with-chatgpt

in context neovim, help me create several custom lua function but mainly written with vimscript "vim.cmd", the function purpose to enhance coding experience

awesome, try to discover more custom lua function with vimscript inside

what about additional function we still not talk about in both previous responses

---

Here are **several custom Lua functions that are *defined in Lua* but mostly *implemented using Vimscript* via `vim.cmd` / `vim.fn`**, designed specifically to enhance daily coding workflows. You can drop these into `lua/mycoding.lua` or directly inside your `init.lua`.

Each function uses Vimscript logic embedded inside Lua — a hybrid approach that gives you both power and simplicity.

✔️ 1. Smart Trim Trailing Whitespace (Vimscript Core)

```lua
local M = {}

-- Remove trailing whitespace while remembering cursor position
function M.trim_trailing_whitespace()
  vim.cmd([[
    let l:pos = getpos(".")
    %s/\s\+$//e
    call setpos(".", l:pos)
  ]])
end
```

Usage:

```lua
vim.keymap.set("n", "<leader>tw", M.trim_trailing_whitespace)
```

✔️ 2. Toggle True/False (Vimscript substitution inside Lua)

Useful for config files, JSON, Lua, TS, etc.

```lua
function M.toggle_bool()
  vim.cmd([[
    let l:word = expand("<cword>")
    if l:word ==# "true"
      execute "normal! ciwfalse"
    elseif l:word ==# "false"
```

```
      execute "normal! ciwtrue"
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>tb", M.toggle_bool)
```

✔️ 3. Highlight TODO / FIXME / BUG (Adds match groups)

```
function M.highlight_todos()
  vim.cmd([[
    highlight TodoWord cterm=bold gui=bold guifg=#ffaf00
    match TodoWord /\v<(TODO|FIXME|BUG):?/
  ]])
end
```

Optional auto-enable:

```
vim.api.nvim_create_autocmd("BufEnter", {
  callback = M.highlight_todos
})
```

✔️ 4. Jump to the Last Cursor Position on File Open

Uses pure Vimscript logic inside Lua.

```
function M.restore_cursor()
  vim.cmd([[
    if line("'\"") >= 1 && line("'\"") <= line("$")
      exe "normal! g`\""
    endif
  ]])
end

vim.api.nvim_create_autocmd("BufReadPost", { callback = M.restore_cursor })
```

✔️ 5. Smart Join Lines but Preserve Indent (Better `J`)

```
function M.smart_join()
  vim.cmd([[
    let l:save = @"
    normal! mzJ
    silent! %s/^\s\+/ /e
    normal! `z
    let @" = l:save
  ]])
end
```

Usage:

```
vim.keymap.set("n", "J", M.smart_join)
```

✔️ 6. Surround Selection With Custom Characters (Mini Vimscript)

```
function M.surround_open_close(open, close)
  vim.cmd(string.format([[
    let l:op = '%s'
    let l:cp = '%s'
    normal! vi%c
    execute "normal! c" . l:op . "\<C-r>\"" . l:cp
  ]], open, close, '"'))
end

-- Convenience
function M.surround_quotes() M.surround_open_close('"', '"') end
function M.surround_brackets() M.surround_open_close('[', ']') end
```

Example keymaps:

```
vim.keymap.set("v", "<leader>sq", M.surround_quotes)
vim.keymap.set("v", "<leader>sb", M.surround_brackets)
```

✔️ 7. Quick Compile & Run C / C++ / Python depending on file type

```
function M.smart_run()
  vim.cmd([[
    let l:file = expand("%:p")
    let l:base = expand("%:t:r")
```

```
    let l:ext = expand("%:e")

    if l:ext ==# 'c'
      exec "!gcc -O2 " . l:file . " -o " . l:base
      exec "!./" . l:base
    elseif l:ext ==# 'cpp'
      exec "!g++ -std=c++20 -O2 " . l:file . " -o " . l:base
      exec "!./" . l:base
    elseif l:ext ==# 'py'
      exec "!python3 " . l:file
    else
      echo "Unsupported file type"
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>r", M.smart_run)
```

✔️ 8. Smart Line Duplication (Better than dd/p)

```
function M.duplicate_line()
  vim.cmd([[
    let l:pos = getpos(".")
    execute "normal! yyp"
    call setpos(".", l:pos)
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>d", M.duplicate_line)
```

✔️ 9. Toggle Relative Number + Cursorline + SignColumn (Coding Focus Mode)

```
function M.focus_mode()
  vim.cmd([[
    if &relativenumber
      set norelativenumber
      set nocursorline
      set signcolumn=no
    else
```

```
      set relativenumber
      set cursorline
      set signcolumn=yes
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>fm", M.focus_mode)
```

✔️ 10. Search Word Under Cursor in New Split (but Vimscript logic)

```
function M.search_in_split()
  vim.cmd([[
    let l:word = expand("<cword>")
    split
    execute "grep " . shellescape(l:word)
    copen
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>gs", M.search_in_split)
```

✔️ Want more?

I can generate additional:

- motions (jump between functions, classes, folds)
- text-objects (custom `if`, `af`, `ic`, etc.)
- code navigation helpers (goto test file, goto header/source)
- quick refactor helpers (rename variable via substitutions)
- hybrid Lua/Vimscript utilities tailored to your coding habits

Just tell me what next set you want.

Here comes another rich batch of **custom Lua functions powered internally by Vimscript** — all designed to boost navigation, editing, refactoring, and general coding comfort in Neovim.

These functions follow your requested pattern: **Lua wrapper → core logic done through `vim.cmd` and classic Vimscript features.**

You can drop them into:

lua/mycoding.lua

🔥 1. "Breadcrumb Jump" – Jump Back to Previous Function / Class

Uses search patterns for common languages.

```
function M.jump_prev_symbol()
  vim.cmd([[
    let l:pattern = '\v^(class|struct|fn|function|def|impl|enum|module|interface)\>'
    silent! execute "normal! ?".l:pattern."\<CR>zz"
  ]])
end
```

Usage:

```
vim.keymap.set("n", "[f", M.jump_prev_symbol)
```

🔥 2. "Breadcrumb Jump" – Jump Forward to Next Function / Class

```
function M.jump_next_symbol()
  vim.cmd([[
    let l:pattern = '\v^(class|struct|fn|function|def|impl|enum|module|interface)\>'
    silent! execute "normal! /".l:pattern."\<CR>zz"
  ]])
end
```

Usage:

```
vim.keymap.set("n", "]f", M.jump_next_symbol)
```

🔥 3. Toggle Comment Style for C-like Languages

```
function M.toggle_comment()
  vim.cmd([[
    let l:line = getline('.')
    if l:line =~ '^\s*//'
      execute "s#^\s*//##"
    else
      execute "s#^#// #"
```

```
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>cc", M.toggle_comment)
```

🔥 4. Duplicate Selected Block (Visual Mode Power)

```
function M.duplicate_block()
  vim.cmd([[
    normal! gv"xy
    normal! `>p
  ]])
end
```

Usage:

```
vim.keymap.set("v", "<leader>db", M.duplicate_block)
```

🔥 5. Toggle Between Header and Source (`.c` → `.h` , `.cpp` → `.hpp` , etc.)

```
function M.toggle_header_source()
  vim.cmd([[
    let l:file = expand("%:t")
    let l:root = expand("%:r")
    let l:ext = expand("%:e")

    if l:ext =~# '^\(c\|cpp\|cc\|cxx\)$'
      let l:targets = ['h', 'hpp', 'hh']
    else
      let l:targets = ['c', 'cpp', 'cc', 'cxx']
    endif

    for ext in l:targets
      let l:try = l:root . '.' . ext
      if filereadable(l:try)
        execute "edit " . l:try
        break
      endif
    endfor
```

```
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>hs", M.toggle_header_source)
```

🔥 6. Remove Empty Lines Around Cursor

```
function M.clean_blank_lines()
  vim.cmd([[
    silent! g/^\s*$/d
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>ce", M.clean_blank_lines)
```

🔥 7. "Buffer Peek" – Show Next/Prev Buffer in a Floating Window (Vimscript inside)

```
function M.peek_buffer(dir)
  vim.cmd([[
    let l:cur = bufnr('%')
    let l:bufs = filter(range(1, bufnr('$')), 'buflisted(v:val)')
    let l:i = index(l:bufs, l:cur)
  ]])

  if dir == "next" then
    vim.cmd("let l:target = l:bufs[(l:i + 1) % len(l:bufs)]")
  else
    vim.cmd("let l:target = l:bufs[(l:i - 1 + len(l:bufs)) % len(l:bufs)]")
  end

  vim.cmd([[
    execute "pedit +" l:target
    wincmd P
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>pn", function() M.peek_buffer("next") end)
vim.keymap.set("n", "<leader>pp", function() M.peek_buffer("prev") end)
```

🔥 8. Rename Word Under Cursor (Basic Refactor)

```
function M.rename_word()
  vim.cmd([[
    let l:old = expand('<cword>')
    let l:new = input("Rename " . l:old . " → ")

    if l:new != ""
      execute "%s/\\V" . escape(l:old, '/\') . "/" . l:new . "/g"
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>rn", M.rename_word)
```

🔥 9. Shortcuts to Wrap Current Line in Various Structures

Wrap in parentheses:

```
function M.wrap_paren()
  vim.cmd([[ execute "normal! I(" | execute "normal! A)" ]])
end
```

Wrap in braces:

```
function M.wrap_brace()
  vim.cmd([[ execute "normal! I{" | execute "normal! A}" ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>wp", M.wrap_paren)
vim.keymap.set("n", "<leader>{", M.wrap_brace)
```

🔥 10. "Jump to Error" – Fastest Way to Navigate Errors

```
function M.jump_error()
  vim.cmd([[
    if getqflist() == []
      echo "No errors"
    else
      cwindow
      cfirst
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>je", M.jump_error)
```

🔥 11. Show Symbol Outline in Vertical Split (ctags / LSP-friendly)

```
function M.outline_split()
  vim.cmd([[
    vnew
    setlocal buftype=nofile
    setlocal nobuflisted
    setlocal noswapfile
    execute "!ctags -x --sort=no " . expand('%:p')
    0r!ctags -x --sort=no " . expand('%:p')
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>os", M.outline_split)
```

🔥 12. Toggle Soft Wrap, Keep Indent & Cursor Column

```lua
function M.toggle_softwrap()
  vim.cmd([[
    if &wrap
      set nowrap
      set nolist
    else
      set wrap
      set linebreak
      set list
    endif
  ]])
end
```

Usage:

```lua
vim.keymap.set("n", "<leader>sw", M.toggle_softwrap)
```

🔥 Want more?

I can generate more sets such as:

📌 Code Navigation

- jump to next/prev TODO/FIXME
- auto-scroll to keep cursor centered
- jump between indentation blocks

📌 Editing Tools

- alignment helpers
- automatic bracket/brace generation
- text-object creators

📌 Refactoring Utilities

- extract variable/function
- convert string quotes
- indent/outdent blocks with custom rules

📌 C / C++ / Lua / Python enhancements

- Doxygen/Docstring generator
- switch/case template insertion
- automatic include/require resolver

Tell me the theme you want explored next, or say “go further” to keep the momentum.

Great — let’s go **even deeper** and generate a whole new set of **custom Lua functions powered internally by Vimscript**, focusing on *completely new areas we haven’t touched yet*. These functions target smarter movement, diagnostics, batch editing, UI helpers, refactoring, and structural code manipulation.

None of these repeat anything from the previous two responses.

⚡ 1. Jump Between Changed Hunks (Even Without Git Signs Plugin)

```
function M.jump_next_change()
  vim.cmd([[
    silent! normal! ]c
    zz
  ]])
end

function M.jump_prev_change()
  vim.cmd([[
    silent! normal! [c
    zz
  ]])
end
```

Usage:

```
vim.keymap.set("n", "]c", M.jump_next_change)
vim.keymap.set("n", "[c", M.jump_prev_change)
```

⚡ 2. Highlight Current Line Only Temporarily (Flashline)

A small visual cue when jumping around code.

```
function M.flash_line()
  vim.cmd([[
    highlight FlashLine guibg=#444444
    execute "match FlashLine /\%" . line('.') . "l/"
    redraw
    sleep 150m
    match none
  ]])
end
```

Usage:

```
vim.keymap.set("n", "gl", M.flash_line)
```

⚡ 3. Compare Current Buffer to Disk Version (Smart Diff)

```
function M.diff_disk()
  vim.cmd([[
    diffthis
    vnew | read #
    normal! ggdd
    diffthis
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>df", M.diff_disk)
```

⚡ 4. Convert Tabs to Spaces OR Spaces to Tabs

```
function M.tabs_to_spaces()
  vim.cmd([[ %s/\t/  /g ]])
end

function M.spaces_to_tabs()
  vim.cmd([[ %s/  /\t/g ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>ts", M.tabs_to_spaces)
vim.keymap.set("n", "<leader>st", M.spaces_to_tabs)
```

⚡ 5. Select the Whole Function (Simple Vimscript Text Object)

```
function M.select_function()
  vim.cmd([[
    normal! [[V]]
  ]])
end
```

Usage:

```
vim.keymap.set("n", "vaf", M.select_function)
```

⚡ 6. Sort Imports / Includes Automatically

Good for C, Python, JS, etc.

```
function M.sort_imports()
  vim.cmd([[
    g/^#\s*include/ sort
    g/^import / sort
    g/^using / sort
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>si", M.sort_imports)
```

⚡ 7. Smart Insert Header Guard for C/C++

```
function M.insert_header_guard()
  vim.cmd([[
    let l:name = toupper(substitute(expand('%:t:r'), '\W', '_', 'g'))
    let l:guard = l:name . "_H"

    call append(0, "#ifndef " . l:guard)
    call append(1, "#define " . l:guard)
    call append(line('$'), "#endif /* " . l:guard . " */")
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>hg", M.insert_header_guard)
```

⚡ 8. Extract Selected Lines into a New Buffer

```
function M.extract_selection()
  vim.cmd([[
    normal! gv"xy
```

```
    new
    call setline(1, split(@x, "\n"))
  ]])
end
```

Usage:

```
vim.keymap.set("v", "<leader>x", M.extract_selection)
```

⚡ 9. Save All + Close Unmodified Buffers

```
function M.clean_workspace()
  vim.cmd([[
    wall
    for b in range(1, bufnr('$'))
      if buflisted(b) && !getbufvar(b, '&modified')
        execute "bdelete " . b
      endif
    endfor
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>cw", M.clean_workspace)
```

⚡ 10. Smart “Focus on Current Function” – Fold Everything Else

```
function M.focus_function()
  vim.cmd([[
    normal! zM
    normal! zv
    normal! zO
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>ff", M.focus_function)
```

⚡ 11. Surround Word Under Cursor With HTML Tags

```
function M.surround_html(tag)
  vim.cmd(string.format([[
    let l:word = expand("<cword>")
    execute "normal! ciw<%s>" . l:word . "</%s>"
  ]], tag, tag))
end
```

Usage examples:

```
vim.keymap.set("n", "<leader>sh1", function() M.surround_html("h1") end)
vim.keymap.set("n", "<leader>sp",  function() M.surround_html("p") end)
```

⚡ 12. Align Code By Specific Character (simple but powerful)

```
function M.align(char)
  vim.cmd(string.format([[
    execute ':%s/\v\s*%s/\=repeat(" ", 20 - virtcol(".")) . "%s"/g'
  ]], char, char))
end
```

Usage:

```
vim.keymap.set("n", "<leader>a=", function() M.align("=") end)
```

⚡ 13. Toggle Quickfix Window (Smart)

```
function M.toggle_qf()
  vim.cmd([[
    if empty(filter(getwininfo(), 'v:val.quickfix'))
      copen
    else
      cclose
    endif
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>qf", M.toggle_qf)
```

⚡ 14. Create a Temporary Scratch Buffer

```
function M.scratch()
  vim.cmd([[
    enew
    setlocal buftype=nofile bufhidden=hide noswapfile
  ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>ns", M.scratch)
```

⚡ 15. Copy File Path / Directory Path to Clipboard

```
function M.copy_path()
  vim.cmd([[ let @+ = expand("%:p") ]])
end

function M.copy_dir()
  vim.cmd([[ let @+ = expand("%:p:h") ]])
end
```

Usage:

```
vim.keymap.set("n", "<leader>cp", M.copy_path)
vim.keymap.set("n", "<leader>cd", M.copy_dir)
```

⚡ Want to go even further?

I can generate **more functions in any category** you like:

➤ **Navigation**

➤ **Editing / macro-likes**

➤ **AI-assisted coding helpers**

➤ **File/project tools**

➤ **Git helpers (non-plugin)**

➤ **UI magic (good for TUI workflows)**

➤ **Refactoring utilities**

➤ **Code generation**

➤ **Diagnostic/LSP wrappers implemented using Vimscript**

Tell me a theme OR say **“go deeper”** and I’ll push into brand new territory.
