How to print a number using commas as thousands separators

Asked 13 years, 4 months ago Modified 5 months ago Viewed 820k times

1145

How do I print an integer with commas as thousands separators?

```
1234567   →   1,234,567
```

It does not need to be locale-specific to decide between periods and commas.

python number-formatting

Share Improve this question Follow

edited Jun 4, 2022 at 22:27
Mateen Ulhaq
23.6k ● 16 ● 95 ● 132

asked Nov 30, 2009 at 23:11
Elias Zamaria
94.8k ● 32 ● 113 ● 148

30 Answers

Sorted by: Highest score (default)

2318

Locale unaware

```
'{:,}'.format(value)  # For Python ≥2.7
f'{value:,}'          # For Python ≥3.6
```

Locale aware

```
import locale
locale.setlocale(locale.LC_ALL, '')  # Use '' for auto, or force e.g. to
'en_US.UTF-8'

'{:n}'.format(value)  # For Python ≥2.7
f'{value:n}'          # For Python ≥3.6
```

Reference

Per Format Specification Mini-Language,

> The `','` option signals the use of a comma for a thousands separator. For a locale aware separator, use the `'n'` integer presentation type instead.

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

edited Jun 4, 2022 at 22:29
Mateen Ulhaq
23.6k ● 16 ● 95 ● 132

answered May 24, 2012 at 18:02
Ian Schneider
23.4k ● 2 ● 14 ● 5

31 Note that this won't be correct outside the US and few other places, in that case the chosen locale.format() is the correct answer. – Gringo Suave Sep 5, 2014 at 2:06

15 The keyword argument form: `{val:,}.format(val=val)` – CivFan Aug 25, 2015 at 18:48

23 Great thanks. For money amounts, with 2 decimal places - "{:,.2f}".format(value) – dlink Mar 21, 2016 at 18:44

17 In python 3.6 and up, f-strings add even more convenience. E.g. `f"{2 ** 64 - 1:,}"` – CJ Gaconnet Apr 19, 2017 at 15:03

5 How does the locale aware version work? (use the 'n' integer presentation type) – compie Dec 11, 2017 at 8:40

349

I'm surprised that no one has mentioned that you can do this with f-strings in Python 3.6+ as easy as this:

```
>>> num = 10000000
>>> print(f"{num:,}")
10,000,000
```

... where the part after the colon is the format specifier. The comma is the separator character you want, so `f"{num:_}"` uses underscores instead of a comma. Only "," and "_" is possible to use with this method.

This is equivalent of using `format(num, ",")` for older versions of python 3.

This might look like magic when you see it the first time, but it's not. It's just part of the language, and something that's commonly needed enough to have a shortcut available. To read more about it, have a look at the group subcomponent.

Share Improve this answer Follow

edited Sep 10, 2021 at 11:06

answered Jul 28, 2017 at 7:38
Emil Stenström
13k ● 8 ● 51 ● 75

14 This is easier than any of the higher voted answers, and requires no additional imports. – Z4-tier Jun 5, 2020 at 22:58

1 @NoName This answer also mentions how to do it with `str.format()` which works on all versions of python3, is almost as easy and doesn't use f-strings – Z4-tier Jun 16, 2020 at 19:12

7 Sometimes I scroll down on SO questions. When I find gems like this, I know it was worth it. – MLavrentyev

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

1 @Hills: It's only possible to use , and _ with this method (updated above). Here's the details: realpython.com/python-formatted-output/#the-group-subcomponent – Emil Stenström May 3, 2021 at 8:58

1 @RastkoGojgic: It's part of the language. See the link I posted above. – Emil Stenström Jul 27, 2021 at 8:01

311

I got this to work:

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'en_US')
'en_US'
>>> locale.format("%d", 1255000, grouping=True)
'1,255,000'
```

Sure, you don't *need* internationalization support, but it's clear, concise, and uses a built-in library.

P.S. That "%d" is the usual %-style formatter. You can have only one formatter, but it can be whatever you need in terms of field width and precision settings.

P.P.S. If you can't get `locale` to work, I'd suggest a modified version of Mark's answer:

```
def intWithCommas(x):
    if type(x) not in [type(0), type(0L)]:
        raise TypeError("Parameter must be an integer.")
    if x < 0:
        return '-' + intWithCommas(-x)
    result = ''
    while x >= 1000:
        x, r = divmod(x, 1000)
        result = ",%03d%s" % (r, result)
    return "%d%s" % (x, result)
```

Recursion is useful for the negative case, but one recursion per comma seems a bit excessive to me.

Share Improve this answer Follow edited Sep 1, 2015 at 19:39 answered Nov 30, 2009 at 23:22

Mike DeSimone
41.2k ● 10 ● 72 ● 96

15 I tried your code, and unfortunately, I get this: "locale.Error: unsupported locale setting". :-s – Mark Byers Nov 30, 2009 at 23:25

12 Mark: If you're on Linux, you might want to look at what is in your /etc/locale.gen, or whatever your glibc is using to build its locales. You might also want to try ""en", "en_US.utf8", "en_US.UTF-8", 'en_UK" (sp?), etc. mikez: There needs to be a book: "Dr. PEP: Or How I Learned to Stop Worrying and Love docs.python.org." I gave up memorizing all the libraries back around Python 1.5.6. As for `locale`, I use as little of it as I can.

10 You can use '' for `setlocale` to use the default, which hopefully will be appropriate. – Mark Ransom Nov 30, 2009 at 23:49

25 Try this: locale.setlocale(locale.LC_ALL, '') It worked for me – Nadia Alramli Dec 1, 2009 at 0:00

1 Although clever, I don't like functions that make global settings... Using 'blah'.format() is the better way to go. – Cerin May 20, 2012 at 18:14

169

For inefficiency and unreadability it's hard to beat:

```
>>> import itertools
>>> s = '-1234567'
>>> ','.join(["%s%s%s" % (x[0], x[1] or '', x[2] or '') for x in
itertools.izip_longest(s[::-1][::3], s[::-1][1::3], s[::-1][2::3])])
[::-1].replace('-,','-')
```

Share Improve this answer Follow

answered Jan 19, 2012 at 19:35

Kasey Kirkham
1,755 ●1 ●10 ●2

3 would be nice if this at least would work. try this number "17371830" it becomes "173.718.3.0" =) – holms Feb 17, 2012 at 18:15

7 Periods? That ain't even possible, holms. This piece of junk totally ignores locale. I wonder how you got that result. Your example produces '17,371,830' for me, as expected. – Kasey Kirkham Feb 25, 2012 at 1:42

16 To make this a function I would suggest: `lambda x: (lambda s: ','.join(["%s%s%s" % (x[0], x[1] or '', x[2] or '') for x in itertools.izip_longest(s[::-1][::3], s[::-1][1::3], s[::-1][2::3])])[::-1].replace('-,','-'))(str(x))` just to keep the obfuscation theme. – quantum Nov 23, 2012 at 3:24

Why does this have so many upvotes? – Dudelstein Apr 12 at 10:48

121

Here is the locale grouping code after removing irrelevant parts and cleaning it up a little:

(The following only works for integers)

```
def group(number):
    s = '%d' % number
    groups = []
    while s and s[-1].isdigit():
        groups.append(s[-3:])
        s = s[:-3]
    return s + ','.join(reversed(groups))

>>> group(-23432432434.34)
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

There are already some good answers in here. I just want to add this for future reference. In python 2.7 there is going to be a format specifier for thousands separator. According to python docs it works like this

```
>>> '{:20,.2f}'.format(f)
'18,446,744,073,709,551,616.00'
```

In python3.1 you can do the same thing like this:

```
>>> format(1234567, ',d')
'1,234,567'
```

Share Improve this answer Follow

edited Mar 11, 2013 at 18:13
Martijn Pieters ♦
1.0m ● 288 ● 4008 ● 3310

answered Nov 30, 2009 at 23:42
Nadia Alramli
111k ● 36 ● 172 ● 152

Yeah, the harder ways are mainly for folks on older Pythons, such as those shipped with RHEL and other long-term support distros. – Mike DeSimone May 20, 2012 at 22:21

3 how to express this with format strings ? "%,d" % 1234567 does not work – Frederic Bazin May 6, 2013 at 8:31

For me this answer provided `{:15,d}'.format(len(SortedList))` which worked perfectly. – WinEunuuchs2Unix Jul 25, 2020 at 1:15

Surprisingly, `format(1234567, ',d')` also works in python 2.7 – synkro Mar 1, 2021 at 3:41

3 surely `format(1234567, ',')` is the simplest answer. – D.L May 6, 2021 at 12:33

45

Here's a one-line regex replacement:

```
re.sub("(\d)(?=(\d{3})+(?!\d))", r"\1,", "%d" % val)
```

Works only for inegral outputs:

```
import re
val = 1234567890
re.sub("(\d)(?=(\d{3})+(?!\d))", r"\1,", "%d" % val)
# Returns: '1,234,567,890'

val = 1234567890.1234567890
# Returns: '1,234,567,890'
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

```
re.sub("(\d)(?=(\d{3})+(?!\d))", r"\1,", "%.3f" % val)
# Returns: '1,234,567,890.123'
```

NB: Doesn't work correctly with more than three decimal digits as it will attempt to group the decimal part:

```
re.sub("(\d)(?=(\d{3})+(?!\d))", r"\1,", "%.5f" % val)
# Returns: '1,234,567,890.12,346'
```

How it works

Let's break it down:

```
re.sub(pattern, repl, string)

pattern = \
    "(\d)           # Find one digit...
     (?=            # that is followed by...
         (\d{3})+   # one or more groups of three digits...
         (?!\d)     # which are not followed by any more digits.
     )",

repl = \
    r"\1,",         # Replace that one digit by itself, followed by a comma,
                    # and continue looking for more matches later in the string.
                    # (re.sub() replaces all matches it finds in the input)

string = \
    "%d" % val      # Format the string as a decimal to begin with
```

Share Improve this answer Follow   edited Aug 16, 2013 at 15:13   answered Jul 19, 2012 at 13:42
Daniel Fortunov
42.9k ● 26 ● 80 ● 106

1 use verbose mode and you can have comments right inside the code – Daniel Jan 12, 2018 at 9:19

Couldn't you replace the "(?!\d)" with a "$ " ? – GL2014 Jan 31, 2020 at 21:25

41

This is what I do for floats. Although, honestly, I'm not sure which versions it works for - I'm using 2.7:

```
my_number = 4385893.382939491

my_string = '{:0,.2f}'.format(my_number)
```

Update: I recently had an issue with this format (couldn't tell you the exact reason), but was able to fix it by dropping the `0`:

```
my_string = '{:,.2f}'.format(my_number)
```

Share Improve this answer Follow edited May 9, 2017 at 14:03 answered Jul 24, 2016 at 14:26

elPastor
8,255 ● 11 ● 54 ● 81

22

You can also use `'{:n}'.format( value )` for a locale representation. I think this is the simpliest way for a locale solution.

For more information, search for `thousands` in Python DOC.

For currency, you can use `locale.currency`, setting the flag `grouping`:

Code

```
import locale

locale.setlocale( locale.LC_ALL, '' )
locale.currency( 1234567.89, grouping = True )
```

Output

```
'Portuguese_Brazil.1252'
'R$ 1.234.567,89'
```

Share Improve this answer Follow answered Aug 17, 2015 at 12:43

Diogo
1,056 ● 8 ● 15

21

Slightly expanding the answer of Ian Schneider:

If you want to use a custom thousands separator, the simplest solution is:

```
'{:,}'.format(value).replace(',', your_custom_thousands_separator)
```

Examples

If you want the German representation like this, it gets a bit more complicated:

```
('{:,.2f}'.format(123456789.012345)
          .replace(',', ' ')  # 'save' the thousands separators
          .replace('.', ',')  # dot to comma
          .replace(' ', '.')) # thousand separators to dot
```

Share Improve this answer Follow

answered Sep 28, 2017 at 9:39
Martin Thoma
121k ● 154 ● 603 ● 927

Slightly shorter: `'{:_.2f}'.format(12345.6789).replace('.', ',').replace('_', '.')`
– Tom Pohl May 16, 2019 at 17:40

15

Here are some ways to do it with formatting (compatible with floats and ints)

```
num = 2437.68

# Way 1: String Formatting

'{:,}'.format(num)
>>> '2,437.68'


# Way 2: F-Strings

f'{num:,}'
>>> '2,437.68'


# Way 3: Built-in Format Function

format(num, ',')
>>> '2,437.68'
```

Share Improve this answer Follow

edited May 5, 2021 at 9:58
Thavas Antonio
5,779 ● 1 ● 14 ● 42

answered May 4, 2021 at 20:39
RookieCoder
171 ● 1 ● 6

12

I'm sure there must be a standard library function for this, but it was fun to try to write it myself using recursion so here's what I came up with:

```
def intToStringWithCommas(x):
    if type(x) is not int and type(x) is not long:
        raise TypeError("Not an integer!")
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

```
    else:
        return intToStringWithCommas(x / 1000) + ',' + '%03d' % (x % 1000)
```

Having said that, if someone else does find a standard way to do it, you should use that instead.

Share Improve this answer Follow edited Dec 1, 2009 at 0:17 answered Nov 30, 2009 at 23:19
Mark Byers
802k ● 189 ● 1573 ● 1449

Unfortunately doesn't work in all cases. intToStringWithCommas(1000.1) -> '1.0001,000' – Nadia Alramli Nov 30, 2009 at 23:53

He specifically said integers and that it should be as simple as possible, so I decided not to handle datatypes other than integers. I also made it explicit in the function name _int_ToStringWithCommas. Now I've also added a raise to make it more clear. – Mark Byers Dec 1, 2009 at 0:20

9

The accepted answer is fine, but I actually prefer `format(number,',')`. Easier for me to interpret and remember.

https://docs.python.org/3/library/functions.html#format

Share Improve this answer Follow answered Nov 8, 2017 at 22:13
Magenta Nova
691 ● 1 ● 9 ● 15

Works perfectly, also avoiding to display too many decimal digits for floats. – Rexcirus Aug 14, 2019 at 22:16

8

From the comments to activestate recipe 498181 I reworked this:

```
import re
def thous(x, sep=',', dot='.'):
    num, _, frac = str(x).partition(dot)
    num = re.sub(r'(\d{3})(?=\d)', r'\1'+sep, num[::-1])[::-1]
    if frac:
        num += dot + frac
    return num
```

It uses the regular expressions feature: lookahead i.e. `(?=\d)` to make sure only groups of three digits that have a digit 'after' them get a comma. I say 'after' because the string is reverse at this point.

`[::-1]` just reverses a string.

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

Tom Viner
6,595 ● 7 ● 38 ● 40

8

Simplest answer:

```
format (123456, ",")
```

Result:

```
'123,456'
```

Share Improve this answer Follow

answered Nov 9, 2022 at 7:34
leenremm
913 ● 12 ● 17

1 Simplest indeed! – Merin Nakarmi Jan 7 at 23:36

1 I would argue that this is not only simplest, but *more correct* than using `str.format` or f-strings. – shadowtalker Feb 2 at 14:50

7

Python 3

--

Integers (without decimal):

`"{:,d}".format(1234567)`

--

Floats (with decimal):

`"{:,.2f}".format(1234567)`

where the number before `f` specifies the number of decimal places.

--

Bonus

Quick-and-dirty starter function for the Indian lakhs/crores numbering system (12,34,567):

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

Adam Starrh
6,248 ● 8 ● 50 ● 88

6

from Python version 2.6 you can do this:

```
def format_builtin(n):
    return format(n, ',')
```

For Python versions < 2.6 and just for your information, here are 2 manual solutions, they turn floats to ints but negative numbers work correctly:

```
def format_number_using_lists(number):
    string = '%d' % number
    result_list = list(string)
    indexes = range(len(string))
    for index in indexes[::-3][1:]:
        if result_list[index] != '-':
            result_list.insert(index+1, ',')
    return ''.join(result_list)
```

few things to notice here:

- this line: **string = '%d' % number** beautifully converts a number to a string, it supports negatives and it drops fractions from floats, making them ints;
- this slice **indexes[::-3]** returns each third item starting from the end, so I used another slice **[1:]** to remove the very last item cuz I don't need a comma after the last number;
- this conditional **if l[index] != '-'** is being used to support negative numbers, do not insert a comma after the minus sign.

And a more hardcore version:

```
def format_number_using_generators_and_list_comprehensions(number):
    string = '%d' % number
    generator = reversed(
        [
            value+',' if (index!=0 and value!='-' and index%3==0) else value
            for index,value in enumerate(reversed(string))
        ]
    )
    return ''.join(generator)
```

Share Improve this answer Follow

answered Jul 2, 2016 at 4:54
arka
69 ● 1 ● 5

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

3

I am a Python beginner, but an experienced programmer. I have Python 3.5, so I can just use the comma, but this is nonetheless an interesting programming exercise. Consider the case of an unsigned integer. The most readable Python program for adding thousands separators appears to be:

```
def add_commas(instr):
    out = [instr[0]]
    for i in range(1, len(instr)):
        if (len(instr) - i) % 3 == 0:
            out.append(',')
        out.append(instr[i])
    return ''.join(out)
```

It is also possible to use a list comprehension:

```
add_commas(instr):
    rng = reversed(range(1, len(instr) + (len(instr) - 1)//3 + 1))
    out = [',' if j%4 == 0 else instr[-(j - j//4)] for j in rng]
    return ''.join(out)
```

This is shorter, and could be a one liner, but you will have to do some mental gymnastics to understand why it works. In both cases we get:

```
for i in range(1, 11):
    instr = '1234567890'[:i]
    print(instr, add_commas(instr))
```

```
1 1
12 12
123 123
1234 1,234
12345 12,345
123456 123,456
1234567 1,234,567
12345678 12,345,678
123456789 123,456,789
1234567890 1,234,567,890
```

The first version is the more sensible choice, if you want the program to be understood.

Share Improve this answer Follow

edited Mar 28, 2016 at 12:56
Tunaki
131k ● 46 ● 330 ● 415

answered Mar 28, 2016 at 12:51
Geoff Fergusson
39 ● 1

2

I have found some issues with the dot separator in the previous top voted answers. I have designed a **universal solution where you can use whatever you want as a thousand separator without modifying the locale**. I know it's not the most elegant solution, but it gets the job done. Feel free to improve it !

```
def format_integer(number, thousand_separator='.'):
    def reverse(string):
        string = "".join(reversed(string))
        return string

    s = reverse(str(number))
    count = 0
    result = ''
    for char in s:
        count = count + 1
        if count % 3 == 0:
            if len(s) == count:
                result = char + result
            else:
                result = thousand_separator + char + result
        else:
            result = char + result
    return result


print(format_integer(50))
# 50
print(format_integer(500))
# 500
print(format_integer(50000))
# 50.000
print(format_integer(50000000))
# 50.000.000
```

Share Improve this answer Follow

answered Nov 7, 2019 at 10:42

Manrique
1,983 ●3 ●15 ●35

Use separators and decimals together in float numbers : (In this example, two decimal places)

2

```
large_number = 4545454.26262666
print(f"Formatted: {large_number:,.2f}")
```

Result: Formatted: 4,545,454.26

Share Improve this answer Follow

answered May 21, 2021 at 17:28

Seyed Mahdi Kermani
33 ●5

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

1

```
def float2comma(f):
    s = str(abs(f)) # Convert to a string
    decimalposition = s.find(".") # Look for decimal point
    if decimalposition == -1:
        decimalposition = len(s) # If no decimal, then just work from the end
    out = ""
    for i in range(decimalposition+1, len(s)): # do the decimal
        if not (i-decimalposition-1) % 3 and i-decimalposition-1: out = out+","
        out = out+s[i]
    if len(out):
        out = "."+out # add the decimal point if necessary
    for i in range(decimalposition-1,-1,-1): # working backwards from decimal
point
        if not (decimalposition-i-1) % 3 and decimalposition-i-1: out = ","+out
        out = s[i]+out
    if f < 0:
        out = "-"+out
    return out
```

Usage Example:

```
>>> float2comma(10000.1111)
'10,000.111,1'
>>> float2comma(656565.122)
'656,565.122'
>>> float2comma(-656565.122)
'-656,565.122'
```

Share Improve this answer Follow edited Nov 6, 2010 at 14:06 answered Nov 6, 2010 at 13:33

Edward van Kuik
1,337 ● 1 ● 9 ● 9

1 `float2comma(12031023.1323)` returns: '12,031,023.132,3' – demux Jul 5, 2011 at 21:53

1

One liner for Python 2.5+ and Python 3 (positive int only):

```
''.join(reversed([x + (',' if i and not i % 3 else '') for i, x in
enumerate(reversed(str(1234567)))]))
```

Share Improve this answer Follow edited Jul 30, 2015 at 15:26 answered Jun 4, 2013 at 17:48

Collin Anderson
14.5k ● 6 ● 64 ● 56

I'm using python 2.5 so I don't have access to the built-in formatting.

Join Stack Overflow to find the best answer to your technical question, help others answer theirs. Sign up either. This is not necessary an 'issue' as django isn't really THAT focused on this kind of low-level performance. Also, I was expecting a factor of 10 difference in performance, but it's only 3 times slower.

Out of curiosity I implemented a few versions of intcomma to see what the performance advantages are when using regex. My test data concludes a slight advantage for this task, but surprisingly not much at all.

I also was pleased to see what I suspected: using the reverse xrange approach is unnecessary in the no-regex case, but it does make the code look slightly better at the cost of ~10% performance.

Also, I assume what you're passing in is a string and looks somewhat like a number. Results undetermined otherwise.

```
from __future__ import with_statement
from contextlib import contextmanager
import re,time

re_first_num = re.compile(r"\d")
def intcomma_noregex(value):
    end_offset, start_digit, period =
len(value),re_first_num.search(value).start(),value.rfind('.')
    if period == -1:
        period=end_offset
    segments,_from_index,leftover = [],0,(period-start_digit) % 3
    for _index in xrange(start_digit+3 if not leftover else
start_digit+leftover,period,3):
        segments.append(value[_from_index:_index])
        _from_index=_index
    if not segments:
        return value
    segments.append(value[_from_index:])
    return ','.join(segments)

def intcomma_noregex_reversed(value):
    end_offset, start_digit, period =
len(value),re_first_num.search(value).start(),value.rfind('.')
    if period == -1:
        period=end_offset
    _from_index,segments = end_offset,[]
    for _index in xrange(period-3,start_digit,-3):
        segments.append(value[_index:_from_index])
        _from_index=_index
    if not segments:
        return value
    segments.append(value[:_from_index])
    return ','.join(reversed(segments))

re_3digits = re.compile(r'(?<=\d)\d{3}(?!\d)')
def intcomma(value):
    segments,last_endoffset=[],len(value)
    while last_endoffset > 3:
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs. Sign up

```
            segments.append(value[digit_group.start():last_endoffset])
            last_endoffset=digit_group.start()
    if not segments:
        return value
    if last_endoffset:
        segments.append(value[:last_endoffset])
    return ','.join(reversed(segments))

def intcomma_recurs(value):
    """
    Converts an integer to a string containing commas every three digits.
    For example, 3000 becomes '3,000' and 45000 becomes '45,000'.
    """
    new = re.sub("^(-?\d+)(\d{3})", '\g<1>,\g<2>', str(value))
    if value == new:
        return new
    else:
        return intcomma(new)

@contextmanager
def timed(save_time_func):
    begin=time.time()
    try:
        yield
    finally:
        save_time_func(time.time()-begin)

def testset_xsimple(func):
    func('5')

def testset_simple(func):
    func('567')

def testset_onecomma(func):
    func('567890')

def testset_complex(func):
    func('-1234567.024')

def testset_average(func):
    func('-1234567.024')
    func('567')
    func('5674')

if __name__ == '__main__':
    print 'Test results:'
    for test_data in ('5','567','1234','1234.56','-253892.045'):
        for func in
(intcomma,intcomma_noregex,intcomma_noregex_reversed,intcomma_recurs):
            print func.__name__,test_data,func(test_data)
    times=[]
    def overhead(x):
        pass
    for test_run in xrange(1,4):
        for func in
(intcomma,intcomma_noregex,intcomma_noregex_reversed,intcomma_recurs,overhead):
            for testset in
(testset_xsimple,testset_simple,testset_onecomma,testset_complex,testset_average):
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

```
                        testset(func)
    for (test_run,func,testset),_delta in times:
        print test_run,func.__name__,testset.__name__,_delta
```

And here are the test results:

```
intcomma 5 5
intcomma_noregex 5 5
intcomma_noregex_reversed 5 5
intcomma_recurs 5 5
intcomma 567 567
intcomma_noregex 567 567
intcomma_noregex_reversed 567 567
intcomma_recurs 567 567
intcomma 1234 1,234
intcomma_noregex 1234 1,234
intcomma_noregex_reversed 1234 1,234
intcomma_recurs 1234 1,234
intcomma 1234.56 1,234.56
intcomma_noregex 1234.56 1,234.56
intcomma_noregex_reversed 1234.56 1,234.56
intcomma_recurs 1234.56 1,234.56
intcomma -253892.045 -253,892.045
intcomma_noregex -253892.045 -253,892.045
intcomma_noregex_reversed -253892.045 -253,892.045
intcomma_recurs -253892.045 -253,892.045
1 intcomma testset_xsimple 0.0410001277924
1 intcomma testset_simple 0.0369999408722
1 intcomma testset_onecomma 0.213000059128
1 intcomma testset_complex 0.296000003815
1 intcomma testset_average 0.503000020981
1 intcomma_noregex testset_xsimple 0.134000062943
1 intcomma_noregex testset_simple 0.134999990463
1 intcomma_noregex testset_onecomma 0.190999984741
1 intcomma_noregex testset_complex 0.209000110626
1 intcomma_noregex testset_average 0.513000011444
1 intcomma_noregex_reversed testset_xsimple 0.124000072479
1 intcomma_noregex_reversed testset_simple 0.12700009346
1 intcomma_noregex_reversed testset_onecomma 0.230000019073
1 intcomma_noregex_reversed testset_complex 0.236999988556
1 intcomma_noregex_reversed testset_average 0.56299996376
1 intcomma_recurs testset_xsimple 0.348000049591
1 intcomma_recurs testset_simple 0.34600019455
1 intcomma_recurs testset_onecomma 0.625
1 intcomma_recurs testset_complex 0.773999929428
1 intcomma_recurs testset_average 1.6890001297
1 overhead testset_xsimple 0.0179998874664
1 overhead testset_simple 0.0190000534058
1 overhead testset_onecomma 0.0190000534058
1 overhead testset_complex 0.0190000534058
1 overhead testset_average 0.0309998989105
2 intcomma testset_xsimple 0.0360000133514
2 intcomma testset_simple 0.0369999408722
2 intcomma testset_onecomma 0.207999944687
2 intcomma testset_complex 0.302000045776
2 intcomma testset_average 0.523000001907
```

```
2 intcomma_noregex testset_complex 0.200999975204
2 intcomma_noregex testset_average 0.523000001907
2 intcomma_noregex_reversed testset_xsimple 0.130000114441
2 intcomma_noregex_reversed testset_simple 0.129999876022
2 intcomma_noregex_reversed testset_onecomma 0.236000061035
2 intcomma_noregex_reversed testset_complex 0.241999864578
2 intcomma_noregex_reversed testset_average 0.582999944687
2 intcomma_recurs testset_xsimple 0.351000070572
2 intcomma_recurs testset_simple 0.352999925613
2 intcomma_recurs testset_onecomma 0.648999929428
2 intcomma_recurs testset_complex 0.808000087738
2 intcomma_recurs testset_average 1.81900000572
2 overhead testset_xsimple 0.0189998149872
2 overhead testset_simple 0.0189998149872
2 overhead testset_onecomma 0.0190000534058
2 overhead testset_complex 0.0179998874664
2 overhead testset_average 0.0299999713898
3 intcomma testset_xsimple 0.0360000133514
3 intcomma testset_simple 0.0360000133514
3 intcomma testset_onecomma 0.210000038147
3 intcomma testset_complex 0.305999994278
3 intcomma testset_average 0.493000030518
3 intcomma_noregex testset_xsimple 0.131999969482
3 intcomma_noregex testset_simple 0.136000156403
3 intcomma_noregex testset_onecomma 0.192999839783
3 intcomma_noregex testset_complex 0.202000141144
3 intcomma_noregex testset_average 0.509999990463
3 intcomma_noregex_reversed testset_xsimple 0.125999927521
3 intcomma_noregex_reversed testset_simple 0.126999855042
3 intcomma_noregex_reversed testset_onecomma 0.235999822617
3 intcomma_noregex_reversed testset_complex 0.243000030518
3 intcomma_noregex_reversed testset_average 0.56200003624
3 intcomma_recurs testset_xsimple 0.337000131607
3 intcomma_recurs testset_simple 0.342000007629
3 intcomma_recurs testset_onecomma 0.609999895096
3 intcomma_recurs testset_complex 0.75
3 intcomma_recurs testset_average 1.68300008774
3 overhead testset_xsimple 0.0189998149872
3 overhead testset_simple 0.018000125885
3 overhead testset_onecomma 0.018000125885
3 overhead testset_complex 0.0179998874664
3 overhead testset_average 0.0299999713898
```

Share Improve this answer Follow

edited Aug 5, 2017 at 13:47
idjaw
25.1k ● 7 ● 67 ● 82

answered Jun 22, 2012 at 2:04
parity3
633 ● 9 ● 18

I thought Daniel Fortunov's one-regex solution would be #1 and beat all the algorithms because regex is so refined/optimized and coded in C, but nope.. I guess the pattern and lookahead's are too expensive. it falls in at about double the time of the intcomma above, even with precompiling the regex. – parity3 Jul 22, 2012 at 19:32

this is baked into python per PEP -> https://www.python.org/dev/peps/pep-0378/

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

there are more formats described in the PEP, have at it

Share Improve this answer Follow

answered Jan 24, 2019 at 15:42

Raveen Beemsingh
560 ●5 ●7

1

babel module in python has feature to apply commas depending on the locale provided.

To install babel run the below command.

```
pip install babel
```

usage

```
format_currency(1234567.89, 'USD', locale='en_US')
# Output: $1,234,567.89
format_currency(1234567.89, 'USD', locale='es_CO')
# Output: US$ 1.234.567,89 (raw output US$\xa01.234.567,89)
format_currency(1234567.89, 'INR', locale='en_IN')
# Output: ₹12,34,567.89
```

Share Improve this answer Follow

answered Dec 27, 2020 at 11:59

navule
3,030 ●2 ●34 ●51

0

This does money along with the commas

```
def format_money(money, presym='$', postsym=''):
    fmt = '%0.2f' % money
    dot = string.find(fmt, '.')
    ret = []
    if money < 0 :
        ret.append('(')
        p0 = 1
    else :
        p0 = 0
    ret.append(presym)
    p1 = (dot-p0) % 3 + p0
    while True :
        ret.append(fmt[p0:p1])
        if p1 == dot : break
        ret.append(',')
        p0 = p1
        p1 += 3
    ret.append(fmt[dot:])   # decimals
    ret.append(postsym)
```

Share Improve this answer Follow

answered Dec 8, 2013 at 19:51

Bob F
69 ● 1

I know this is an old answer but anyone is in for a world of hurt if they use floats for monetary calculations. Floats are not precise! Lookup IEEE 754, and get off the road to hell. – ingyhere Dec 2, 2021 at 6:40

@ingyhere ironically an IEEE 754 double can hold any integer up to 2**53 without any loss in precision. Javascript doesn't even bother having an integer type. – Mark Ransom May 6, 2022 at 3:00

0

I have a python 2 and python 3 version of this code. I know that the question was asked for python 2 but now (8 years later lol) people will probably be using python 3.

Python 3 Code:

```
import random
number = str(random.randint(1, 10000000))
comma_placement = 4
print('The original number is: {}. '.format(number))
while True:
    if len(number) % 3 == 0:
        for i in range(0, len(number) // 3 - 1):
            number = number[0:len(number) - comma_placement + 1] + ',' +
number[len(number) - comma_placement + 1:]
            comma_placement = comma_placement + 4
    else:
        for i in range(0, len(number) // 3):
            number = number[0:len(number) - comma_placement + 1] + ',' +
number[len(number) - comma_placement + 1:]
    break
print('The new and improved number is: {}'.format(number))
```

Python 2 Code: (Edit. The python 2 code isn't working. I am thinking that the syntax is different).

```
import random
number = str(random.randint(1, 10000000))
comma_placement = 4
print 'The original number is: %s.' % (number)
while True:
    if len(number) % 3 == 0:
        for i in range(0, len(number) // 3 - 1):
            number = number[0:len(number) - comma_placement + 1] + ',' +
number[len(number) - comma_placement + 1:]
            comma_placement = comma_placement + 4
    else:
        for i in range(0, len(number) // 3):
            number = number[0:len(number) - comma_placement + 1] + ',' +
number[len(number) - comma_placement + 1:]
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

Share Improve this answer Follow

edited Jul 28, 2017 at 8:49

answered Jul 28, 2017 at 8:34

Dinosaur212
123 ● 2 ● 9

-1

Here is another variant using a generator function that works for integers:

```
def ncomma(num):
    def _helper(num):
        # assert isinstance(numstr, basestring)
        numstr = '%d' % num
        for ii, digit in enumerate(reversed(numstr)):
            if ii and ii % 3 == 0 and digit.isdigit():
                yield ','
            yield digit

    return ''.join(reversed([n for n in _helper(num)]))
```

And here's a test:

```
>>> for i in (0, 99, 999, 9999, 999999, 1000000, -1, -111, -1111, -111111,
-1000000):
...     print i, ncomma(i)
...
0 0
99 99
999 999
9999 9,999
999999 999,999
1000000 1,000,000
-1 -1
-111 -111
-1111 -1,111
-111111 -111,111
-1000000 -1,000,000
```

Share Improve this answer Follow

answered Oct 4, 2012 at 17:53

user650654
5,510 ● 3 ● 40 ● 43

-1

Italy:

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL,"")
'Italian_Italy.1252'
>>> f"{1000:n}"
'1.000'
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

PythonProgrammi
22k ●3 ●40 ●34

-2

Just subclass `long` (or `float`, or whatever). This is highly practical, because this way you can still use your numbers in math ops (and therefore existing code), but they will all print nicely in your terminal.

```
>>> class number(long):

        def __init__(self, value):
            self = value

        def __repr__(self):
            s = str(self)
            l = [x for x in s if x in '1234567890']
            for x in reversed(range(len(s)-1)[::3]):
                l.insert(-x, ',')
            l = ''.join(l[1:])
            return ('-'+l if self < 0 else l)

>>> number(-100000)
-100,000
>>> number(-100)
-100
>>> number(-12345)
-12,345
>>> number(928374)
928,374
>>> 345
```

Share Improve this answer Follow

edited Aug 5, 2017 at 13:48
idjaw
25.1k ●7 ●67 ●82

answered Nov 30, 2009 at 23:25
tweneale
2,796 ●4 ●28 ●32

8 I like the subclass idea, but is `__repr__()` the correct method to override? I would suggest overriding `__str__()` and leaving `__repr__()` alone, because `int(repr(number(928374)))` ought to work, but `int()` will choke on the commas. – steveha Dec 1, 2009 at 1:02

@steveha has a good point, but the justification should have been that `number(repr(number(928374)))` doesn't work, not `int(repr(number(928374)))`. All the same, to make this approach work directly with `print`, as the OP requested, the `__str__()` method should be the one overridden rather than `__repr__()`. Regardless, there appears to be a bug in the core comma insertion logic. – martineau Oct 18, 2010 at 12:29

-8

For floats:

```
float(filter(lambda x: x!=',', '1,234.52'))
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

For ints:

```
int(filter(lambda x: x!=',', '1,234'))
# returns 1234
```

Share Improve this answer Follow

edited Dec 29, 2011 at 23:16
Dennis Williamson
341k ● 89 ● 373 ● 437

answered Dec 20, 2011 at 5:03
Jennifer
1

5 That *removes* the commas. While handy, the OP asked for a way to *add* them. Besides, something like `float('1,234.52'.translate(None, ','))` might be more straightforward and possibly faster.
– Dennis Williamson Dec 29, 2011 at 23:14