Log to the base 2 in python

How should I compute log to the base two in python. Eg. I have this equation where I am using log base 2

```
import math
e = -(t/T)* math.log((t/T)[, 2])
```

python logarithm

edited Jul 12 '12 at 14:10
Nicolae Surdu
3,915 6 46 75

asked Sep 15 '10 at 16:15
Compuser7
1,368 6 21 28

2 What you have should work if you take the square brackets out around the ", 2" in the `math.log()` call. Have you tried it? – martineau Sep 15 '10 at 18:44

2 nice entropy calculation – Muhammad Alkarouri Sep 16 '10 at 1:36

math.log(value, base) – Valentin Heinitz Jan 7 '15 at 22:10

9 Answers

It's good to know that

$$\log_b(a) = \frac{\log(a)}{\log(b)}$$

but also know that `math.log` takes an optional second argument which allows you to specify the base:

```
In [22]: import math

In [23]: math.log?
Type:       builtin_function_or_method
Base Class: <type 'builtin_function_or_method'>
String Form:    <built-in function log>
Namespace:  Interactive
Docstring:
    log(x[, base]) -> the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.


In [25]: math.log(8,2)
Out[25]: 3.0
```

answered Sep 15 '10 at 16:23
unutbu
428k 62 843 950

3 +1. Change-of-base formula FTW – Matt Ball Sep 15 '10 at 17:01

1 `base` argument added in version 2.3, btw. – Joe Koberg Sep 15 '10 at 18:09

2 What is this '?' syntax ? I can't find reference for it. – wap26 Apr 30 '13 at 13:59

7 @wap26: Above, I'm using the IPython interactive interpreter. One of its features (accessed with the `?`) is dynamic object introspection. – unutbu Apr 30 '13 at 17:51

If all you need is the **integer part** of log base 2, `math.frexp()` could be pretty efficient:

```
import math

log2int_slow = int(math.floor(math.log(x, 2.0)))
log2int_fast = math.frexp(x)[1]-1
```

The C function it calls just grabs and tweaks the exponent.

Splainin: frexp() returns a tuple (mantissa, exponent). So `[1]` gets the exponent part. For integral powers of 2 the exponent is one more than you might expect. For example 32 is stored as $0.5x2^{6}$. This explains the `-1` above. Also works for 1/32 which is stored as $0.5x2^{-4}$.

If both input and output are integers, the integer method `.bit_length()` could be even more efficient:

```
log2int_faster = int(x).bit_length()-1
```

`-1` because 2^{n} requires n+1 bits. This is the only option that works for very large integers, e.g. `2**10000` .

All these options floor the log toward negative infinity, so $\log_2 31$ is 4 not 5.

edited Mar 23 at 11:05

answered Jan 19 '15 at 20:41

BobStein-VisiBone
3,423 1 32 45

1 Interesting. So you're subtracting 1 there because the mantissa is in the range [0.5, 1.0)? I would give this one a few more upvotes if I could. – LarsH Feb 23 '15 at 11:49

1 Exactly right @LarsH. 32 is stored as $0.5x2^{6}$ so if you want $\log_2 32=5$ you need to **subtract 1**. Also true for 1/32 which is stored as $0.5x2^{-4}$. – BobStein-VisiBone Feb 23 '15 at 14:10

Using numpy:

```
In [1]: import numpy as np

In [2]: np.log2?
Type:           function
Base Class:     <type 'function'>
String Form:    <function log2 at 0x03049030>
Namespace:      Interactive
File:           c:\python26\lib\site-packages\numpy\lib\ufunclike.py
Definition:     np.log2(x, y=None)
Docstring:
    Return the base 2 logarithm of the input array, element-wise.

Parameters
----------
x : array_like
  Input array.
y : array_like
  Optional output array with the same shape as `x`.

Returns
-------
y : ndarray
  The logarithm to the base 2 of `x` element-wise.
  NaNs are returned where `x` is negative.

See Also
--------
log, log1p, log10
```

```
Examples
--------
>>> np.log2([-1, 2, 4])
array([ NaN,   1.,   2.])

In [3]: np.log2(8)
Out[3]: 3.0
```

answered Sep 15 '10 at 16:37

riza
4,932 6 21 24

http://en.wikipedia.org/wiki/Binary_logarithm

```
def lg(x, tol=1e-13):
  res = 0.0

  # Integer part
  while x<1:
    res -= 1
    x *= 2
  while x>=2:
    res += 1
    x /= 2

  # Fractional part
  fp = 1.0
  while fp>=tol:
    fp /= 2
    x *= x
    if x >= 2:
        x /= 2
        res += fp

  return res
```

answered Sep 15 '10 at 16:24

log0
7,811 2 16 50

Extra points for an algorithm that can be adapted to always give the correct integer part, unlike int(math.log(x, 2)) – user12861 Jan 10 '12 at 13:43

```
>>> def log2( x ):
...     return math.log( x ) / math.log( 2 )
...
>>> log2( 2 )
1.0
>>> log2( 4 )
2.0
>>> log2( 8 )
3.0
>>> log2( 2.4 )
1.2630344058337937
>>>
```

answered Sep 15 '10 at 16:19

puzz
636 4 7

This is built in to the `math.log` function. See unutbu's answer. – tgray Sep 15 '10 at 16:26

You're right, didn't know that - thanks ;) – puzz Sep 15 '10 at 16:34

If you are on python 3.4 or above then it already has a built-in function for computing log2(x)

```
import math
'finds log base2 of x'
answer = math.log2(x)
```

If you are on older version of python then you can do like this

```
import math
'finds log base2 of x'
answer = math.log(x)/math.log(2)
```

answered Nov 16 '14 at 9:02

akashchandrakar
824 2 10 37

logbase2(x) = log(x)/log(2)

answered Sep 15 '10 at 16:17

Conor
451 3 11

log_base_2(x) = log(x) /
log(2)

answered Sep 15 '10 at 16:16

Alexandre C.
39.7k 6 88 171

Don't forget that *log[base A] x = log[base B] x / log[base B] A*.

So if you only have `log` (for natural log) and `log10` (for base-10 log), you can use

```
myLog2Answer = log10(myInput) / log10(2)
```

answered Sep 15 '10 at 16:20

Platinum Azure
30.3k 2 78 115