

Calculate bmax over the values in an array.

Equivalent to:	max(sequence)

Call formats:

```
result = bmax(sequence)
result = bmax(sequence, maxlen=y)
result = bmax(sequence, nosimd=False)
```

- sequence - The input bytes or bytearray to be examined.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The maximum of all the values in the sequence.

bmin

Calculate bmin over the values in an array.

Equivalent to:	min(sequence)

Call formats:

```
result = bmin(sequence)
result = bmin(sequence, maxlen=y)
result = bmin(sequence, nosimd=False)
```

- sequence - The input bytes or bytearray to be examined.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The minimum of all the values in the sequence.

bsum

Calculate the arithmetic sum of an bytes or bytearray sequence.

Equivalent to:	sum(sequence)

Call formats:

```
result = bsum(sequence)
result = bsum(sequence, maxlen=y)
result = bsum(sequence, matherrors=False)
result = bsum(sequence, nosimd=False)
```

- sequence - An input bytes or bytearray to be examined.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- matherrors - If True, checks for numerical errors including integer overflow are ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - The sum of the sequence.

eq

Calculate eq over the values in a bytes or bytearray object.

Equivalent to:	all([x == param for x in sequence])
or	all([param == x for x in sequence])
or	all([x == y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = eq(sequence1, param)
result = eq(param, sequence1)
result = eq(sequence1, sequence2)
result = eq(sequence1, param, maxlen=y)
result = eq(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.

- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

findindex

Calculate findindex over the values a bytes or bytearray object.

Equivalent to:	[x for x,y in enumerate(array) if y > param][0]

Call formats:

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

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

  These are: '==', '>', '>=', '<', '<=', '!='.
- sequence - An input bytes or bytearray to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - The resulting index. This will be negative if no match was found.

ge

Calculate ge over the values in a bytes or bytearray object.

Equivalent to:	all([x >= param for x in sequence])
or	all([param >= x for x in sequence])
or	all([x >= y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = ge(sequence1, param)
result = ge(param, sequence1)
```

```
result = ge(sequence1, sequence2)
result = ge(sequence1, param, maxlen=y)
result = ge(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.
- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

gt

Calculate gt over the values in a bytes or bytearray object.

Equivalent to:	all([x > param for x in sequence])
or	all([param > x for x in sequence])
or	all([x > y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = gt(sequence1, param)
result = gt(param, sequence1)
result = gt(sequence1, sequence2)
result = gt(sequence1, param, maxlen=y)
result = gt(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.
- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).

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

invert

Calculate invert over the values in a bytes or bytearray object.

Equivalent to:	[~x for x in sequence1]

Call formats:

```
invert(sequence1)
invert(sequence1, outpseq)
invert(sequence1, maxlen=y)
invert(sequence1, nosimd=False)
```

- sequence1 - The input bytes or bytearray to be examined. If no output bytearray is provided the results will overwrite the input data, in which case it must be a bytearray.
- outpseq - The output bytearray. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

le

Calculate le over the values in a bytes or bytearray object.

Equivalent to:	all([x <= param for x in sequence])
or	all([param <= x for x in sequence])
or	all([x <= y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = le(sequence1, param)
result = le(param, sequence1)
result = le(sequence1, sequence2)
result = le(sequence1, param, maxlen=y)
result = le(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.
- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.

- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

lshift

Calculate lshift over the values in a bytes or bytearray object.

Equivalent to:	[x << param for x in sequence1]
or	[param << x for x in sequence1]
or	[x << y for x,y in zip(sequence1, sequence2)]

Call formats:

```
lshift(sequence1, param)
lshift(sequence1, param, outpsequence)
lshift(param, sequence1)
lshift(param, sequence1, outpsequence)
lshift(sequence1, sequence2)
lshift(sequence1, sequence2, outpsequence)
lshift(sequence1, param, maxlen=y)
lshift(sequence1, param, nosimd=False)
```

- sequence1 - The first input data bytes or bytearray sequence to be examined. If no output sequence is provided the results will overwrite the input data.
- param - A non-sequence numeric parameter.
- sequence2 - A second input data sequence. Each element in this sequence is applied to the corresponding element in the first sequence.
- outpsequence - The output sequence. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

lt

Calculate lt over the values in a bytes or bytearray object.

Equivalent to:	all([x < param for x in sequence])
or	all([param < x for x in sequence])
or	all([x < y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = lt(sequence1, param)
result = lt(param, sequence1)
result = lt(sequence1, sequence2)
result = lt(sequence1, param, maxlen=y)
result = lt(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.
- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

ne

Calculate ne over the values in a bytes or bytearray object.

Equivalent to:	all([x != param for x in sequence])
or	all([param != x for x in sequence])
or	all([x != y for x,y in zip(sequence1, sequence2)])

Call formats:

```
result = ne(sequence1, param)
result = ne(param, sequence1)
result = ne(sequence1, sequence2)
result = ne(sequence1, param, maxlen=y)
result = ne(sequence1, param, nosimd=False)
```

- sequence1 - An input bytes or bytearray to be examined.
- sequence2 - An input bytes or bytearray to be examined.
- param - A integer numeric input parameter in the range 0 - 255.
- The first and second parameters are compared to each other. If one parameter is a sequence and the other is an integer, the integer is compared to each element in the sequence. If both parameters are sequences, each element of one sequence is compared to the corresponding element of the other sequence.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

*or_*

Calculate or_ over the values in a bytes or bytearray object.

Equivalent to:	[x | param for x in sequence1]
or	[param | x for x in sequence1]
or	[x | y for x,y in zip(sequence1, sequence2)]

Call formats:

```
or_(sequence1, param)
or_(sequence1, param, outpsequence)
or_(param, sequence1)
or_(param, sequence1, outpsequence)
or_(sequence1, sequence2)
or_(sequence1, sequence2, outpsequence)
or_(sequence1, param, maxlen=y)
or_(sequence1, param, nosimd=False)
```

- sequence1 - The first input data bytes or bytearray sequence to be examined. If no output sequence is provided the results will overwrite the input data.
- param - A non-sequence numeric parameter.
- sequence2 - A second input data sequence. Each element in this sequence is applied to the corresponding element in the first sequence.
- outpsequence - The output sequence. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

rshift

Calculate rshift over the values in a bytes or bytearray object.

Equivalent to:	[x >> param for x in sequence1]
or	[param >> x for x in sequence1]
or	[x >> y for x,y in zip(sequence1, sequence2)]

Call formats:

```
rshift(sequence1, param)
rshift(sequence1, param, outpsequence)
rshift(param, sequence1)
rshift(param, sequence1, outpsequence)
rshift(sequence1, sequence2)
rshift(sequence1, sequence2, outpsequence)
rshift(sequence1, param, maxlen=y)
rshift(sequence1, param, nosimd=False)
```

- sequence1 - The first input data bytes or bytearray sequence to be examined. If no output sequence is provided the results will overwrite the input data.

- param - A non-sequence numeric parameter.
- sequence2 - A second input data sequence. Each element in this sequence is applied to the corresponding element in the first sequence.
- outpsequence - The output sequence. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

xor

Calculate xor over the values in a bytes or bytearray object.

Equivalent to:	[x ^ param for x in sequence1]
or	[param ^ x for x in sequence1]
or	[x ^ y for x,y in zip(sequence1, sequence2)]

Call formats:

```
xor(sequence1, param)
xor(sequence1, param, outpsequence)
xor(param, sequence1)
xor(param, sequence1, outpsequence)
xor(sequence1, sequence2)
xor(sequence1, sequence2, outpsequence)
xor(sequence1, param, maxlen=y)
xor(sequence1, param, nosimd=False)
```

- sequence1 - The first input data bytes or bytearray sequence to be examined. If no output sequence is provided the results will overwrite the input data.
- param - A non-sequence numeric parameter.
- sequence2 - A second input data sequence. Each element in this sequence is applied to the corresponding element in the first sequence.
- outpsequence - The output sequence. This parameter is optional.
- maxlen - Limit the length of the sequence 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 sequence is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Parameter Details

Comparison Operators

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

Operator	Description
'<'	Less than.
'<='	Less than or equal to.
'>'	Greater than.

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

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

Numeric Parameters

"Bytes" and "bytearray" objects are sequences of 8 bit bytes with each element being in the range of 0 to 255. When a function accepts a non-sequence numeric parameter, this must also be in the range of 0 to 255.

Using Less than the Entire Sequence

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

```
x = bytes([20,21,22,23,24,25])
result = bytesfunc.bmax(x, maxlen=3)
```

Suppressing or Ignoring Math Errors

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

```
x = bytes([20,21,22,23,24,250,250])
result = bytesfunc.sum(x, matherrors=True)
```

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. Benchmark your calculation before deciding if this is worth while.

Differences with Native Python

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

Think of BytesFunc 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.

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 may make use of these instructions to speed up execution.

Disabling SIMD

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.

Platform Support

SIMD instructions are presently supported only on the following:

- 64 bit x86 (i.e. AMD64) using GCC.
- 32 bit ARMv7 using GCC (tested on Raspberry Pi 3).
- 64 bit ARMv8 AARCH64 using GCC (tested on Raspberry Pi 4).

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

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

Raspberry Pi 32 versus 64 bit

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

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

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

The "setup.py" file uses platform detection code to determine which ARM CPU and mode it is running on. Due to the availability of hardware for testing, this code is tailored to the Raspberry Pi 3 and Raspberry Pi 4 and the operating systems listed. This code then selects the appropriate compiler arguments to pass to the setup routines to tell the compiler what mode to compile for.

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

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

SIMD Function Support

The following table shows which functions are supported by SIMD on which CPU architectures.

Function	x86	ARMv7	ARMv8
and_	X	X	X
ball	X	X	X
bany	X	X	X
bmax	X	X	X
bmin	X	X	X
bsum		X	X

eq	X	X	X
findindex	X	X	X
ge	X	X	X
gt	X	X	X
invert	X	X	X
le	X	X	X
lshift	X	X	X
lt	X	X	X
ne	X	X	X
or_	X	X	X
rshift	X	X	X
xor	X	X	X

SIMD Support Attributes

"Simdsupport" provides information on the SIMD level compiled into this version of the library. There are two attributes, 'hassimd' and 'simdarch'.

- 'hassimd' is TRUE if the CPU supports the required SIMD features.
- **'simdarch' contains a string indicating the CPU architecture the library** was compiled for.

Example:

```
>>> bytesfunc.simdsupport.hassimd
True
```

Example:

```
>>> bytesfunc.simdsupport.simdarch
'x86_64'
```

This was created primarily for unit testing and benchmarking and should not be considered to be a permanent or stable part of the library.

Performance

Variables affecting Performance

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

- The function.
- Function options. Turning checking off will result in faster performance.
- The data in the sequence and the parameters.
- The size of the sequence.

- The platform, including CPU type (e.g. x86 or ARM), operating system, and compiler.

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

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

Note: Some more complex BytesFunc functions do not work exactly the same way as the native Python equivalents. This means that the benchmark results should be taken as general guidelines rather than precise comparisons.

Typical Performance Readings

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

The Bytesfunc versus Python factor of 100.0 means the bytesfunc version ran 100 times faster than in native Python on that platform. Benchmarks for different hardware and platforms cannot be compared via this benchmark in terms of absolute performance as these are relative, not absolute numbers.

An SIMD versus non-SIMD factor of 10.0 means the SIMD version was 10 times faster than the non-SIMD version. An SIMD versus non-SIMD factor of 0.0 means the function did not support SIMD on the tested platform.

x86-64 Benchmarks

The following tests were conducted on an x86-64 CPU.

Relative Performance - Python Time / Bytesfunc Time.

function	Bytesfunc vs Python	SIMD vs non-SIMD
and_	911.2	9.3
ball	735.3	15.6
bany	737.1	15.9
bmax	78.9	2.7
bmin	76.7	2.6
bsum	16.0	
eq	751.3	15.7
findindex	1055.4	14.8
ge	744.6	14.9
gt	592.6	11.7
invert	726.3	9.6
le	740.4	15.0
lshift	1285.9	6.3
lt	578.2	11.7
ne	758.3	15.3

or_	946.7	10.2
rshift	892.1	6.2
xor	1007.0	9.2

Stat	Value
Average:	702
Maximum:	1286
Minimum:	16.0
Array size:	100000

ARMv7 Benchmarks

The following tests were conducted on an ARM CPU in 32 bit mode (ARMv7) on a Raspberry Pi 3.

Relative Performance - Python Time / Bytesfunc Time.

function	Bytesfunc vs Python	SIMD vs non-SIMD
and_	1100.2	3.7
ball	343.5	2.6
bany	331.3	2.4
bmax	268.2	5.0
bmin	265.0	5.0
bsum	75.5	3.3
eq	350.7	2.6
findindex	511.0	3.2
ge	361.8	2.6
gt	362.3	2.6
invert	897.2	3.7
le	362.3	2.6
lshift	1351.8	4.2
lt	362.5	2.6
ne	332.4	2.4
or_	1109.0	3.7
rshift	949.2	4.3
xor	1141.1	3.7

Stat	Value
Average:	582
Maximum:	1352
Minimum:	75.5
Array size:	100000

ARMv8 Benchmarks

The following tests were conducted on an ARM CPU in 64 bit mode (ARMv8) on a Raspberry Pi 4.

Relative Performance - Python Time / Bytesfunc Time.

function	Bytesfunc vs Python	SIMD vs non-SIMD
and_	769.2	6.5
ball	500.2	6.0
bany	532.0	6.1
bmax	358.4	13.8
bmin	372.8	13.8
bsum	113.6	6.3
eq	501.8	6.0
findindex	730.8	6.1
ge	533.0	6.0
gt	528.6	6.0
invert	853.3	9.2
le	537.4	6.0
lshift	1164.8	6.7
lt	582.1	6.0
ne	569.0	6.1
or_	896.8	6.2
rshift	657.9	5.9
xor	761.8	6.8

Stat	Value
Average:	609
Maximum:	1165
Minimum:	113.6
Array size:	100000

Platform Effects

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

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

Platform support

List of tested Operation Systems, Compilers, and CPU Architectures

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

OS	Hardware	Bits	Compiler	Python Version
Ubuntu 20.04 LTS	x86_64	64	GCC	3.8
Ubuntu 22.04	x86_64	64	GCC	3.10
Debian 11	i686	32	GCC	3.9
Debian 11	x86_64	64	GCC	3.9
OpenSuse 15.3	x86_64	64	GCC	3.6
Alma 9	x86_64	64	GCC	3.9
Alpine 3.16.0	i686	32	GCC	3.10
FreeBSD 13	x86_64	64	LLVM	3.8
OpenBSD 7.1	x86_64	64	LLVM	3.9
MS Windows 10	x86_64	64	MS VS C v.1929	3.10
MS Windows 11	x86_64	64	MS VS C v.1929	3.10
Raspberry Pi 2022-04-04	RPi 3	32	GCC	3.9
Ubuntu 22.04	RPi 4	64	GCC	3.10

- The Rasberry Pi 3 tests were conducted on a Raspberry Pi 3 ARM CPU running in 32 bit mode.
- The Ubuntu ARM tests were conducted on a Raspberry Pi 4 ARM CPU running in 64 bit mode.
- All others were conducted using VMs running on x86 hardware.

Platform Oddities

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

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

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

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

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

The conclusion is that bit shift operations which use a shift amount which is not in the range of 0 to "maximum number" may produce undefined results. So valid bit shift amounts should be 0 to 7.