



Inhaltsverzeichnis

Modul:ConvertNumeric

Usage

```
{{#invoke:ConvertNumeric|function_name}}
```

See also

- [Vorlage:TI](#)
- [Vorlage:TI](#)
- [Module:StripToNumbers](#) - extract a number from a string (supports negatives and decimals) and return it, or optionally return a halved value

```
-- Module for converting between different representations of numbers. See talk p
-- For unit tests see: [[Module:ConvertNumeric/testcases]]
-- When editing, preview with: [[Module_talk:ConvertNumeric/testcases]]
-- First, edit [[Module:ConvertNumeric/sandbox]], then preview with [[Module_talk
```

```
local ones_position = {
    [0] = 'zero',
    [1] = 'one',
    [2] = 'two',
    [3] = 'three',
    [4] = 'four',
    [5] = 'five',
    [6] = 'six',
    [7] = 'seven',
    [8] = 'eight',
    [9] = 'nine',
    [10] = 'ten',
    [11] = 'eleven',
    [12] = 'twelve',
    [13] = 'thirteen',
    [14] = 'fourteen',
    [15] = 'fifteen',
    [16] = 'sixteen',
    [17] = 'seventeen',
    [18] = 'eighteen',
    [19] = 'nineteen'
}
```

```
local ones_position_ord = {
    [0] = 'zeroth',
    [1] = 'first',
    [2] = 'second',
    [3] = 'third',
    [4] = 'fourth',
    [5] = 'fifth',
    [6] = 'sixth',
    [7] = 'seventh',
    [8] = 'eighth',
    [9] = 'ninth',
    [10] = 'tenth',
```

```
[11] = 'eleventh',
[12] = 'twelfth',
[13] = 'thirteenth',
[14] = 'fourteenth',
[15] = 'fifteenth',
[16] = 'sixteenth',
[17] = 'seventeenth',
[18] = 'eighteenth',
[19] = 'nineteenth'
}

local ones_position_plural = {
    [0] = 'zeros',
    [1] = 'ones',
    [2] = 'twos',
    [3] = 'threes',
    [4] = 'fours',
    [5] = 'fives',
    [6] = 'sixes',
    [7] = 'sevens',
    [8] = 'eights',
    [9] = 'nines',
    [10] = 'tens',
    [11] = 'elevens',
    [12] = 'twelves',
    [13] = 'thirteens',
    [14] = 'fourteens',
    [15] = 'fifteens',
    [16] = 'sixteens',
    [17] = 'seventeens',
    [18] = 'eighteens',
    [19] = 'nineteens'
}

local tens_position = {
    [2] = 'twenty',
    [3] = 'thirty',
    [4] = 'forty',
    [5] = 'fifty',
    [6] = 'sixty',
    [7] = 'seventy',
    [8] = 'eighty',
    [9] = 'ninety'
}

local tens_position_ord = {
    [2] = 'twentieth',
    [3] = 'thirtieth',
    [4] = 'fortieth',
    [5] = 'fiftieth',
    [6] = 'sixtieth',
    [7] = 'seventieth',
    [8] = 'eightieth',
    [9] = 'ninetieth'
}

local tens_position_plural = {
    [2] = 'twenties',
    [3] = 'thirties',
    [4] = 'forties',
    [5] = 'fifties',
    [6] = 'sixties',
    [7] = 'seventies',
    [8] = 'eighties',
```



```
    [9] = 'nineties'
}
local groups = {
    [1] = 'thousand',
    [2] = 'million',
    [3] = 'billion',
    [4] = 'trillion',
    [5] = 'quadrillion',
    [6] = 'quintillion',
    [7] = 'sextillion',
    [8] = 'septillion',
    [9] = 'octillion',
    [10] = 'nonillion',
    [11] = 'decillion',
    [12] = 'undecillion',
    [13] = 'duodecillion',
    [14] = 'tredecillion',
    [15] = 'quattuordecillion',
    [16] = 'quindecillion',
    [17] = 'sexdecillion',
    [18] = 'septendecillion',
    [19] = 'octodecillion',
    [20] = 'novemdecillion',
    [21] = 'vigintillion',
    [22] = 'unvigintillion',
    [23] = 'duovigintillion',
    [24] = 'tresvigintillion',
    [25] = 'quattuorvigintillion',
    [26] = 'quingvigintillion',
    [27] = 'sesvigintillion',
    [28] = 'septemvigintillion',
    [29] = 'octovigintillion',
    [30] = 'novemvigintillion',
    [31] = 'trigintillion',
    [32] = 'untrigintillion',
    [33] = 'duotrigintillion',
    [34] = 'trestrigintillion',
    [35] = 'quattuortrigintillion',
    [36] = 'quinguatrigintillion',
    [37] = 'sestrigintillion',
    [38] = 'septentrigintillion',
    [39] = 'octotrigintillion',
    [40] = 'noventrigintillion',
    [41] = 'quadragintillion',
    [51] = 'quingagintillion',
    [61] = 'sexagintillion',
    [71] = 'septuagintillion',
    [81] = 'octogintillion',
    [91] = 'nonagintillion',
    [101] = 'centillion',
    [102] = 'uncentillion',
    [103] = 'duocentillion',
    [104] = 'trescentillion',
    [111] = 'decicentillion',
    [112] = 'undecicentillion',
    [121] = 'viginticentillion',
    [122] = 'unviginticentillion',
    [131] = 'trigintacentillion',
    [141] = 'quadragintacentillion',
    [151] = 'quingagintacentillion',
    [161] = 'sexagintacentillion',
    [171] = 'septuagintacentillion',
    [181] = 'octogintacentillion',
```



```
[191] = 'nonagintacenticillion',
[201] = 'ducentillion',
[301] = 'trecentillion',
[401] = 'quadrilingentillion',
[501] = 'quingentillion',
[601] = 'sescentillion',
[701] = 'septingentillion',
[801] = 'octingentillion',
[901] = 'nongentillion',
[1001] = 'millinillion',
}

local roman_numerals = {
    I = 1,
    V = 5,
    X = 10,
    L = 50,
    C = 100,
    D = 500,
    M = 1000
}

local engord_tens_end = {
    ['twentieth'] = 20,
    ['thirtieth'] = 30,
    ['fortieth'] = 40,
    ['fiftieth'] = 50,
    ['sixtieth'] = 60,
    ['seventieth'] = 70,
    ['eightieth'] = 80,
    ['ninetieth'] = 90,
}

local eng_tens_cont = {
    ['twenty'] = 20,
    ['thirty'] = 30,
    ['forty'] = 40,
    ['fifty'] = 50,
    ['sixty'] = 60,
    ['seventy'] = 70,
    ['eighty'] = 80,
    ['ninety'] = 90,
}

-- Converts a given valid roman numeral (and some invalid roman numerals) to a number
local function roman_to_numeral(roman)
    if type(roman) ~= "string" then return -1, "roman numeral not a string" end
    local rev = roman:reverse()
    local raising = true
    local last = 0
    local result = 0
    for i = 1, #rev do
        local c = rev:sub(i, i)
        local next = roman_numerals[c]
        if next == nil then return -1, "roman numeral contains illegal character" end
        if next > last then
            result = result + next
            raising = true
        elseif next < last then
            result = result - next
            raising = false
        elseif raising then
            result = result + next
        else
            result = result - next
        end
        last = next
    end
end
```

```
                result = result - next
            end
            last = next
        end
        return result
    end

-- Converts a given integer between 0 and 100 to English text (e.g. 47 -> forty-
local function numeral_to_english_less_100(num, ordinal, plural, zero)
    local terminal_ones, terminal_tens
    if ordinal then
        terminal_ones = ones_position_ord
        terminal_tens = tens_position_ord
    elseif plural then
        terminal_ones = ones_position_plural
        terminal_tens = tens_position_plural
    else
        terminal_ones = ones_position
        terminal_tens = tens_position
    end

    if num == 0 and zero ~= nil then
        return zero
    elseif num < 20 then
        return terminal_ones[num]
    elseif num % 10 == 0 then
        return terminal_tens[num / 10]
    else
        return tens_position[math.floor(num / 10)] .. '-' .. terminal_ones[num % 10]
    end
end

local function standard_suffix(ordinal, plural)
    if ordinal then return 'th' end
    if plural then return 's' end
    return ''
end

-- Converts a given integer (in string form) between 0 and 1000 to English text
local function numeral_to_english_less_1000(num, use_and, ordinal, plural, zero)
    num = tonumber(num)
    if num < 100 then
        return numeral_to_english_less_100(num, ordinal, plural, zero)
    elseif num % 100 == 0 then
        return ones_position[num/100] .. ' hundred' .. standard_suffix(ordinal, plural)
    else
        return ones_position[math.floor(num/100)] .. ' hundred ' .. (use_and and 'and ') .. numeral_to_english_less_100(num % 100, ordinal, plural, zero)
    end
end

-- Converts an ordinal in English text from 'zeroth' to 'ninety-ninth' inclusive
local function english_to_ordinal(english)
    local eng = string.lower(english or '')

    local engord_lt20 = {} -- ones_position_ord{} keys & values swapped
    for k, v in pairs(ones_position_ord) do
        engord_lt20[v] = k
    end

    if engord_lt20[eng] then
        return engord_lt20[eng] -- e.g. first -> 1
    elseif engord_tens_end[eng] then
        return engord_tens_end[eng] -- e.g. ninetieth -> 90
    else
        return nil
    end
end
```



```
        local tens, ones = string.match(eng, '^([a-z]+)[%s%-]+([a-z]+)$')
        if tens and ones then
            local tens_cont = eng_tens_cont[tens]
            local ones_end = engord_lt20[ones]
            if tens_cont and ones_end then
                return tens_cont + ones_end -- e.g. ninety-ninth
            end
        end
    end
end
return -1 -- Failed
end

-- Converts a number in English text from 'zero' to 'ninety-nine' inclusive to a
local function english_to_numeral(english)
    local eng = string.lower(english or '')

    local eng_lt20 = { ['single'] = 1 } -- ones_position{} keys & values swapped
    for k, v in pairs(ones_position) do
        eng_lt20[v] = k
    end

    if eng_lt20[eng] then
        return eng_lt20[eng] -- e.g. one -> 1
    elseif eng_tens_cont[eng] then
        return eng_tens_cont[eng] -- e.g. ninety -> 90
    else
        local tens, ones = string.match(eng, '^([a-z]+)[%s%-]+([a-z]+)$')
        if tens and ones then
            local tens_cont = eng_tens_cont[tens]
            local ones_end = eng_lt20[ones]
            if tens_cont and ones_end then
                return tens_cont + ones_end -- e.g. ninety-nine
            end
        end
    end
end
return -1 -- Failed
end

-- Converts a number expressed as a string in scientific notation to a string in
-- e.g. 1.23E5 -> 123000, 1.23E-5 = .0000123. Conversion is exact, no rounding is
local function scientific_notation_to_decimal(num)
    local exponent, subs = num:gsub("^%-?%d*%?.?%d*%-?[Ee]([+%-]?%d+)$", "%1")
    if subs == 0 then return num end -- Input not in scientific notation, just
    exponent = tonumber(exponent)

    local negative = num:find("^%-")
    local _, decimal_pos = num:find("%.")
    -- Mantissa will consist of all decimal digits with no decimal point
    local mantissa = num:gsub("^%-?(%d*)%.?(%d*)%-?[Ee][+%-]?%d+$", "%1%2")
    if negative and decimal_pos then decimal_pos = decimal_pos - 1 end
    if not decimal_pos then decimal_pos = #mantissa + 1 end

    -- Remove leading zeros unless decimal point is in first position
    while decimal_pos > 1 and mantissa:sub(1,1) == '0' do
        mantissa = mantissa:sub(2)
        decimal_pos = decimal_pos - 1
    end
    -- Shift decimal point right for exponent > 0
    while exponent > 0 do
        decimal_pos = decimal_pos + 1
        exponent = exponent - 1
        if decimal_pos > #mantissa + 1 then mantissa = mantissa .. '0' end
        -- Remove leading zeros unless decimal point is in first position
        while decimal_pos > 1 and mantissa:sub(1,1) == '0' do
```

```
        mantissa = mantissa:sub(2)
        decimal_pos = decimal_pos - 1
    end
end
-- Shift decimal point left for exponent < 0
while exponent < 0 do
    if decimal_pos == 1 then
        mantissa = '0' .. mantissa
    else
        decimal_pos = decimal_pos - 1
    end
    exponent = exponent + 1
end

-- Insert decimal point in correct position and return
return (negative and '-' or '') .. mantissa:sub(1, decimal_pos - 1) .. '
end

-- Rounds a number to the nearest integer (NOT USED)
local function round_num(x)
    if x%1 >= 0.5 then
        return math.ceil(x)
    else
        return math.floor(x)
    end
end

-- Rounds a number to the nearest two-word number (round = up, down, or "on" for
-- Numbers with two digits before the decimal will be rounded to an integer as sp
-- Larger numbers will be rounded to a number with only one nonzero digit in front
-- Negative sign is preserved and does not count towards word limit.
local function round_for_english(num, round)
    -- If an integer with at most two digits, just return
    if num:find("^%-?%d?%d%.?") then return num end

    local negative = num:find("^%-")
    if negative then
        -- We're rounding magnitude so flip it
        if round == 'up' then round = 'down' elseif round == 'down' then
    end

    -- If at most two digits before decimal, round to integer and return
    local _, _, small_int, trailing_digits, round_digit = num:find("^%-(%d?%d%.?)")
    if small_int then
        if small_int == '' then small_int = '0' end
        if (round == 'up' and trailing_digits:find('[1-9]')) or (round ==
            small_int = tostring(tonumber(small_int) + 1)
        end
        return (negative and '-' or '') .. small_int
    end

    -- When rounding up, any number with > 1 nonzero digit will round up (e.g
    local nonzero_digits = 0
    for digit in num:gfind("[1-9]") do
        nonzero_digits = nonzero_digits + 1
    end

    num = num:gsub("%.%d*$", "") -- Remove decimal part
    -- Second digit used to determine which way to round lead digit
    local _, _, lead_digit, round_digit, round_digit_2, rest = num:find("^%-(%d?%d%.?)")
    if tonumber(lead_digit .. round_digit) < 20 and (1 + #rest) % 3 == 0 then
        -- In English numbers < 20 are one word so put 2 digits in lead a
        lead_digit = lead_digit .. round_digit
        round_digit = round_digit_2
    end
end
```



```
else
    rest = round_digit_2 .. rest
end

if (round == 'up' and nonzero_digits > 1) or (round == 'on' and tonumber(
    lead_digit = tostring(tonumber(lead_digit) + 1)
end
-- All digits but lead digit will turn to zero
rest = rest:gsub("%d", "0")
return (negative and '-' or '') .. lead_digit .. '0' .. rest
end

local denominators = {
    [2] = { 'half', plural = 'halves' },
    [3] = { 'third' },
    [4] = { 'quarter', us = 'fourth' },
    [5] = { 'fifth' },
    [6] = { 'sixth' },
    [8] = { 'eighth' },
    [9] = { 'ninth' },
    [10] = { 'tenth' },
    [16] = { 'sixteenth' },
}

-- Return status, fraction where:
-- status is a string:
--     "finished" if there is a fraction with no whole number;
--     "ok" if fraction is empty or valid;
--     "unsupported" if bad fraction;
-- fraction is a string giving (numerator / denominator) as English text, or is
-- Only unsigned fractions with a very limited range of values are supported,
-- except that if whole is empty, the numerator can use "-" to indicate negative
-- whole (string or nil): nil or "" if no number before the fraction
-- numerator (string or nil): numerator, if any (default = 1 if a denominator is
-- denominator (string or nil): denominator, if any
-- sp_us (boolean): true if sp=us
-- negative_word (string): word to use for negative sign, if whole is empty
-- use_one (boolean): false: 2+1/2 → "two and a half"; true: "two and one-half"
local function fraction_to_english(whole, numerator, denominator, sp_us, negative)
    if numerator or denominator then
        local finished = (whole == nil or whole == '')
        local sign = ''
        if numerator then
            if finished and numerator:sub(1, 1) == '-' then
                numerator = numerator:sub(2)
                sign = negative_word .. ' '
            end
        end
        else
            numerator = '1'
        end
        if not numerator:match('^%d+$') or not denominator or not denominator:match('^%d+$') then
            return 'unsupported', ''
        end
        numerator = tonumber(numerator)
        denominator = tonumber(denominator)
        local dendata = denominators[denominator]
        if not (dendata and 1 <= numerator and numerator <= 99) then
            return 'unsupported', ''
        end
        local numstr, denstr
        local sep = '-'
        if numerator == 1 then
            denstr = sp_us and dendata.us or dendata[1]
            if finished or use_one then
```



```

        numstr = 'one'
    elseif denstr:match('^[aeiou]') then
        numstr = 'an'
        sep = ' '
    else
        numstr = 'a'
        sep = ' '
    end
end
else
    numstr = numeral_to_english_less_100(numerator)
    denstr = dendata.plural
    if not denstr then
        denstr = (sp_us and dendata.us or dendata[1]) ..
    end
end
end
if finished then
    return 'finished', sign .. numstr .. sep .. denstr
end
return 'ok', ' and ' .. numstr .. sep .. denstr
end
return 'ok', ''
end

-- Takes a decimal number and converts it to English text.
-- Return nil if a fraction cannot be converted (only some numbers are supported)
-- num (string or nil): the number to convert.
--     Can be an arbitrarily large decimal, such as "-123456789123456789.345", and
--     can use scientific notation (e.g. "1.23E5").
--     May fail for very large numbers not listed in "groups" such as "1E4000".
--     num is nil if there is no whole number before a fraction.
-- numerator (string or nil): numerator of fraction (nil if no fraction)
-- denominator (string or nil): denominator of fraction (nil if no fraction)
-- capitalize (boolean): whether to capitalize the result (e.g. 'One' instead of 'one')
-- use_and (boolean): whether to use the word 'and' between tens/ones place and fraction
-- hyphenate (boolean): whether to hyphenate all words in the result, useful as a
-- ordinal (boolean): whether to produce an ordinal (e.g. 'first' instead of 'one')
-- plural (boolean): whether to pluralize the resulting number
-- links: nil: do not add any links; 'on': link "billion" and larger to Orders of
--     any other text: list of numbers to link (e.g. "billion,quadrillion")
-- negative_word: word to use for negative sign (typically 'negative' or 'minus')
-- round: nil or '': no rounding; 'on': round to nearest two-word number; 'up'/'down':
--     round up/down to nearest two-word number
-- zero: word to use for value '0' (nil to use default)
-- use_one (boolean): false: 2+1/2 -> "two and a half"; true: "two and one-half"
local function numeral_to_english(num, numerator, denominator, capitalize, use_and,
    if not negative_word then
        if use_and then
            -- TODO Should 'minus' be used when do not have sp=us?
            --     If so, need to update testcases, and need to fix
            -- negative_word = 'minus'
            negative_word = 'negative'
        else
            negative_word = 'negative'
        end
    end
end
local status, fraction_text = fraction_to_english(num, numerator, denominator,
if status == 'unsupported' then
    return nil
end
if status == 'finished' then
    -- Input is a fraction with no whole number.
    -- Hack to avoid executing stuff that depends on num being a number
    local s = fraction_text
    if hyphenate then s = s:gsub("%s", "-") end
    if capitalize then s = s:gsub("^%l", string.upper) end

```



```
        return s
    end
    num = scientific_notation_to_decimal(num)
    if round and round ~= '' then
        if round ~= 'on' and round ~= 'up' and round ~= 'down' then
            error("Invalid rounding mode")
        end
        num = round_for_english(num, round)
    end

    -- Separate into negative sign, num (digits before decimal), decimal_places
    local MINUS = '-' -- Unicode U+2212 MINUS SIGN (may be in values from [
    if num:sub(1, #MINUS) == MINUS then
        num = '-' .. num:sub(#MINUS + 1) -- replace MINUS with '-'
    elseif num:sub(1, 1) == '+' then
        num = num:sub(2) -- ignore any '+'
    end
    local negative = num:find("^%-")
    local decimal_places, subs = num:gsub("^%-?%d*%.(%d+)$", "%1")
    if subs == 0 then decimal_places = nil end
    num, subs = num:gsub("^%-?(%d*)%.?%d*$", "%1")
    if num == '' and decimal_places then num = '0' end
    if subs == 0 or num == '' then error("Invalid decimal numeral") end

    -- For each group of 3 digits except the last one, print with appropriate
    local s = ''
    while #num > 3 do
        if s ~= '' then s = s .. ' ' end
        local group_num = math.floor((#num - 1) / 3)
        local group = groups[group_num]
        local group_digits = #num - group_num*3
        s = s .. numeral_to_english_less_1000(num:sub(1, group_digits), t
        if links and (((links == 'on' and group_num >= 3) or links:find('o
            s = s .. '['Orders_of_magnitude_(numbers)#10' .. group_nu
        else
            s = s .. group
        end
        num = num:sub(1 + group_digits)
        num = num:gsub("^0*", "") -- Trim leading zeros
    end

    -- Handle final three digits of integer part
    if s ~= '' and num ~= '' then
        if #num <= 2 and use_and then
            s = s .. ' and '
        else
            s = s .. ' '
        end
    end

    if s == '' or num ~= '' then
        s = s .. numeral_to_english_less_1000(num, use_and, ordinal, plu
    elseif ordinal or plural then
        -- Round numbers like "one million" take standard suffixes for o
        s = s .. standard_suffix(ordinal, plural)
    end

    -- For decimal places (if any) output "point" followed by spelling out d
    if decimal_places then
        s = s .. ' point'
        for i = 1, #decimal_places do
            s = s .. ' ' .. ones_position[tonumber(decimal_places:sub
        end
    end
end
```

```
s = s:gsub("^%s*(.)%s*$", "%1") -- Trim whitespace
if ordinal and plural then s = s .. 's' end -- s suffix works for all of
if negative and s ~= zero then s = negative_word .. ' ' .. s end
s = s:gsub("negative zero", "zero")
s = s .. fraction_text
if hyphenate then s = s:gsub("%s", "-") end
if capitalize then s = s:gsub("^%l", string.upper) end
return s
end

local function _numeral_to_english2(args)
    local num = args.num

    if (not tonumber(num)) then
        num = num:gsub("^%s*(.)%s*$", "%1") -- Trim whitespace
        num = num:gsub(",", "") -- Remove commas
        num = num:gsub("<span[^<>]*></span>", "") -- Generated by Template
        if num ~= '' then -- a fraction may have an empty whole number
            if not num:find("^%-?%d*%?.?%d*%-?[Ee]?[+%-]?%d*$") then
                -- Input not in a valid format, try to pass it through
                -- if that produces a number (e.g. "3 + 5" will produce 8)
                num = mw.getCurrentFrame():preprocess('{{#expr:
            end
        end

    end

    -- Pass args from frame to helper function
    return _numeral_to_english(
        num,
        args['numerator'],
        args['denominator'],
        args['capitalize'],
        args['use_and'],
        args['hyphenate'],
        args['ordinal'],
        args['plural'],
        args['links'],
        args['negative_word'],
        args['round'],
        args['zero'],
        args['use_one']
    ) or ''
end

local p = { -- Functions that can be called from another module
    roman_to_numeral = roman_to_numeral,
    spell_number = _numeral_to_english,
    spell_number2 = _numeral_to_english2,
    english_to_ordinal = english_to_ordinal,
    english_to_numeral = english_to_numeral,
}

function p.roman_to_numeral(frame) -- Callable via {{#invoke:ConvertNumeric|roman_to_numeral}}
    return roman_to_numeral(frame.args[1])
end

function p.english_to_ordinal(frame) -- callable via {{#invoke:ConvertNumeric|english_to_ordinal}}
    return english_to_ordinal(frame.args[1])
end

function p.english_to_numeral(frame) -- callable via {{#invoke:ConvertNumeric|english_to_numeral}}
    return english_to_numeral(frame.args[1])
end
```

```
function p.numeral_to_english(frame)
  local args = frame.args
  local num = args[1]
  num = num:gsub("^%s*(.)%s*$", "%1") -- Trim whitespace
  num = num:gsub(",", "") -- Remove commas
  num = num:gsub("<span[<>]*></span>", "") -- Generated by Template:age
  if num ~= '' then -- a fraction may have an empty whole number
    if not num:find("^%-?%d*%?.?%d*%-?[Ee]?[+%-]?%d*$") then
      -- Input not in a valid format, try to pass it through #e
      -- if that produces a number (e.g. "3 + 5" will become "8
      num = frame:preprocess('{{#expr: ' .. num .. '}}')
    end
  end

end

-- Pass args from frame to helper function
return _numeral_to_english(
  num,
  args['numerator'],
  args['denominator'],
  args['case'] == 'U' or args['case'] == 'u',
  args['sp'] ~= 'us',
  args['adj'] == 'on',
  args['ord'] == 'on',
  args['pl'] == 'on',
  args['lk'],
  args['negative'],
  args['round'],
  args['zero'],
  args['one'] == 'one' -- experiment: using '|one=one' makes fract
) or ''
end

---- recursive function for p.decToHex
local function decToHexDigit(dec)
  local dig = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E"}
  local div = math.floor(dec/16)
  local mod = dec-(16*div)
  if div >= 1 then return decToHexDigit(div)..dig[mod+1] else return dig[mod+1]
end -- I think this is supposed to be done with a tail call but first I want some

---- finds all the decimal numbers in the input text and hexes each of them
function p.decToHex(frame)
  local args=frame.args
  local parent=frame.getParent(frame)
  local pargs={}
  if parent then pargs=parent.args end
  local text=args[1] or pargs[1] or ""
  local minlength=args.minlength or pargs.minlength or 1
  minlength=tonumber(minlength)
  local prowl=mw.usttring.gmatch(text,"(.)%d+")
  local output=""
  repeat
    local chaff,dec=prowl()
    if not(dec) then break end
    local hex=decToHexDigit(dec)
    while (mw.usttring.len(hex)<minlength) do hex="0"..hex end
    output=output..chaff..hex
  until false
  local chaff=mw.usttring.match(text,"%D+$") or ""
  return output..chaff
end

return p
```

