
Learning VimScript
@PrabirShrestha – Sep 2020

OSS VIM Projects

- vim-lsp
- asyncomplete.vim
- quickpick.vim
- async.vim
- callbag.vim
- typescript-language-server

Vimscript v9

- This talk will not cover vimscript version 9.
- Vimscript/Viml in this presentation will refer to version <= 8.

Hello World Vimscript

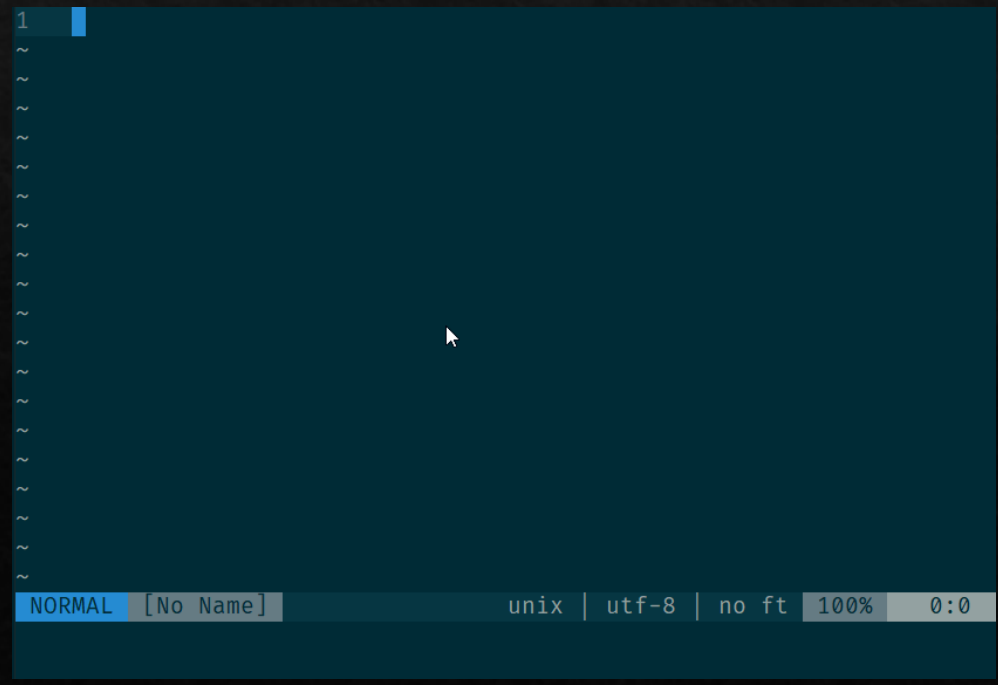
```
:echo "Hello world"
```

```
:let message = "Hello world"
:echo message
```

.vimrc

```
echo "Hello"
echo "World"
```

```
echom "Hello"
echom "World"
```



What is Viml/Vimscript <= 8?

- Interpreted
- Imperative
- Dynamic
- Partial Runtime Type Checks
- Functional
- Optional Simple Object Oriented

:help

:help is your best friend

Variables

:help variables

- string
- number
- float
- null
- dictionary
- list
- funcref
- Job
- Channel
- Blob

:help variables

```
let string = "String"
let number = 1
let float = 1.0
let list = [1, 2, "hello", "world" ]
let dictionary = { "message": "hello world" }
```

Multiline statements

```
" list with default values
let list = [
  \ 1, 2,
  \ "hello",
  \ "world" ]

" dictionary default values
let dictionary = {
  \ "message": "hello world"
  \ }
```

Comments in multiline statements???

```
let list = [
  \ 1, " first
  \ 2, " second
  \ "hello", " third
  \ "world" " forth
  \ ]
```

Special Variables

```
let t = v:true
let f = v:false
let n = v:null
```

What is the scope of the variable???

```
let string = "String"
let number = 1
let float = 1.0
let list = [1, 2, "hello", "world" ]
let dictionary = { "message": "hello world" }
```

Variable Scoping

```
let g:global_var = "global variable accessible across all script"
let s:script_var = "script level variable accessible across the entire script/file"
let l:local_var = "local variable accessible only in the scope"
let b:buffer_var = "buffer scope"
let w:window_var = "window scope"
let t:tab_var = "tab scope"
```

Functions

:h function

```
let s:greeting = 'Hello'
function g:Greet(name)
    echom s:greeting . " " . a:name
endfunction

call g:Greet("world")
```

function: error handling

```
function g:Greet(message)
    echom s:greeting
    echom a:name
endfunction

call g:Greet("world")
```

```
NORMAL  [No Name]
Messages maintainer: Bram Moolenaar <Bram@vim.org>
Error detected while processing C:\Users\prshrest\.vimrc[375]..
line    1:
E121: Undefined variable: s:greeting
line    2:
E121: Undefined variable: a:name
Press ENTER or type command to continue
```

function: abort

```
function g:Greet(message) abort
    echom s:greeting
    echom a:name
endfunction

call g:Greet("world")
```

```
NORMAL  [No Name]
Messages maintainer: Bram Moolenaar <Bram@vim.org>
Error detected while processing C:\Users\prshrest\.vimrc[375]..
line    1:
E121: Undefined variable: s:greeting
Press ENTER or type command to continue
```

try..catch

```
try
    throw 'this is an error message'
catch
    echom v:exception
    echom v:throwpoint
endtry
```

```
~
 NORMAL  [No Name]
Messages maintainer: Bram Moolenaar <Bram@vim.org>
this is an error message
C:\Users\prshrest\.vimrc[375]..C:\Users\prshrest\.vimrc.local, line 12
Press ENTER or type command to continue
```

:h E128

```
try
    function g:hello() abort
    endfunction

    call g:hello()
catch /E128/
    echom 'caught E128:'
    echom v:exception
    echom v:throwpoint
catch /.*/
    echom 'catch'
finally
    echom 'finally'
endtry
```

variable function arguments ...

```
function! s:log(message, ...) abort
    echom a:message
    if a:0 > 0
        echom a:1
    endif
endfunction

call s:log("invalid error")
call s:log("invalid buffer", 1)
```

if | ... | endif

```
function! s:log(message, ...) abort
    echom a:message
    if a:0 > 2 | return | endif
    if a:0 > 0 | echom a:1 | endif
    if a:0 > 1 | echom a:2 | endif
endfunction

call s:log("invalid error")
call s:log("invalid buffer", 1)
```

a:000

```
function! s:log(message, ...) abort
    echom a:message
    for l:item in a:000
        echom l:item
    endfor
endfunction

call s:log("invalid error")
call s:log("invalid buffer", 1)
call s:log("invalid buffer", 1, 2, "three")
```

Strings

String Concatenation

```
let msg = 'Hello' . ' ' . 'world'
```

noignorecase

```
set noignorecase

function! s:log(word) abort
    if a:word == 'one'
        echom '1'
    elseif a:word == 'two'
        echom '2'
    else
        echom 'unknown'
    endif
endfunction

call s:log('one')
call s:log('One')
```

Comparison Operators

	use 'ignorecase'	match case	ignore case
equal	==	==#	==?
not equal	!=	!=#	!=?
greater than	>	>#	>?
greater than or equal	>=	>=#	>=?
smaller than	<	<#	<?
smaller than or equal	<=	<=#	<=?
regexp matches	=~	=~#	=~?
regexp doesn't match	!~	!~#	!~?
same instance	is	is#	is?
different instance	isnot	isnot#	isnot?

Single vs Double quotes

```
echo 'hello\nworld'
" prints
" hello\nworld
```

```
echo "hello\nworld"
" prints:
" hello
" world
```

string slice

```
let message = "helloworld"
echom message[0:4]
" hello

echom message[5:]
" world

echom message[-5:]
" world

echom message[:-6]
" hello
```

List & Dictionary

:h list
:h dict

Learn the apis:

- add
- remove
- has_key
- copy
- keys
- items()

Type Checking

```
echom type('hello') == v:t_string
echom type('world') == type('')
```

Callback

```
function! s:greet(name) abort
    call timer_start(1000, function('s:timer_callback'))
endfunction

function! s:timer_callback(timer) abort
    echom 'Hello name???'
endfunction
```

Callback

```
function! s:greet(name) abort
    call timer_start(1000, function('s:timer_callback', [a:name]))
endfunction

function! s:timer_callback(name, timer) abort
    echom 'Hello ' . a:name
endfunction
```

Lambda

```
function! s:greet(name) abort
    call timer_start(1000, {t->s:timer_callback(name)})
endfunction

function! s:timer_callback(name) abort
    echom 'Hello ' . a:name
endfunction
```

Events

:h autocmd

```
autocmd CursorMoved,CursorMovedI * call s:cursor_moved()

function! s:cursor_moved() abort
    echom "cursor moved"
endfunction
```

:h autocmd-groups

```
augroup mygroup
    autocmd CursorMoved,CursorMovedI * call s:cursor_moved()
augroup END


function! s:cursor_moved() abort
    echom "cursor moved"
endfunction
```

:h autocmd-groups

```
augroup mygroup
    autocmd!
    autocmd CursorMoved,CursorMovedI * call s:cursor_moved()
augroup END


function! s:cursor_moved() abort
    echom "cursor moved"
endfunction
```

:h write-plugin

Writing your first plugin

Hello World Plugin

```
Plug 'prabirshrestha/helloworld'
```

```
helloworld
|--plugin/helloworld.vim
|--doc/helloworld.txt
```

plugin/helloworld.vim

```
if exists('g:loaded_helloworld') | return | end
if let g:loaded_helloworld = 1
```

Adding first command

```
if exists('g:loaded_helloworld') | finish | endif
let g:loaded_helloworld = 1

" user can call using :Greet command
command Greet call s:greet()

function! s:greet() abort
    echom 'hello world’
endfunction
```

autoload

```
helloworld
|--autoload/helloworld/greetings.vim
|--plugin/helloworld.vim
|--doc/helloworld.txt
```

autoload

```
" auoload/helloworld/greetings.vim
function helloworld#greetings#good_morning() abort
    echom 'Good morning’
endfunction

" plugin/helloworld.vim
command Greet call helloworld#greetings#good_morning()
```

autoload – greetings.vim

```
function helloworld#greetings#good_morning() abort
    echom 'Good morning’
endfunction

function! helloworld#greetings#good_afternoon() abort
    call s:good_afternoon()
endfunction

function! s:good_afternoon()
    echom 'Good afternoon’
endfunction
```

Organizing files

```
helloworld
|--autoload/helloworld/greetings.vim
|--autoload/helloworld/utils.vim
|--plugin/helloworld.vim
|--doc/helloworld.txt
```

Organizing files

```
function! helloworld#greetings#greet() abort
    let l:hour = helloworld#greetings#utils#get_hour()
    if l:hour < 10
        call helloworld#greetings#good_morning()
    elseif l:hour < 17
        call helloworld#greetings#good_afternoon()
    else
        echom 'Good evening’
    endif
endfunction
```

Mappings

```
nnoremap <plug>(helloworld-greet) :<c-u>call helloworld#greetings#greet()<cr>

if !hasmapto('<Plug>(helloworld-greet)', 'n') && (mapcheck("<F9>", "n") == "")
    nmap <silent> <F9> <plug>(helloworld-greet)
endif
```

Version & Feature check

```
if has('nvim')
    echom 'in neovim'
else
    echom in 'vim'
endif

if has('patch-8.1.889')
    echom 'has patch 8.1.889'
endif

if !exists('timer')
    echom 'vim compiled with timer required'
endif

if exists('##TextChangedP')
    autocmd TextChangedP * call s:on_text_changed_p()
endif
```

Best Practices

- Set up CI
  - Linting
  - Tests
- Document your plugin in vim doc format
- Document your code, special attention to data structures used in code

callbag.vim

```
let s:Dispose = lsp#callbag#pipe(
    \ lsp#callbag#merge(
    \   lsp#callbag#fromEvent(['CursorMoved', 'CursorHold']),
    \   lsp#callbag#pipe(
    \       lsp#callbag#fromEvent(['InsertEnter', 'BufLeave']),
    \       lsp#callbag#tap({_ -> s:clear_highlights() }),
    \   )
    \ ),
    \ lsp#callbag#filter({_ -> g:lsp_highlight_references_enabled }),
    \ lsp#callbag#debounceTime(g:lsp_highlight_references_delay),
    \ lsp#callbag#map({_->{'bufnr': bufnr('%'), 'curpos': getcurpos()[0:2], 'changedtick': b:changedtick }}),
    \ lsp#callbag#distinctUntilChanged({a,b -> a['bufnr'] == b['bufnr'] && a['curpos'] == b['curpos'] && a['changedtick'] == b['changedtick']}),
    \ lsp#callbag#filter({_->mode() is# 'n' && getbufvar(bufnr('%'), '&buftype') !=# 'terminal' }),
    \ lsp#callbag#switchMap({_->
    \   lsp#callbag#pipe(
    \       s:send_highlight_request(),
    \       lsp#callbag#materialize(),
    \       lsp#callbag#filter({x->lsp#callbag#isNextNotification(x)}),
    \       lsp#callbag#map({x->x['value']}),
    \       lsp#callbag#takeUntil(
    \           lsp#callbag#fromEvent('BufLeave')
    \       )
    \   )
    \ }),
    \ lsp#callbag#filter({_->mode() is# 'n'}),
    \ lsp#callbag#subscribe({x->s:set_highlights(x)}),
    \)
```

Vimscript for the Javascripter

https://w0rp.com/blog/post/vim-script-for-the-javascripter/

What about Lua instead of Vimscript?

Vim and Neovim lua compatiblity #12537

Open prabirshrestha opened this issue on Jun 25 · 15 comments

prabirshrestha commented on Jun 25 • edited

Creating this to track all the incompatibilities across vim and neovim lua compatibility based on the comment by @tjdevries at #12507 (comment).

This is the list I think we should try to maintain 100% compatibility here.

Feature	Neovim	Vim	Notes
Add support for `require('plugin')`	Supported	Supported since 8.2.0858	Neovim PR ported to vim
`vim.call`	Supported since v0.5	Supported since 8.2.0775	Neovim PR ported to Vim
`vim.fn.*`	Supported since v0.5	Supported since 8.2.0775	Neovim PR ported to Vim
Allow lua closures to be passed to vim script	in PR #12507	Supported since 8.2.1054	Vim PR ported to Neovim
`vim.eval`	Not supported	Supported < 8.0	Workaround for neovim: use `vim.api.nvim_eval` instead of `vim.eval`
`vim.command`	Not supported	Supported < 8.0	Workaround for neovim: use `vim.api.nvim_command` instead of `vim.command`
array index should start with 1	prints `1`	~~prints `0`~~ prints `1`	~~`let s:array = [1,2,3] lua print(vim.api.nvim_eval('s:array')[0])` This causes off by 1 bugs a lot in vim and ncovim plugins.~~ Starting with patch-8.2.1066, vim now behaves the same as lua and neovim.
`vim.list({})`	Not supported	Supported < 8.0	Neovim throws error
`vim.dict({})`	Not supported	Supported < 8.0	Neovim throws error
`vim.funcref({})`	Not supported	Supported < 8.0	Neovim throws error
`vim.blob({})`	Not supported	Supported < 8.0	Neovim throws error
`type(vim.list({}))`	Returns table	Returns userdata	`let s:array = [1,2,3] lua print(type(vim.eval('s:array')))`
`vim.type()`	Throws error	Returns list	`let s:array = [1,2,3] lua print(vim.type(vim.eval('s:array')))`
`table.insert` and `table.remove` for vim arrays	Partially supported	Supported since 8.2.1081 requires lua >= 5.3	
`autocmd` apis	In PR #12378	Not Supported	vim/vim#6339

Vim & Neovim lua compatibility

https://github.com/neovim/neovim/issues/12537

Thank You!!!

- twitter.com/PrabirShrestha
- github.com/PrabirShrestha