with-qwen

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

---

I'll help you create several custom Lua functions for Neovim that use `vim.cmd` to execute Vimscript commands for enhanced coding experience. Here are some useful functions:

```lua
-- Custom utility functions for enhanced coding experience
local M = {}

-- Toggle between relative and absolute line numbers
M.toggle_number = function()
    if vim.wo.number and vim.wo.relativenumber then
        vim.wo.number = false
        vim.wo.relativenumber = false
        print("Line numbers: off")
    elseif not vim.wo.number and not vim.wo.relativenumber then
        vim.wo.number = true
        vim.wo.relativenumber = true
        print("Line numbers: relative")
    else
        vim.wo.number = true
        vim.wo.relativenumber = false
        print("Line numbers: absolute")
    end
end

-- Toggle spell checking
M.toggle_spell = function()
    if vim.wo.spell then
        vim.wo.spell = false
        print("Spell check: off")
    else
        vim.wo.spell = true
        print("Spell check: on")
    end
end

-- Toggle cursor line highlighting
M.toggle_cursorline = function()
    if vim.wo.cursorline then
        vim.wo.cursorline = false
        print("Cursor line: off")
    else
        vim.wo.cursorline = true
        print("Cursor line: on")
```

```
    end
end

-- Toggle color column
M.toggle_colorcolumn = function()
    if vim.wo.colorcolumn ~= "" then
        vim.wo.colorcolumn = ""
        print("Color column: off")
    else
        vim.wo.colorcolumn = "80"
        print("Color column: 80")
    end
end

-- Quick save and quit
M.save_quit = function()
    vim.cmd("write")
    vim.cmd("quit")
end

-- Quick save all and quit all
M.save_quit_all = function()
    vim.cmd("wall") -- write all
    vim.cmd("qall") -- quit all
end

-- Toggle paste mode (useful for pasting code without auto-indentation)
M.toggle_paste = function()
    if vim.o.paste then
        vim.o.paste = false
        print("Paste mode: off")
    else
        vim.o.paste = true
        print("Paste mode: on")
    end
end

-- Clear search highlighting
M.clear_highlight = function()
    vim.cmd("nohlsearch")
end

-- Toggle search highlighting
M.toggle_search_highlight = function()
    if vim.v.hlsearch == 1 then
        vim.cmd("set nohlsearch")
        print("Search highlight: off")
    else
        vim.cmd("set hlsearch")
        print("Search highlight: on")
    end
end
```

```lua
-- Format the current buffer using built-in formatter
M.format_buffer = function()
    vim.cmd("normal! gg=G")
    print("Buffer formatted")
end

-- Toggle wrap mode
M.toggle_wrap = function()
    if vim.wo.wrap then
        vim.wo.wrap = false
        print("Wrap: off")
    else
        vim.wo.wrap = true
        print("Wrap: on")
    end
end

-- Toggle conceal level (useful for markdown and other syntax)
M.toggle_conceal = function()
    if vim.o.conceallevel == 0 then
        vim.o.conceallevel = 2
        print("Conceal: on")
    else
        vim.o.conceallevel = 0
        print("Conceal: off")
    end
end

-- Open file explorer (using netrw)
M.open_file_explorer = function()
    vim.cmd("E")
end

-- Toggle between different indent sizes
M.toggle_indent = function()
    local current_indent = vim.o.shiftwidth
    if current_indent == 2 then
        vim.o.shiftwidth = 4
        vim.o.tabstop = 4
        vim.o.softtabstop = 4
        print("Indent: 4 spaces")
    elseif current_indent == 4 then
        vim.o.shiftwidth = 2
        vim.o.tabstop = 2
        vim.o.softtabstop = 2
        print("Indent: 2 spaces")
    else
        vim.o.shiftwidth = 2
        vim.o.tabstop = 2
        vim.o.softtabstop = 2
        print("Indent: 2 spaces")
```

```lua
    end
end

-- Center current line in middle of screen
M.center_line = function()
    vim.cmd("normal! zz")
end

-- Move to middle of screen
M.middle_screen = function()
    vim.cmd("normal! z.")
end

-- Move to bottom of screen
M.bottom_screen = function()
    vim.cmd("normal! z-")
end

-- Toggle termguicolors
M.toggle_termguicolors = function()
    if vim.o.termguicolors then
        vim.o.termguicolors = false
        print("Term colors: 256")
    else
        vim.o.termguicolors = true
        print("Term colors: truecolor")
    end
end

-- Reload current file
M.reload_file = function()
    vim.cmd("edit")
    print("File reloaded")
end

-- Open new tab
M.new_tab = function()
    vim.cmd("tabnew")
end

-- Close current tab
M.close_tab = function()
    vim.cmd("tabclose")
end

-- Move to next tab
M.next_tab = function()
    vim.cmd("tabnext")
end

-- Move to previous tab
M.prev_tab = function()
```

```lua
    vim.cmd("tabprev")
end

-- Split window horizontally
M.split_horizontal = function()
    vim.cmd("split")
end

-- Split window vertically
M.split_vertical = function()
    vim.cmd("vsplit")
end

-- Close all other windows
M.close_other_windows = function()
    vim.cmd("only")
end

-- Toggle terminal in a split
M.toggle_terminal = function()
    if vim.fn.exists("t:terminal_job_id") == 0 then
        vim.cmd("terminal")
        vim.cmd("startinsert")
    else
        vim.cmd("bdelete!")
    end
end

-- Quick comment toggle using built-in method
M.toggle_comment = function()
    local line = vim.fn.line(".")
    local col = vim.fn.col(".")
    local line_text = vim.fn.getline(".")

    if string.match(line_text, "^%s*[^%s]") then
        local indent = string.match(line_text, "^(%s*)")
        if string.match(line_text, "^%s*//") or string.match(line_text, "^%s*#") or
string.match(line_text, "^%s;/*") then
            -- Uncomment
            if string.match(line_text, "^%s*//") then
                vim.cmd("s/\\/\\///")
            elseif string.match(line_text, "^%s*#") then
                vim.cmd("s/#//")
            elseif string.match(line_text, "^%s;/*") then
                vim.cmd("s/;/ /")
            end
        else
            -- Comment
            if vim.bo.filetype == "lua" or vim.bo.filetype == "javascript" or
vim.bo.filetype == "typescript" or vim.bo.filetype == "python" then
                vim.cmd("normal! ^i-- ")
            else
```

```
                vim.cmd("normal! ^i// ")
            end
        end
    end
end

-- Set up key mappings for these functions
M.setup_keymaps = function()
    -- Create a new keymap function for convenience
    local set_keymap = vim.api.nvim_set_keymap

    -- Toggle functions
    set_keymap('n', '<leader>tn', '<cmd>lua
require("user.functions").toggle_number()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>ts', '<cmd>lua require("user.functions").toggle_spell()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tc', '<cmd>lua
require("user.functions").toggle_cursorline()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>tw', '<cmd>lua require("user.functions").toggle_wrap()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tp', '<cmd>lua require("user.functions").toggle_paste()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>th', '<cmd>lua
require("user.functions").toggle_search_highlight()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>tt', '<cmd>lua
require("user.functions").toggle_colorcolumn()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>tf', '<cmd>lua
require("user.functions").format_buffer()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>ti', '<cmd>lua
require("user.functions").toggle_indent()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tr', '<cmd>lua
require("user.functions").toggle_conceal()<CR>', { noremap = true, silent = true })

    -- Navigation and window management
    set_keymap('n', '<C-s>', '<cmd>lua require("user.functions").center_line()<CR>',
{ noremap = true, silent = true })
    set_keymap('n', '<leader>re', '<cmd>lua require("user.functions").reload_file()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>q', '<cmd>lua require("user.functions").save_quit()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>Q', '<cmd>lua require("user.functions").save_quit_all()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>ch', '<cmd>lua
require("user.functions").clear_highlight()<CR>', { noremap = true, silent = true })

    -- Window splitting
    set_keymap('n', '<leader>sh', '<cmd>lua
require("user.functions").split_horizontal()<CR>', { noremap = true, silent = true
})
```

```
    set_keymap('n', '<leader>sv', '<cmd>lua
require("user.functions").split_vertical()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>so', '<cmd>lua
require("user.functions").close_other_windows()<CR>', { noremap = true, silent =
true })

    -- Tab management
    set_keymap('n', '<leader>tn', '<cmd>lua require("user.functions").new_tab()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tc', '<cmd>lua require("user.functions").close_tab()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tt', '<cmd>lua require("user.functions").next_tab()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tT', '<cmd>lua require("user.functions").prev_tab()
<CR>', { noremap = true, silent = true })

    -- Terminal toggle
    set_keymap('n', '<C-\\>', '<cmd>lua require("user.functions").toggle_terminal()
<CR>', { noremap = true, silent = true })
end

return M
```

To use these functions, you'll need to:

1. Save this code in a file like `~/.config/nvim/lua/user/functions.lua`
2. Call the setup function to create keymaps:

```
-- In your init.lua
require('user.functions').setup_keymaps()
```

Or you can source it in your `init.vim`:

```
lua require('user.functions').setup_keymaps()
```

Key Features:

1. **Toggle Functions**: For common settings like line numbers, spell check, cursor line, etc.
2. **Window Management**: Split windows, close others, tab management
3. **Navigation**: Center line, screen positioning
4. **File Operations**: Quick save/quit, reload file
5. **Code Formatting**: Buffer formatting, comment toggling
6. **Terminal Integration**: Toggle terminal in split

Default Key Mappings:

- `<leader>tn` - Toggle line numbers
- `<leader>ts` - Toggle spell check
- `<leader>tc` - Toggle cursor line
- `<leader>tw` - Toggle word wrap

- `<leader>tp` - Toggle paste mode
- `<leader>th` - Toggle search highlight
- `<leader>tt` - Toggle color column
- `<leader>tf` - Format buffer
- `<leader>ti` - Toggle indent size
- `<leader>re` - Reload file
- `<leader>q` - Save and quit
- `<leader>Q` - Save all and quit all
- `<leader>ch` - Clear highlight
- `<C-\>` - Toggle terminal

You can customize these functions or key mappings according to your preferences!

Here are more advanced custom Lua functions with Vimscript integration for enhanced coding experience:

```lua
-- Advanced custom functions with Vimscript integration
local M = {}

-- Function to create a scratch buffer for temporary notes/coding
M.scratch_buffer = function()
    vim.cmd("enew")
    vim.cmd("setlocal buftype=nofile bufhidden=hide noswapfile")
    vim.bo.filetype = "text"
    print("Scratch buffer created")
end

-- Function to duplicate current line
M.duplicate_line = function()
    vim.cmd("t.")
end

-- Function to duplicate current line and move cursor to new line
M.duplicate_line_below = function()
    vim.cmd("t . | +")
end

-- Function to duplicate current line above
M.duplicate_line_above = function()
    vim.cmd("t . | -")
end

-- Function to move current line up/down
M.move_line_up = function()
    vim.cmd("move -2")
    vim.cmd("normal! ==")
end

M.move_line_down = function()
    vim.cmd("move +1")
    vim.cmd("normal! ==")
end
```

```lua
-- Function to join lines (like J but more aggressive)
M.join_lines = function(count)
    count = count or 2
    vim.cmd("normal! " .. count .. "J")
end

-- Function to select the last pasted text
M.select_last_paste = function()
    vim.cmd("normal! `[]v`]")
end

-- Function to align text at cursor position
M.align_text = function()
    local pos = vim.api.nvim_win_get_cursor(0)
    local line = vim.api.nvim_buf_get_lines(0, pos[1] - 1, pos[1], false)[1]
    local col = pos[2]

    -- Go to the end of the line and align
    vim.cmd("normal! $")
    local end_col = vim.api.nvim_win_get_cursor(0)[2]

    if end_col > col then
        vim.cmd("normal! " .. (col + 1) .. "|")
        vim.cmd("normal! i<C-V>")
        vim.cmd("normal! " .. (end_col - col) .. "l")
        vim.cmd("normal! A<Esc>")
    end
end

-- Function to comment/uncomment visual selection
M.visual_comment = function()
    local mode = vim.api.nvim_get_mode().mode
    if mode == "v" or mode == "V" or mode == "□" then
        local comment_char = "#"
        if vim.bo.filetype == "javascript" or vim.bo.filetype == "typescript" or
vim.bo.filetype == "lua" then
            comment_char = "--"
        elseif vim.bo.filetype == "python" or vim.bo.filetype == "yaml" then
            comment_char = "#"
        else
            comment_char = "//"
        end
        vim.cmd("normal! gI" .. comment_char .. "<Esc>")
    end
end

-- Function to find and replace in current buffer
M.find_replace_buffer = function()
    local old_text = vim.fn.input("Find: ")
    if old_text ~= "" then
        local new_text = vim.fn.input("Replace with: ")
```

```
        vim.cmd(":%s/" .. old_text .. "/" .. new_text .. "/gc")
    end
end

-- Function to find and replace in visual selection
M.find_replace_visual = function()
    local old_text = vim.fn.input("Find: ")
    if old_text ~= "" then
        local new_text = vim.fn.input("Replace with: ")
        vim.cmd(":'<,'>s/" .. old_text .. "/" .. new_text .. "/gc")
    end
end

-- Function to sort lines in visual selection
M.sort_visual = function()
    vim.cmd(":'<,'>sort")
end

-- Function to reverse lines in visual selection
M.reverse_visual = function()
    vim.cmd(":'<,'>g/^/m'<-1")
end

-- Function to convert case (uppercase, lowercase, capitalize)
M.toggle_case = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "□" then
        local choice = vim.fn.input("Convert to (u=upper, l=lower, c=capitalize,
t=toggle): ")
        if choice == "u" then
            vim.cmd("normal! U")
        elseif choice == "l" then
            vim.cmd("normal! u")
        elseif choice == "c" then
            vim.cmd("normal! gU")
        elseif choice == "t" then
            vim.cmd("normal! ~")
        end
    else
        print("Please select text in visual mode first")
    end
end

-- Function to trim whitespace
M.trim_whitespace = function()
    vim.cmd([[keeppatterns %s/\s\+$//e]])
    vim.cmd([[keeppatterns %s/\n\+\%$//e]])
end

-- Function to trim trailing whitespace only
M.trim_trailing_whitespace = function()
    vim.cmd([[keeppatterns %s/\s\+$//e]])
```

```lua
end

-- Function to duplicate current word
M.duplicate_word = function()
    vim.cmd("normal! byeWp")
end

-- Function to duplicate current word to next line
M.duplicate_word_next_line = function()
    vim.cmd("normal! byeWj^p")
end

-- Function to jump to matching bracket
M.jump_to_match = function()
    vim.cmd("normal! %")
end

-- Function to insert current date/time
M.insert_datetime = function()
    local date = os.date("%Y-%m-%d %H:%M:%S")
    vim.cmd("normal! a" .. date)
end

-- Function to insert current date
M.insert_date = function()
    local date = os.date("%Y-%m-%d")
    vim.cmd("normal! a" .. date)
end

-- Function to insert current time
M.insert_time = function()
    local time = os.date("%H:%M:%S")
    vim.cmd("normal! a" .. time)
end

-- Function to encode/decode URL
M.url_encode = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "□" then
        vim.cmd([[
            let old_reg = @"
            normal! gvy
            let encoded = substitute(@", '%', '%25', 'g')
            let encoded = substitute(encoded, ' ', '%20', 'g')
            let encoded = substitute(encoded, '!', '%21', 'g')
            let encoded = substitute(encoded, '#', '%23', 'g')
            let encoded = substitute(encoded, '\$', '%24', 'g')
            let encoded = substitute(encoded, '&', '%26', 'g')
            let encoded = substitute(encoded, ''', '%27', 'g')
            let encoded = substitute(encoded, '(', '%28', 'g')
            let encoded = substitute(encoded, ')', '%29', 'g')
            let encoded = substitute(encoded, '*', '%2A', 'g')
```

```
            let encoded = substitute(encoded, '+', '%2B', 'g')
            let encoded = substitute(encoded, ',', '%2C', 'g')
            let encoded = substitute(encoded, '/', '%2F', 'g')
            let encoded = substitute(encoded, ':', '%3A', 'g')
            let encoded = substitute(encoded, ';', '%3B', 'g')
            let encoded = substitute(encoded, '=', '%3D', 'g')
            let encoded = substitute(encoded, '?', '%3F', 'g')
            let encoded = substitute(encoded, '@', '%40', 'g')
            let encoded = substitute(encoded, '[', '%5B', 'g')
            let encoded = substitute(encoded, ']', '%5D', 'g')
            let @" = encoded
            normal! gvp
            let @" = old_reg
        ]])
    end
end

-- Function to run current file based on filetype
M.run_file = function()
    local file_path = vim.fn.expand("%:p")
    local file_type = vim.bo.filetype

    if file_type == "python" then
        vim.cmd("!python3 " .. file_path)
    elseif file_type == "javascript" or file_type == "js" then
        vim.cmd("!node " .. file_path)
    elseif file_type == "lua" then
        vim.cmd("!lua " .. file_path)
    elseif file_type == "sh" or file_type == "bash" then
        vim.cmd("!bash " .. file_path)
    elseif file_type == "go" then
        vim.cmd("!go run " .. file_path)
    elseif file_type == "rust" then
        vim.cmd("!cargo run")
    elseif file_type == "c" then
        vim.cmd("!gcc " .. file_path .. " -o output && ./output")
    elseif file_type == "cpp" then
        vim.cmd("!g++ " .. file_path .. " -o output && ./output")
    else
        print("No run command defined for " .. file_type)
    end
end

-- Function to compile current file
M.compile_file = function()
    local file_path = vim.fn.expand("%:p")
    local file_type = vim.bo.filetype

    if file_type == "c" then
        vim.cmd("!gcc " .. file_path .. " -o " .. vim.fn.expand("%:r"))
    elseif file_type == "cpp" then
        vim.cmd("!g++ " .. file_path .. " -o " .. vim.fn.expand("%:r"))
```

```lua
    elseif file_type == "go" then
        vim.cmd("!go build " .. file_path)
    else
        print("No compile command defined for " .. file_type)
    end
end

-- Function to search for word under cursor in current file
M.search_word_in_file = function()
    local word = vim.fn.expand("<cword>")
    vim.cmd("/\\<" .. word .. "\\>")
end

-- Function to search for word under cursor in project (using grep)
M.search_word_in_project = function()
    local word = vim.fn.expand("<cword>")
    vim.cmd("!grep -r '" .. word .. "' .")
end

-- Function to toggle between different number formats (decimal, hex, binary)
M.cycle_number_format = function()
    local line = vim.fn.getline(".")
    local col = vim.fn.col(".")

    -- Find number at cursor position
    local start_col = col
    while start_col > 1 and string.match(string.sub(line, start_col - 1, start_col -
1), "[0-9a-fA-Fx]") do
        start_col = start_col - 1
    end

    local end_col = col
    while end_col <= #line and string.match(string.sub(line, end_col, end_col), "[0-
9a-fA-Fx]") do
        end_col = end_col + 1
    end

    local num_str = string.sub(line, start_col, end_col - 1)
    local num

    if string.match(num_str, "^0x") then
        num = tonumber(num_str, 16)
    elseif string.match(num_str, "^0b") then
        num = tonumber(string.gsub(num_str, "^0b", ""), 2)
    else
        num = tonumber(num_str)
    end

    if num then
        -- Cycle: decimal -> hex -> binary -> decimal
        local new_num
        if num_str:match("^%d+$") then
```

```
            new_num = string.format("0x%x", num)
        elseif num_str:match("^0x") then
            new_num = string.format("0b%b", num)
        else
            new_num = tostring(num)
        end

        vim.fn.setline(".", string.sub(line, 1, start_col - 1) .. new_num ..
string.sub(line, end_col))
        vim.fn.cursor(0, start_col + #new_num)
    end
end

-- Function to create a new file in same directory as current file
M.new_file_same_dir = function()
    local current_dir = vim.fn.expand("%:p:h")
    local new_file = vim.fn.input("New file name: ", current_dir .. "/", "file")
    if new_file ~= "" then
        vim.cmd("edit " .. new_file)
    end
end

-- Function to open file under cursor
M.open_file_under_cursor = function()
    local file_name = vim.fn.expand("<cfile>")
    if file_name ~= "" then
        vim.cmd("edit " .. file_name)
    end
end

-- Function to create backup of current file
M.backup_file = function()
    local current_file = vim.fn.expand("%:p")
    local backup_file = current_file .. ".bak"
    vim.cmd("!cp '" .. current_file .. "' '" .. backup_file .. "'")
    print("Backup created: " .. backup_file)
end

-- Function to toggle between different color schemes
M.cycle_colorscheme = function()
    local schemes = {"default", "desert", "molokai", "onedark", "gruvbox"}
    local current = vim.o.background
    local next_scheme = schemes[1]

    for i, scheme in ipairs(schemes) do
        if vim.g.colors_name == scheme then
            next_scheme = schemes[(i % #schemes) + 1]
            break
        end
    end

    vim.cmd("colorscheme " .. next_scheme)
```

```lua
end

-- Function to increase/decrease numbers in visual selection
M.increase_number = function(amount)
    amount = amount or 1
    vim.cmd("normal! " .. amount .. "<C-a>")
end

M.decrease_number = function(amount)
    amount = amount or 1
    vim.cmd("normal! " .. amount .. "<C-x>")
end

-- Function to create a quick fix list from search results
M.search_to_qflist = function()
    local search_term = vim.fn.input("Search term: ")
    if search_term ~= "" then
        vim.cmd("grep! " .. search_term)
    end
end

-- Function to toggle auto-indent
M.toggle_autoindent = function()
    if vim.o.autoindent and vim.o.smartindent then
        vim.o.autoindent = false
        vim.o.smartindent = false
        vim.o.cindent = false
        print("Auto-indent: off")
    else
        vim.o.autoindent = true
        vim.o.smartindent = true
        vim.o.cindent = true
        print("Auto-indent: on")
    end
end

-- Function to toggle sign column
M.toggle_signcolumn = function()
    if vim.wo.signcolumn == "yes" then
        vim.wo.signcolumn = "no"
        print("Sign column: off")
    else
        vim.wo.signcolumn = "yes"
        print("Sign column: on")
    end
end

-- Function to toggle virtual text (useful for LSP virtual text)
M.toggle_virtual_text = function()
    if vim.wo.conceallevel == 2 then
        vim.wo.conceallevel = 0
        print("Virtual text: off")
```

```
    else
        vim.wo.conceallevel = 2
        print("Virtual text: on")
    end
end

-- Function to insert empty line above/below
M.insert_empty_line_above = function()
    vim.cmd("normal! O<Esc>")
end

M.insert_empty_line_below = function()
    vim.cmd("normal! o<Esc>")
end

-- Setup key mappings for all functions
M.setup_advanced_keymaps = function()
    local set_keymap = vim.api.nvim_set_keymap

    -- Scratch and utility
    set_keymap('n', '<leader>ns', '<cmd>lua
require("user.functions").scratch_buffer()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>dd', '<cmd>lua
require("user.functions").duplicate_line()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>dj', '<cmd>lua
require("user.functions").duplicate_line_below()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>dk', '<cmd>lua
require("user.functions").duplicate_line_above()<CR>', { noremap = true, silent =
true })

    -- Line movement
    set_keymap('n', '<A-j>', '<cmd>lua require("user.functions").move_line_down()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<A-k>', '<cmd>lua require("user.functions").move_line_up()
<CR>', { noremap = true, silent = true })

    -- Text manipulation
    set_keymap('n', '<leader>j', '<cmd>lua require("user.functions").join_lines(2)
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>J', '<cmd>lua require("user.functions").join_lines(3)
<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>c', '<cmd>lua
require("user.functions").visual_comment()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>fr', '<cmd>lua
require("user.functions").find_replace_buffer()<CR>', { noremap = true, silent =
true })
    set_keymap('v', '<leader>fr', '<cmd>lua
require("user.functions").find_replace_visual()<CR>', { noremap = true, silent =
true })
    set_keymap('v', '<leader>s', '<cmd>lua require("user.functions").sort_visual()
<CR>', { noremap = true, silent = true })
```

```
    set_keymap('v', '<leader>r', '<cmd>lua
require("user.functions").reverse_visual()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>C', '<cmd>lua require("user.functions").toggle_case()
<CR>', { noremap = true, silent = true })

    -- Whitespace and formatting
    set_keymap('n', '<leader>tw', '<cmd>lua
require("user.functions").trim_whitespace()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>tr', '<cmd>lua
require("user.functions").trim_trailing_whitespace()<CR>', { noremap = true, silent
= true })

    -- Date/time insertion
    set_keymap('n', '<leader>id', '<cmd>lua require("user.functions").insert_date()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>it', '<cmd>lua require("user.functions").insert_time()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>idt', '<cmd>lua
require("user.functions").insert_datetime()<CR>', { noremap = true, silent = true })

    -- File operations
    set_keymap('n', '<leader>rf', '<cmd>lua require("user.functions").run_file()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>cf', '<cmd>lua require("user.functions").compile_file()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>nd', '<cmd>lua
require("user.functions").new_file_same_dir()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<C-w>f', '<cmd>lua
require("user.functions").open_file_under_cursor()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>bk', '<cmd>lua require("user.functions").backup_file()
<CR>', { noremap = true, silent = true })

    -- Search functions
    set_keymap('n', '<leader>sw', '<cmd>lua
require("user.functions").search_word_in_file()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>sp', '<cmd>lua
require("user.functions").search_word_in_project()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>sg', '<cmd>lua
require("user.functions").search_to_qflist()<CR>', { noremap = true, silent = true
})

    -- Number manipulation
    set_keymap('v', '<C-a>', '<cmd>lua require("user.functions").increase_number(1)
<CR>', { noremap = true, silent = true })
    set_keymap('v', '<C-x>', '<cmd>lua require("user.functions").decrease_number(1)
<CR>', { noremap = true, silent = true })

    -- Additional toggles
```

```
    set_keymap('n', '<leader>ta', '<cmd>lua
require("user.functions").toggle_autoindent()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>ts', '<cmd>lua
require("user.functions").toggle_signcolumn()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>tv', '<cmd>lua
require("user.functions").toggle_virtual_text()<CR>', { noremap = true, silent =
true })

    -- Empty line insertion
    set_keymap('n', '<leader>o', '<cmd>lua
require("user.functions").insert_empty_line_below()<CR>', { noremap = true, silent =
true })
    set_keymap('n', '<leader>O', '<cmd>lua
require("user.functions").insert_empty_line_above()<CR>', { noremap = true, silent =
true })
end

return M
```

These advanced functions provide:

Text Manipulation:

- Line duplication and movement
- Text alignment and case conversion
- Find and replace operations
- Sorting and reversing text

File Operations:

- Run/compile files based on type
- Create backups
- New file in same directory
- Open file under cursor

Utility Functions:

- Date/time insertion
- URL encoding
- Number format cycling
- Quick fix list creation

Advanced Vimscript Integration:

- Complex regex operations
- Register manipulation
- Visual mode operations
- Mode-aware functions

Key Features:

- Visual mode aware functions
- Context-aware operations
- Filetype-specific behaviors
- Comprehensive key mappings

To use these functions, save them in `~/.config/nvim/lua/user/advanced_functions.lua` and call:

```
require('user.advanced_functions').setup_advanced_keymaps()
```

These functions combine Lua's power with Vimscript's extensive text manipulation capabilities for a superior coding experience!

Here are additional innovative custom functions that go beyond the previous ones, focusing on unique coding enhancements and workflow optimizations:

```
-- Additional advanced utility functions with Vimscript integration
local M = {}

-- Function to create a code snippet template based on filetype
M.create_snippet_template = function()
    local ft = vim.bo.filetype
    local template = ""

    if ft == "python" then
        template = [[
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Module: ${1:module_name}
Author: ${2:author}
Date: ]] .. os.date("%Y-%m-%d") .. [[

Description:
${3:description}
"""

import sys
import logging

def main():
    """Main function."""
    pass

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)
    main()
]]
    elseif ft == "javascript" or ft == "typescript" then
        template = [[
/**
```

```
 * File: ${1:filename}
 * Author: ${2:author}
 * Date: ]] .. os.date("%Y-%m-%d") .. [[
 *
 * Description:
 * ${3:description}
 */

class ${4:ClassName} {
    constructor() {
        // initialization
    }
}

export default ${4:ClassName};
]]
    elseif ft == "lua" then
        template = [[
-- Module: ${1:module_name}
-- Author: ${2:author}
-- Date: ]] .. os.date("%Y-%m-%d") .. [[
--
-- Description:
-- ${3:description}

local M = {}

-- Your code here

return M
]]
    elseif ft == "go" then
        template = [[
package main

import (
    "fmt"
    "log"
)

func main() {
    fmt.Println("Hello, World!")
}
]]
    else
        template = "-- Auto-generated template\n"
    end

    -- Clear current buffer and insert template
    vim.cmd("silent! %d")
    for line in string.gmatch(template, "[^\n]+") do
        vim.cmd("normal! o" .. line)
```

```lua
    end
    vim.cmd("normal! gg")
end

-- Function to generate function documentation (docstring)
M.generate_docstring = function()
    local ft = vim.bo.filetype
    local line = vim.fn.getline(".")
    local func_name = string.match(line, "function%s+(%w+)")

    if not func_name then
        func_name = string.match(line, "def%s+(%w+)")
    end

    if not func_name then
        print("No function definition found")
        return
    end

    local docstring = ""

    if ft == "python" then
        docstring = [[
    """
    ${1:brief_description}

    Args:
        ${2:arg_name} (${3:arg_type}): ${4:arg_description}

    Returns:
        ${5:return_type}: ${6:return_description}

    Raises:
        ${7:ExceptionType}: ${8:exception_description}
    """
]]
    elseif ft == "javascript" or ft == "typescript" then
        docstring = [[
    /**
     * ${1:brief_description}
     *
     * @param {${2:param_type}} ${3:param_name} - ${4:param_description}
     * @returns {${5:return_type}} ${6:return_description}
     * @throws {${7:error_type}} ${8:error_description}
     */
]]
    elseif ft == "lua" then
        docstring = [[
    --- ${1:brief_description}
    -- @param ${2:param_name} ${3:param_type} ${4:param_description}
    -- @return ${5:return_type} ${6:return_description}
    -- @raise ${7:error_type} ${8:error_description}
```

```
]]
    else
        docstring = "--[[\n${1:documentation}\n]]"
    end

    vim.cmd("normal! o" .. string.gsub(docstring, "\n", "\no"))
    vim.cmd("normal! kkk")
end

-- Function to extract variable from selected text
M.extract_variable = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "" then
        -- Yank the visual selection
        vim.cmd("normal! y")
        local selected_text = vim.fn.getreg('"')

        -- Generate variable name from selected text
        local var_name = string.gsub(selected_text, "[^%w_]", "_")
        var_name = string.gsub(var_name, "^_+", "")
        var_name = string.gsub(var_name, "_+$", "")
        var_name = string.lower(var_name)

        -- Insert variable declaration before current line
        vim.cmd("normal! Olocal " .. var_name .. " = \"" .. selected_text .. "\"
<Esc>")

        -- Replace selection with variable name
        vim.cmd("normal! gv\"_c" .. var_name .. "<Esc>")
    end
end

-- Function to wrap selected text with HTML tags
M.wrap_html_tag = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "" then
        local tag = vim.fn.input("HTML Tag: ", "div")
        vim.cmd("normal! S<" .. tag .. ">\"_p</" .. tag .. "><Esc>")
    end
end

-- Function to wrap selected text with markdown formatting
M.wrap_markdown = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "" then
        local format = vim.fn.input("Format (b=bold, i=italic, c=code, l=link): ")
        local wrapper = ""

        if format == "b" then
            wrapper = "**"
        elseif format == "i" then
            wrapper = "*"
```

```
        elseif format == "c" then
            wrapper = "`"
        elseif format == "l" then
            local url = vim.fn.input("URL: ")
            wrapper = "](" .. url .. ")"
            vim.cmd("normal! gvy")
            local text = vim.fn.getreg('"')
            vim.fn.setreg('"', "[" .. text)
            vim.cmd("normal! \"_c[" .. text .. "](" .. url .. ")<Esc>")
            return
        else
            print("Invalid format")
            return
        end

        vim.cmd("normal! S" .. wrapper .. "\"_p" .. wrapper .. "<Esc>")
    end
end

-- Function to convert JSON to YAML (requires Python)
M.json_to_yaml = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "" then
        vim.cmd([[
            let save_reg = @"
            normal! gvy
            let json_text = @"
            let yaml_text = system('python3 -c "import sys, yaml, json;
print(yaml.dump(json.loads(sys.stdin.read()), default_flow_style=False))"',
json_text)
            if v:shell_error == 0
                let @" = yaml_text
                normal! gvp
            else
                echo "Conversion failed"
            endif
            let @" = save_reg
        ]])
    end
end

-- Function to convert YAML to JSON (requires Python)
M.yaml_to_json = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "" then
        vim.cmd([[
            let save_reg = @"
            normal! gvy
            let yaml_text = @"
            let json_text = system('python3 -c "import sys, yaml, json;
print(json.dumps(yaml.safe_load(sys.stdin.read()), indent=2))"', yaml_text)
            if v:shell_error == 0
```

```
                let @" = json_text
                normal! gvp
            else
                echo "Conversion failed"
            endif
            let @" = save_reg
        ]])
    end
end

-- Function to generate UUID
M.generate_uuid = function()
    vim.cmd([[
        let uuid = system('python3 -c "import uuid; print(uuid.uuid4())"')
        let uuid = substitute(uuid, '\n$', '', '')
        normal! a<C-r>=uuid<CR>
    ]])
end

-- Function to encode/decode Base64
M.base64_encode = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "□" then
        vim.cmd([[
            let save_reg = @"
            normal! gvy
            let text = @"
            let encoded = system('echo -n "' . text . '" | base64')
            let encoded = substitute(encoded, '\n$', '', '')
            let @" = encoded
            normal! gvp
            let @" = save_reg
        ]])
    end
end

M.base64_decode = function()
    local mode = vim.fn.mode()
    if mode == "v" or mode == "V" or mode == "□" then
        vim.cmd([[
            let save_reg = @"
            normal! gvy
            let text = @"
            let decoded = system('echo "' . text . '" | base64 -d')
            let @" = decoded
            normal! gvp
            let @" = save_reg
        ]])
    end
end

-- Function to create a table of contents for markdown files
```

```lua
M.create_toc = function()
    if vim.bo.filetype ~= "markdown" then
        print("Only works with markdown files")
        return
    end

    local lines = vim.api.nvim_buf_get_lines(0, 0, -1, false)
    local toc = {}
    local in_code_block = false

    table.insert(toc, "# Table of Contents")
    table.insert(toc, "")

    for _, line in ipairs(lines) do
        if string.match(line, "^```") then
            in_code_block = not in_code_block
        end

        if not in_code_block and string.match(line, "^#{1,6}%s") then
            local level = select(2, string.find(line, "^#+"))
            local content = string.gsub(line, "^#+%s*", "")
            local link = string.gsub(string.lower(content), "[^%w%-]", "-")

            table.insert(toc, string.rep("  ", level - 1) .. "- [" .. content .. "]
(#" .. link .. ")")
        end
    end

    -- Insert TOC at the beginning
    vim.api.nvim_buf_set_lines(0, 0, 0, false, toc)
end

-- Function to jump between matching HTML/XML tags
M.jump_to_matching_tag = function()
    vim.cmd("normal! vip")
    vim.cmd("normal! f<")
end

-- Function to toggle between light and dark theme
M.toggle_theme = function()
    local current_bg = vim.o.background
    if current_bg == "dark" then
        vim.o.background = "light"
        vim.cmd("colorscheme default")
    else
        vim.o.background = "dark"
        vim.cmd("colorscheme default")
    end
    print("Theme: " .. vim.o.background)
end

-- Function to create a quick benchmark/timer
```

```lua
M.start_timer = function()
    M.timer_start = vim.loop.now()
    print("Timer started")
end

M.stop_timer = function()
    if M.timer_start then
        local elapsed = (vim.loop.now() - M.timer_start) / 1000
        print(string.format("Time elapsed: %.3f seconds", elapsed))
        M.timer_start = nil
    else
        print("No timer running")
    end
end

-- Function to measure code execution time (for supported languages)
M.benchmark_code = function()
    local ft = vim.bo.filetype
    local file_path = vim.fn.expand("%:p")

    if ft == "python" then
        vim.cmd("!python3 -m timeit -s 'from " .. vim.fn.expand("%:t:r") .. " import
*' 'main()'")
    elseif ft == "javascript" or ft == "js" then
        vim.cmd("!node -e \"console.time('Execution'); require('./" ..
vim.fn.expand("%:t:r") .. "'); console.timeEnd('Execution');\"")
    elseif ft == "lua" then
        vim.cmd("!time lua " .. file_path)
    else
        print("Benchmarking not supported for this language")
    end
end

-- Function to generate random password
M.generate_password = function(length)
    length = length or 12
    vim.cmd("!openssl rand -base64 " .. length .. " | tr -d '\n=' | cut -c1-" ..
length)
end

-- Function to clean up imports/exports (Python example)
M.organize_imports = function()
    if vim.bo.filetype == "python" then
        vim.cmd([[
            let save_cursor = getpos('.')
            silent! %sort
            let current_line = 1
            let line_count = line('$')
            while current_line < line_count
                let line = getline(current_line)
                let next_line = getline(current_line + 1)
                if match(line, '^import') >= 0 && match(next_line, '^from') >= 0
```

```
                    execute current_line . ',' . (current_line + 1) . 'move ' .
(current_line + 1)
                    let line_count = line_count - 1
                else
                    let current_line = current_line + 1
                endif
            endwhile
            call setpos('.', save_cursor)
        ]])
        print("Imports organized")
    end
end

-- Function to comment out all TODOs/FIXMEs for quick review
M.highlight_todos = function()
    vim.cmd("/\\vTODO|FIXME|XXX|HACK")
    vim.cmd("set hlsearch")
end

-- Function to create a quick backup of entire project
M.backup_project = function()
    local project_root = vim.fn.system("git rev-parse --show-toplevel 2>/dev/null ||
pwd")
    project_root = string.gsub(project_root, '\n$', '')

    local backup_dir = project_root .. "_backup_" .. os.date("%Y%m%d_%H%M%S")
    vim.cmd("!cp -r '" .. project_root .. "' '" .. backup_dir .. "'")
    print("Project backed up to: " .. backup_dir)
end

-- Function to search and replace with confirmation across multiple files
M.multi_file_replace = function()
    local find_term = vim.fn.input("Find: ")
    local replace_term = vim.fn.input("Replace with: ")

    if find_term ~= "" and replace_term ~= "" then
        vim.cmd("grep -r '" .. find_term .. "' .")
        vim.cmd("cdo %s/" .. find_term .. "/" .. replace_term .. "/gc")
    end
end

-- Function to create a simple todo list in current buffer
M.create_todo_list = function()
    local todo_items = {
        "## TODO List",
        "",
        "- [ ] Task 1",
        "- [ ] Task 2",
        "- [ ] Task 3",
        "",
        "## Done",
        "",
```

```
        "- [x] Setup TODO list"
    }

    vim.api.nvim_buf_set_lines(0, 0, 0, false, todo_items)
end

-- Function to toggle task completion in markdown todo list
M.toggle_todo = function()
    local line = vim.fn.getline(".")
    if string.match(line, "%[ %]") then
        vim.cmd("s/\\[ \\]/[x]/")
    elseif string.match(line, "%[x\\]") then
        vim.cmd("s/\\[x\\]/[ ]/")
    end
end

-- Function to create a daily journal entry
M.create_journal_entry = function()
    local date_str = os.date("%Y-%m-%d")
    local day_str = os.date("%A")

    local journal_content = {
        "# " .. date_str .. " - " .. day_str,
        "",
        "## Accomplishments",
        "",
        "- ",
        "",
        "## Challenges",
        "",
        "- ",
        "",
        "## Learnings",
        "",
        "- ",
        "",
        "## Tomorrow's Goals",
        "",
        "- "
    }

    vim.api.nvim_buf_set_lines(0, 0, 0, false, journal_content)
    vim.cmd("normal! G")
end

-- Setup key mappings for these new functions
M.setup_additional_keymaps = function()
    local set_keymap = vim.api.nvim_set_keymap

    -- Template and documentation
    set_keymap('n', '<leader>st', '<cmd>lua
require("user.additional").create_snippet_template()<CR>', { noremap = true, silent
```

```
= true })
    set_keymap('n', '<leader>doc', '<cmd>lua
require("user.additional").generate_docstring()<CR>', { noremap = true, silent =
true })

    -- Text wrapping and extraction
    set_keymap('v', '<leader>w', '<cmd>lua
require("user.additional").wrap_html_tag()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>m', '<cmd>lua
require("user.additional").wrap_markdown()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>ev', '<cmd>lua
require("user.additional").extract_variable()<CR>', { noremap = true, silent = true
})

    -- Data conversion
    set_keymap('v', '<leader>jy', '<cmd>lua
require("user.additional").json_to_yaml()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>yj', '<cmd>lua
require("user.additional").yaml_to_json()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>b64e', '<cmd>lua
require("user.additional").base64_encode()<CR>', { noremap = true, silent = true })
    set_keymap('v', '<leader>b64d', '<cmd>lua
require("user.additional").base64_decode()<CR>', { noremap = true, silent = true })

    -- Utilities
    set_keymap('n', '<leader>uuid', '<cmd>lua
require("user.additional").generate_uuid()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>pass', '<cmd>lua
require("user.additional").generate_password()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>toc', '<cmd>lua require("user.additional").create_toc()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>mt', '<cmd>lua
require("user.additional").jump_to_matching_tag()<CR>', { noremap = true, silent =
true })

    -- Theme and timing
    set_keymap('n', '<leader>tt', '<cmd>lua
require("user.additional").toggle_theme()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>ts', '<cmd>lua require("user.additional").start_timer()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>te', '<cmd>lua require("user.additional").stop_timer()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>bm', '<cmd>lua
require("user.additional").benchmark_code()<CR>', { noremap = true, silent = true })

    -- Project utilities
    set_keymap('n', '<leader>oi', '<cmd>lua
require("user.additional").organize_imports()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>ht', '<cmd>lua
require("user.additional").highlight_todos()<CR>', { noremap = true, silent = true
```

```
})
    set_keymap('n', '<leader>bp', '<cmd>lua
require("user.additional").backup_project()<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>mfr', '<cmd>lua
require("user.additional").multi_file_replace()<CR>', { noremap = true, silent =
true })

    -- Personal productivity
    set_keymap('n', '<leader>td', '<cmd>lua
require("user.additional").create_todo_list()<CR>', { noremap = true, silent = true
})
    set_keymap('n', '<leader>tt', '<cmd>lua require("user.additional").toggle_todo()
<CR>', { noremap = true, silent = true })
    set_keymap('n', '<leader>jr', '<cmd>lua
require("user.additional").create_journal_entry()<CR>', { noremap = true, silent =
true })
end

return M
```

Key New Features:

Code Generation & Templates

- `create_snippet_template()` : Generates language-specific boilerplate code
- `generate_docstring()` : Creates formatted documentation for functions
- `extract_variable()` : Extracts selected text into a variable

Text Transformation

- `wrap_html_tag()` : Wraps selection with HTML tags
- `wrap_markdown()` : Applies markdown formatting
- `json_to_yaml()` / `yaml_to_json()` : Converts between formats
- `base64_encode()` / `base64_decode()` : Encodes/decodes Base64

Productivity Tools

- `create_toc()` : Generates table of contents for Markdown
- `create_todo_list()` : Sets up a structured todo list
- `create_journal_entry()` : Creates a daily journal template
- `toggle_todo()` : Toggles task completion status

Development Utilities

- `generate_uuid()` : Creates UUIDs
- `generate_password()` : Generates secure passwords
- `benchmark_code()` : Measures code execution time
- `organize_imports()` : Sorts and organizes imports

Project Management

- `backup_project()` : Creates full project backups
- `multi_file_replace()` : Search/replace across files
- `highlight_todos()` : Finds all TODO comments

Workflow Enhancements

- `start_timer()` / `stop_timer()` : Simple stopwatch functionality
- `toggle_theme()` : Switches between light/dark themes

These functions leverage Vimscript's pattern matching, external command integration, and text manipulation capabilities within Lua's structured approach, providing powerful tools for various development scenarios.
