

with-chatgpt

in state of neovim lua configuration, suggest me several custom functions that are plug-and-play

awesome, try to discover more useful yet ergonomic functions

based on both previous responses, suggest me functions that are still not mentioned yet

what about more functions, but inspired by some plugins' functionality

Here are several **plug-and-play custom Lua functions** you can drop directly into your Neovim configuration. Each function is self-contained, doesn't require external plugins (unless stated), and works immediately once pasted into your `init.lua` or any Lua module.

✓ 1. Smart Move to Last Location

Automatically jump to the last cursor position when reopening a file.

```
function _G.restore_last_location()
  local last_pos = vim.fn.line("'\"")
  if last_pos > 1 and last_pos <= vim.fn.line("$") then
    vim.cmd("normal! g`\"")
  end
end

vim.api.nvim_create_autocmd("BufReadPost", { callback = _G.restore_last_location })
```

✓ 2. Toggle Numbering (absolute ↔ relative)

Quickly switch between line number modes.

```
function _G.toggle_number_mode()
  if vim.wo.relativenumber then
    vim.wo.relativenumber = false
    vim.wo.number = true
  else
    vim.wo.relativenumber = true
  end
end

vim.keymap.set("n", "<leader>tn", _G.toggle_number_mode, { desc = "Toggle number mode" })
```

✓ 3. Reload Current Lua File

Great for rapidly testing your config.

```
function _G.reload_current_lua()
    local file = vim.fn.expand("%")
    if file:match("%.lua$") then
        dofile(file)
        print("Reloaded " .. file)
    else
        print("Not a Lua file")
    end
end

vim.keymap.set("n", "<leader>rl", _G.reload_current_lua)
```

✓ 4. Simple Floating Terminal

Creates a minimal floating terminal popup.

```
function _G.open_float_term()
    local buf = vim.api.nvim_create_buf(false, true)
    local width = math.floor(vim.o.columns * 0.8)
    local height = math.floor(vim.o.lines * 0.8)
    local row = math.floor((vim.o.lines - height) / 2)
    local col = math.floor((vim.o.columns - width) / 2)

    vim.api.nvim_open_win(buf, true, {
        relative = "editor",
        row = row,
        col = col,
        width = width,
        height = height,
        style = "minimal",
        border = "rounded"
    })
    vim.fn.termopen(vim.o.shell)
end

vim.keymap.set("n", "<leader>ft", _G.open_float_term, { desc = "Floating terminal"
})
```

✓ 5. Quick Buffer Wipe

Delete current buffer without closing the window layout.

```
function _G.smart_wipe()
    local bufnr = vim.api.nvim_get_current_buf()
    vim.cmd("bprevious")
    vim.cmd("bdelete " .. bufnr)
end
```

```
vim.keymap.set("n", "<leader>bd", _G.smart_wipe, { desc = "Wipe buffer" })
```

✓ 6. Highlight Under Cursor (flash for visibility)

Momentarily highlight the word under your cursor.

```
function _G.flash_word()
    local matches = vim.fn.matchadd("Search", "\\<" .. vim.fn.expand("<cword>") .. "\\>")
    vim.defer_fn(function()
        vim.fn.matchdelete(matches)
    end, 300)
end

vim.keymap.set("n", "<leader>fw", _G.flash_word)
```

✓ 7. Copy Current File Path to Clipboard

Supports absolute, relative, and filename-only.

```
function _G.copy_path(mode)
    local path = nil
    if mode == "absolute" then
        path = vim.fn.expand("%:p")
    elseif mode == "relative" then
        path = vim.fn.expand("%")
    elseif mode == "file" then
        path = vim.fn.expand("%:t")
    end
    if path then
        vim.fn.setreg("+", path)
        print("Copied: " .. path)
    end
end

vim.keymap.set("n", "<leader>pa", function() _G.copy_path("absolute") end)
vim.keymap.set("n", "<leader>pr", function() _G.copy_path("relative") end)
vim.keymap.set("n", "<leader>pf", function() _G.copy_path("file") end)
```

✓ 8. Toggle Color Column

Useful for writing code with width guidelines.

```
function _G.toggle_color_column()
  if vim.wc.colorcolumn == "" then
    vim.wc.colorcolumn = "80"
  else
    vim.wc.colorcolumn = ""
  end
end

vim.keymap.set("n", "<leader>tc", _G.toggle_color_column)
```

Want even more plug-and-play functions?

I can generate:

✓ window management helpers ✓ search/navigation utilities ✓ diagnostics/LSP helpers ✓ text-editing shortcuts
✓ buffer/tab/split automation ✓ project-specific functions

Just tell me what category you want next!

Here are **more ergonomic, practical, plug-and-play Lua functions** you can immediately drop into your Neovim setup. These focus on *speed, comfort, and reducing repetitive actions*.

★ 9. Quick Replace Word Under Cursor

Opens a command-line replace prompt with the current word already inserted.

```
function _G.replace_cword()
  local word = vim.fn.expand("<cword>")
  vim.api.nvim_feedkeys(":%s/" .. word .. "/", "n", false)
end

vim.keymap.set("n", "<leader>sr", _G.replace_cword, { desc = "Replace word under cursor" })
```

★ 10. Toggle Diagnostics (LSP)

For quickly cleaning up the screen when errors are distracting.

```
local diagnostics_enabled = true

function _G.toggle_diagnostics()
  diagnostics_enabled = not diagnostics_enabled
  if diagnostics_enabled then
    vim.diagnostic.enable()
  else
    vim.diagnostic.disable()
  end
end
```

```
end
end

vim.keymap.set("n", "<leader>td", _G.toggle_diagnostics, { desc = "Toggle
diagnostics" })
```

★ 11. Duplicate Current Line or Visual Selection

Behaves like many editors.

```
function _G.duplicate()
  local mode = vim.fn.mode()
  if mode == "v" or mode == "V" then
    vim.cmd("'<,'>copy '>")
  else
    vim.cmd("copy .")
  end
end

vim.keymap.set({"n", "v"}, "<leader>dd", _G.duplicate, { desc = "Duplicate
line/selection" })
```

★ 12. Trim Trailing Whitespace

But *only in the visible lines*, not the entire file.

```
function _G.trim_visible_whitespace()
  local first = vim.fn.line("w0")
  local last = vim.fn.line("w$")
  vim.cmd(first .. ", " .. last .. "s/\\s\\+$/e")
end

vim.keymap.set("n", "<leader>tw", _G.trim_visible_whitespace)
```

★ 13. Yank Entire Buffer (consistent across all states)

Handy for copying whole content quickly.

```
function _G.yank_buffer()
  vim.cmd("%y+")
  print("Buffer yanked to clipboard")
end
```

```
vim.keymap.set("n", "<leader>ya", _G.yank_buffer, { desc = "Yank entire buffer" })
```

★ 14. Better Quickfix Navigation (wrap around)

Jump to next/prev quickfix item and wrap when reaching edges.

```
function _G.qf_next()
    local idx = vim.fn.getqflist({ idx = 0 }).idx
    local size = #vim.fn.getqflist()
    vim.cmd("cnext")
    if vim.fn.getqflist({ idx = 0 }).idx == idx then
        vim.cmd("cc 1") -- wrap
    end
end

function _G.qf_prev()
    local idx = vim.fn.getqflist({ idx = 0 }).idx
    vim.cmd("cprev")
    if vim.fn.getqflist({ idx = 0 }).idx == idx then
        vim.cmd("cc $") -- wrap
    end
end

vim.keymap.set("n", "]q", _G.qf_next)
vim.keymap.set("n", "[q", _G.qf_prev)
```

★ 15. Toggle Wrap + Adjust UI Automatically

Enable wrapping, linebreak, indent folding, etc.

```
function _G.toggle_wrap_mode()
    if vim.wo.wrap then
        vim.wo.wrap = false
        vim.wo.linebreak = false
        vim.wo.breakindent = false
    else
        vim.wo.wrap = true
        vim.wo.linebreak = true
        vim.wo.breakindent = true
    end
end

vim.keymap.set("n", "<leader>tw", _G.toggle_wrap_mode)
```

★ 16. Toggle Rendered Markdown Preview (built-in)

Uses Neovim's terminal + markdown viewer (`glow` , `mdcat` , etc.). Works if you have one installed; otherwise harmless.

```
function _G.preview_markdown()
  local file = vim.fn.expand("%:p")
  local term = "glow " .. file

  vim.cmd("botright split | resize 20 | terminal " .. term)
end

vim.keymap.set("n", "<leader>mp", _G.preview_markdown)
```

★ 17. Open NOTE File for Current Project

Automatically opens `notes.md` in project root.

```
function _G.open_project_notes()
  local root = vim.fn.getcwd()
  local file = root .. "/notes.md"
  vim.cmd("edit " .. file)
end

vim.keymap.set("n", "<leader>pn", _G.open_project_notes)
```

★ 18. Switch Between Last Two Files

A more reliable alternative to `<C-^>` .

```
function _G.switch_last_two()
  local alt = vim.fn.expand("#")
  if alt == "" then return end
  vim.cmd("edit " .. alt)
end

vim.keymap.set("n", "<leader><leader>", _G.switch_last_two)
```

★ 19. Fuzzy Search for Words Under Cursor (no plugin)

Quick grep without telescope/ripgrep dependency.

```
function _G.search_word_project()
    local word = vim.fn.expand("<cword>")
    vim.cmd("vimgrep /" .. word .. "/gj **/*")
    vim.cmd("copen")
end

vim.keymap.set("n", "<leader>gw", _G.search_word_project)
```

★ 20. Bookmark Line Without Plugin

Simple file-local bookmarking.

```
local bookmarks = {}

function _G.toggle_bookmark()
    local file = vim.fn.expand("%:p")
    local line = vim.fn.line(".")
    bookmarks[file] = bookmarks[file] or {}
    if bookmarks[file][line] then
        bookmarks[file][line] = nil
        print("Bookmark removed")
    else
        bookmarks[file][line] = true
        print("Bookmark added")
    end
end

function _G.jump_bookmarks()
    local file = vim.fn.expand("%:p")
    bookmarks[file] = bookmarks[file] or {}
    for l, _ in pairs(bookmarks[file]) do
        vim.cmd(": " .. l)
    end
    return
end

vim.keymap.set("n", "<leader>bm", _G.toggle_bookmark)
vim.keymap.set("n", "<leader>bj", _G.jump_bookmarks)
```

Want more categories?

I can generate another batch focused on any of these:

- ◆ **Editing & Text Manipulation**

- ♦ **Window / Buffer UX**
- ♦ **LSP workflow helpers**
- ♦ **Git helpers**
- ♦ **Project navigation**
- ♦ **UI enhancements**
- ♦ **Terminal integration**

Tell me what direction you want next!

Here is a fresh batch of **new functions not mentioned previously**, chosen to fill the *remaining gaps* in ergonomics, workflow, navigation, windowing, file management, and quality-of-life behavior in Neovim. All are **plug-and-play**, no plugins needed.

21. Restore Cursor After Search Jump

When jumping with `n / N`, store the previous position and allow going back instantly.

```
function _G.jump_back()
  vim.cmd("normal! g`'")
end

vim.keymap.set("n", "<leader>jb", _G.jump_back, { desc = "Jump back to pre-search position" })
```

22. Select the Entire Function Block

Automatically selects the function you're inside (Lua/C/JavaScript friendly).

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

vim.keymap.set("n", "<leader>sf", _G.select_function)
```

23. Toggle Spell Check + Convenient Popups

Ideal for writing docs without permanently enabling spell mode.

```
function _G.toggle_spelling()
  vim.wo.spell = not vim.wo.spell
  print("Spell check:", vim.wo.spell)
end
```

```
vim.keymap.set("n", "<leader>ts", _G.toggle_spelling)
```

NEW 24. Toggle Statusline Visibility

Minimal reading mode.

```
function _G.toggle_statusline()
  if vim.o.laststatus == 0 then
    vim.o.laststatus = 3
  else
    vim.o.laststatus = 0
  end
end

vim.keymap.set("n", "<leader>tsl", _G.toggle_statusline)
```

NEW 25. Open Current File's Directory in a Split

Without using netrw explicitly.

```
function _G.open_file_directory()
  local dir = vim.fn.expand("%:p:h")
  vim.cmd("vsplit " .. dir)
end

vim.keymap.set("n", "<leader>od", _G.open_file_directory)
```

NEW 26. Sort Selected Lines (stable, case-sensitive)

Pure Lua sorting w/ visual selection or whole buffer.

```
function _G.sort_lines()
  local start_line = vim.fn.line("'<")
  local end_line   = vim.fn.line("'>")
  local lines = vim.api.nvim_buf_get_lines(0, start_line-1, end_line, false)

  table.sort(lines)
  vim.api.nvim_buf_set_lines(0, start_line-1, end_line, false, lines)
end

vim.keymap.set("v", "<leader>so", _G.sort_lines)
```

27. Swap Two Windows

Like i3wm or tmux: switch positions of two open splits.

```
function _G.swap_windows()
    local win = vim.api.nvim_get_current_win()
    vim.cmd("wincmd r")
    local new = vim.api.nvim_get_current_win()
    vim.api.nvim_set_current_win(win)
    vim.cmd("wincmd R")
    vim.api.nvim_set_current_win(new)
end

vim.keymap.set("n", "<leader>ww", _G.swap_windows, { desc = "Swap split windows" })
```

28. Maximize Split / Restore Previous Layout

A common killer productivity feature.

```
local previous_layout = nil

function _G.toggle_maximize()
    if previous_layout then
        vim.fn.winrestview(previous_layout.view)
        vim.fn.winrestcmd(previous_layout.cmd)
        previous_layout = nil
        return
    end

    previous_layout = {
        view = vim.fn.winsaveview(),
        cmd = vim.fn.winrestcmd()
    }
    vim.cmd("wincmd |")
    vim.cmd("wincmd _")
end

vim.keymap.set("n", "<leader>wm", _G.toggle_maximize)
```

29. Rename File Directly (no plugin)

Safer than writing `:saveas` manually.

```
function _G.rename_file()
    local old = vim.fn.expand("%:p")
    local new = vim.fn.input("Rename to: ", old)
```

```

if new ~= "" and new ~= old then
    vim.cmd("saveas " .. new)
    vim.fn.delete(old)
    print("Renamed:", old, "→", new)
end
end

vim.keymap.set("n", "<leader>rn", _G.rename_file)

```

30. Make a New File Beside the Current One

Smart “sibling file creation”.

```

function _G.create_sibling_file()
    local dir = vim.fn.expand("%:p:h")
    local name = vim.fn.input("New file name: ", "")
    if name ~= "" then
        vim.cmd("edit " .. dir .. "/" .. name)
    end
end

vim.keymap.set("n", "<leader>nf", _G.create_sibling_file)

```

31. Cycle Indentation Size (2 → 4 → 8 → 2...)

Helpful when switching languages.

```

local indent_cycle = {2, 4, 8}
local idx = 1

function _G.cycle_indent()
    idx = idx % #indent_cycle + 1
    local s = indent_cycle[idx]
    vim.bo.shiftwidth = s
    vim.bo.tabstop = s
    print("Indent:", s)
end

vim.keymap.set("n", "<leader>ci", _G.cycle_indent)

```

32. Show Highlight Groups at Cursor

Debug highlighting without extra plugins.

```
function _G.show_hl_group()
  local synID = vim.fn.synID(vim.fn.line("."), vim.fn.col("."), true)
  local name = vim.fn.synIDattr(synID, "name")
  print("Highlight:", name)
end

vim.keymap.set("n", "<leader>sh", _G.show_hl_group)
```

33. Quickfix: Toggle the Window

Open only if closed; close only if open.

```
function _G.toggle_qf()
  for _, win in pairs(vim.api.nvim_list_wins()) do
    if vim.fn.getwininfo(win)[1].quickfix == 1 then
      vim.cmd("cclose")
      return
    end
  end
  vim.cmd("copen")
end

vim.keymap.set("n", "<leader>qq", _G.toggle_qf)
```

34. Repeat Last Command-line Command

Like hitting @: repeatedly.

```
function _G.repeat_last_cmd()
  vim.cmd("normal! @:")
end

vim.keymap.set("n", "<leader>rr", _G.repeat_last_cmd)
```

35. Simple Scratch Buffer

Ephemeral buffer for quick notes.

```
function _G.scratch()
  vim.cmd("enew")
  vim.bo.buftype = "nofile"
  vim.bo.bufhidden = "wipe"
  vim.bo.swapfile = false
end
```

```
vim.keymap.set("n", "<leader>sc", _G.scratch)
```

Want the next batch?

I can generate more in themes such as:

- ♦ **Advanced motions**
- ♦ **Text objects (custom!)**
- ♦ **More powerful window/split tools**
- ♦ **Project/file navigation**
- ♦ **Code editing power tools**
- ♦ **LSP/treesitter helpers**
- ♦ **Git & diff utilities**
- ♦ **UI/UX polish**

Tell me which category you want to explore next!

Here is a full new set of **native Neovim-enhancing functions** — meaning: ✓ no plugins, ✓ no dependencies, ✓ pure Lua + built-ins, ✓ and each one improves or extends something Neovim already *can* do, but ergonomically.

These will boost native motions, native editing, native windowing, native marks, native registers, and buffer behavior.

Here is a **fresh batch of “plugin-style” functions** that mimic the behavior of popular Neovim plugins — **but implemented in pure Lua**, fully plug-and-play, no external dependencies.

These functions give you plugin-like power while keeping your config lightweight.

1. Mini-surround-like: Add Surrounding Characters

Emulates *adding* surroundings (like parentheses, quotes, brackets).

```
function _G.add_surround(char)
  local left = char
  local right = char

  if char == "(" then right = ")" end
  if char == "[" then right = "]" end
  if char == "{" then right = "}" end
  if char == "<" then right = ">" end
```

```

vim.cmd("normal! viw")
vim.cmd("normal! c" .. left .. vim.fn.getreg('') .. right)
end

vim.keymap.set("n", "ysa", function()
    local c = vim.fn.getcharstr()
    _G.add_surround(c)
end, { desc = "Add surround" })

```

Mimics `ysiw()` from *mini.surround* or *surround.vim*.

2. Mini.move-like: Move Line/Block Up or Down

Recreates block movement logic.

```

function _G.move_block(direction)
    local cmd = direction == "up" and "'<,'>move '<-2" or "'<,'>move '>+1"
    vim.cmd(cmd)
    vim.cmd("normal! gv")
end

vim.keymap.set("v", "J", function() _G.move_block("down") end)
vim.keymap.set("v", "K", function() _G.move_block("up") end)

```

Mimics movement from *mini.move* or *vim-move*.

3. Flash.nvim-like Jump Hint (Simple Version)

Jump to any character in view with highlights.

```

function _G.flash_jump()
    local char = vim.fn.getcharstr()
    local positions = {}

    local lines = vim.api.nvim_buf_get_lines(0, 0, -1, false)
    for i, line in ipairs(lines) do
        for col = 1, #line do
            if line:sub(col, col) == char then
                table.insert(positions, { i, col })
            end
        end
    end

    if #positions == 0 then return end

    for _, pos in ipairs(positions) do
        vim.fn.matchaddpos("Search", { pos })
    end
end

```

```

end

vim.defer_fn(function() vim.cmd("match none") end, 200)

local idx = 1
local target = positions[idx]
vim.api.nvim_win_set_cursor(0, target)
end

vim.keymap.set("n", "s", _G.flash_jump)

```

This is *extremely* lightweight compared to flash.nvim.

4. Telescope-like Quick File Picker (No Plugin)

Uses built-in fzf-like filtering and UI.

```

function _G.pick_file()
  local files = vim.fn.globpath(vim.fn.getcwd(), "**/*", false, true)
  local choice = vim.fn.inputlist(files)

  if choice > 0 and choice <= #files then
    vim.cmd("edit " .. files[choice])
  end
end

vim.keymap.set("n", "<leader>pf", _G.pick_file)

```

Not as powerful as Telescope, but the idea is plugin-like.

5. Comment.nvim-style Toggle Comment

Handles both line and visual selection.

```

function _G.toggle_comment()
  local mode = vim.fn.mode()
  local start, finish

  if mode == "v" or mode == "V" then
    start = vim.fn.line("'<")
    finish = vim.fn.line("'>")
  else
    start = vim.fn.line(".")
    finish = start
  end

  for i = start, finish do
    local line = vim.fn.getline(i)

```



```

    if line:match("^%s*//") then
        line = line:gsub("^(%s*)//", "%1")
    else
        line = "// " .. line
    end
    vim.fn.setline(i, line)
end
end

vim.keymap.set({"n", "v"}, "<leader>/", _G.toggle_comment)

```

Replicates *Comment.nvim* with basic logic.

6. Harpoon-lite: Quick File Marking System

Simple mark & jump system (like harpoon).

```

local marks = {}

function _G.mark_file()
    local path = vim.fn.expand("%:p")
    table.insert(marks, path)
    print("Marked:", path)
end

function _G.jump_mark(n)
    local file = marks[n]
    if file then vim.cmd("edit " .. file) end
end

vim.keymap.set("n", "<leader>m", _G.mark_file)
vim.keymap.set("n", "<leader>1", function() _G.jump_mark(1) end)
vim.keymap.set("n", "<leader>2", function() _G.jump_mark(2) end)
vim.keymap.set("n", "<leader>3", function() _G.jump_mark(3) end)
vim.keymap.set("n", "<leader>4", function() _G.jump_mark(4) end)

```

Plugin-like navigation without dependencies.

7. Mini.ai-like Custom Text Objects

Add a simple text-object that selects the text between quotes.

```

function _G.textobj_quotes(inner)
    local b = inner and "i" or "a"
    vim.cmd("normal! " .. b .. "'")
end

vim.keymap.set("o", "aq", function() _G.textobj_quotes(false) end)

```

```
vim.keymap.set("o", "iq", function() _G.textobj_quotes(true) end)
vim.keymap.set("x", "aq", function() _G.textobj_quotes(false) end)
vim.keymap.set("x", "iq", function() _G.textobj_quotes(true) end)
```

Mimics a small piece of *mini.ai*.

8. Autopairs-like Basic Auto-Closing

Simple autopair logic without any plugin.

```
function _G.insert_pair()
  local char = vim.fn.getcharstr()
  local pairs = { ["("]=")" , ["["]="]" , ["{"]="}" , ["'"]="' ' ' , ["'"]="' ' "}
  if pairs[char] then
    vim.api.nvim_put({ char .. pairs[char] }, "c", false, true)
    vim.cmd("normal! h")
  else
    vim.api.nvim_feedkeys(char, "i", false)
  end
end

vim.keymap.set("i", "(", _G.insert_pair)
vim.keymap.set("i", "[", _G.insert_pair)
vim.keymap.set("i", "{", _G.insert_pair)
vim.keymap.set("i", "'", _G.insert_pair)
vim.keymap.set("i", "\"", _G.insert_pair)
```

Very small, very fast.

9. Indent-blankline-like: Show Indent Ruler Temporarily

Show indentation guides for 1 second.

```
function _G.flash_indent_guides()
  vim.cmd("set listchars=tab:|\ ,space:·")
  vim.cmd("set list")
  vim.defer_fn(function() vim.cmd("set nolist") end, 1000)
end

vim.keymap.set("n", "<leader>ig", _G.flash_indent_guides)
```

Not persistent, but plugin-like.

10. Zen Mode-like Minimal UI Toggle

Emulates *folke/zen-mode.nvim*.

```
local zen_active = false

function _G.toggle_zen()
  zen_active = not zen_active

  if zen_active then
    vim.o.laststatus = 0
    vim.o.number = false
    vim.o.relativenumber = false
    vim.o.signcolumn = "no"
    vim.cmd("set colorcolumn=")
  else
    vim.o.laststatus = 3
    vim.o.number = true
    vim.o.relativenumber = true
    vim.o.signcolumn = "yes"
  end
end

vim.keymap.set("n", "<leader>zz", _G.toggle_zen)
```

Lightweight alternative to Zen Mode.

Want more plugin-inspired custom functions?

I can build pure-Lua equivalents for:

- **gitsigns-lite**
- **leap-lite** or **hop-lite**
- **treesitter-like structural movement**
- **align.nvim** behavior
- **oil.nvim-like file explorer**
- **snippets engine (Lua-side)**
- **mini.animate / smooth scrolling**
- **bufferline / tabline helpers**
- **noice.nvim-lite (better messages)**

Which plugin functionality should I re-implement next?
