



Python으로 만드는 NEOVIM ASYNC PLUGIN

송재학 (master@hpi.cc)

발표자

송재학

- 현재 백수이지만 열심히 살고 있습니다.
- (전) 문래빗 주식회사 대표
 - 판타지x러너즈 for 카카오 서버 개발 (python)
- omnisharp-sublime (sublime text를 위한 C# IDE 플러그인)

저는 오랫동안 vim을 주력 에디터로 쓰고 싶었습니다.

vim은 위대한 텍스트 에디터이지만

매우 주관적인 장애물이 좀 있습니다.



- 로고에 그라데이션이 없음
- 힙하지 않음
- VimScript 코딩하기 쉽음
- 대부분의 플러그인이 sync로 동작함





말그대로 Vim의 미래

Vim을 적극적으로 리팩토링한 superset



- 로고에 그라데이션이 있는가? (O)



- 로고에 그라데이션이 있는가? (O)
- 힙한가? (O)



- 로고에 그라데이션이 있는가? (O)
- 힙한가? (O)
- VimScript 없이 Python으로 플러그인 개발을 할 수 있는가? (O)



- 로고에 그라데이션이 있는가? (O)
- 힙한가? (O)
- VimScript 없이 Python으로 플러그인 개발을 할 수 있는가? (O)
- Async 플러그인 개발이 쉬운가? (O)

다 좋은데, 안정적일까?

현황

(2016.08.12 기준)

- stable version: v0.1.4
 - 현재 안정적이고 실사용 가능.

어느날 갑자기 망하지는 않을까?

어느날 갑자기 망하지는 않을까?

 Watch ▾

956

 Unstar

19,600

 Fork

1,385



지금 당장 Neovim을 설치해야겠다!

특징

공식 사이트에 따르면...

- 더 강력한 플러그인
- 더 나은 기본 기능과 설정
- 기본으로 내장된 embedding
- Vim에서 쉽게 옮길 수 있음

더 강력한 플러그인

- 오늘은 여기에 집중합시다.

Msgpack-RPC

- Neovim은 서버입니다.
- stdin/stdout, socket 등을 통해 Neovim의 RPC API를 사용할 수 있습니다.

현재 API 클라이언트 모듈이 있는 플랫폼

- C#
- C++
- Clojure
- Common Lisp
- Elixir
- Filesystem
- Go
- Haskell
- Java
- Julia
- Lua
- Node.js
- Perl
- R
- Ruby
- Rust

...

그리고 당연히 Python!!

Hello World!

Hello World!

0. python-neovim 설치

```
> # neovim의 RPC API를 사용하기 위한 모듈  
> pip install neovim
```

1. Neovim 실행

```
> # NVIM_LISTEN_ADDRESS: neovim의 RPC 주소 지정  
> NVIM_LISTEN_ADDRESS=/tmp/nvim nvim
```

2. Python 쉘에서 명령어 실행

```
> python  
>>> from neovim import attach  
>>> nvim = attach('socket', path='/tmp/nvim')  
>>> nvim.command('echo "hello world!"')
```

Hello World!

The screenshot shows a tmux session with two panes. The left pane displays the Neovim welcome screen, which includes the following text:

```
[No Name] 2. pycon2016_program31: tmux (cat)
buffers
~/Projects/pycon2016_program31 gh-pages*
> workon neovim
pyt
~/Projects/pycon2016_program31 gh-pages*
(neovim) > python
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from neovim import attach
>>> nvim = attach('socket', path='127.0.0.1:121212')
>>> nvim.command('echo "hello world!"')
>>>
```

NVIM 0.1.4

by Bram Moolenaar et al.
빔은 소스가 열려 있고 공짜로 배포됩니다

First time using a vi-like editor?
Type :Tutor<Enter> to get started!

Already know your way around Vim?
See :help nvim-intro for an introduction to Neovim.

Still have questions?
Reach out to the Neovim community at neovim.io/community.

이에 대한 정보를 보려면 :help register<엔터> 입력
끝내려면 :q<엔터> 입력
온라인 도움말을 보려면 :help<엔터> 또는 <F1> 입력
type :help nvim<Enter> for Neovim help

A large blue arrow points downwards from the bottom of the Neovim pane towards the tmux status bar.

The tmux status bar at the bottom shows:

```
NORMAL [이름 없음]
hello world!
[14] 0:python> 100% 0: 1
"pycon2016_program31: " 22:21 02- 8-16
```

왜 Python REPL을 사용한거죠?

- 부담 없이 API를 테스트 해볼 수 있습니다.
- help 함수로 API 문서를 즉석에서 확인할 수 있습니다.
- 디버깅할 때도 유용함

"REMOTE PLUGIN"

Neovim의 Msgpack-RPC를 통해 동작하는 플러그인

간단한 리모트 플러그인을 만들어 봅시다.

파일 하나만 있으면 됩니다.

SimplePlugin 플러그인 예제

```
# ~/.config/nvim/rplugin/python/simple.py

import neovim

@neovim.plugin
class SimplePlugin(object):

    def __init__(self, nvim):
        self.nvim = nvim

    @neovim.function('SimpleFunc')
    def func(self, args):
        self.nvim.command('echo "simple func"')

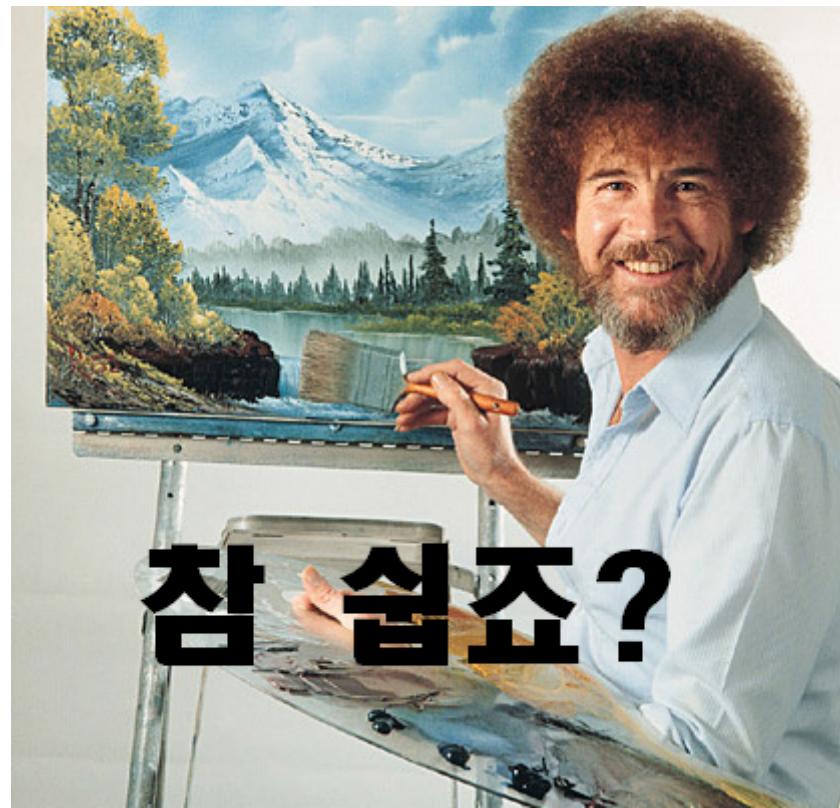
    @neovim.command('SimpleCommand', range='', nargs='*')
    def command(self, args, range):
        self.nvim.command('echo "simple command"')

    @neovim.autocmd('BufEnter', pattern="*.py")
    def autocmd(self):
        self.nvim.command('echo "simple autocmd"')
```

파일 작성 후, neovim상에서

```
:UpdateRemotePlugins
```

해주면 플러그인이 동작합니다!



참 쉽죠?

이제,

이제,

1. nvim의 RPC API를 파이썬 쉘에서 실행해볼 수 있습니다.

이제,

1. nvim의 RPC API를 파이썬 쉘에서 실행해볼 수 있습니다.
2. 리모트 플러그인을 만들 수 있습니다.
 1. 함수를 만들 수 있습니다.
 2. command를 만들 수 있습니다.
 3. 이벤트를 처리할 수 있습니다. (autocmd)



본격적으로 플러그인을 만들 준비가 되었습니다!

플러그인 만들어보기

사례

- 덕룡이는 최근 Python으로 알고리즘 문제를 풀고 있습니다.
- 그런데, 제대로 풀었는지 확인하기가 너무 귀찮습니다.
- 코드를 저장할 때마다 자동으로 체점이 되었으면 합니다.

우리가 해결할 수 있을 것 같습니다!

채점 스크립트 만들기

- Neovim을 캡니다.

채점 스크립트 만들기

- Neovim을 캡니다.



완성된 스크립트

```
> python checker.py solution.py
=====
input
=====
3
1 1
2 2
3 3

=====
expected output
=====
2
4
6

=====
output
=====
2
4
6

=====
run time: 0.039005
=====
succeed!
```

플러그인 만들기

- 이 스크립트를 Vim에 붙여봅시다.

커맨드 만들기

```
:CheckSolution <file 경로>
```

로 실행할 수 있는 커맨드를 만들어 봅시다.

커맨드 만들기

```
import neovim
import os
from .checker import check

@neovim.plugin
class AlgoTestPlugin(object):

    def __init__(self, nvim):
        self.nvim = nvim
```

플러그인 클래스를 정의해주고,

커맨드 만들기

```
@neovim.command('CheckSolution', nargs='*')
def command(self, args):
    for filename in args:
        dirname = os.path.dirname(filename)
        inputfile = os.path.join(dirname, 'input.txt')
        outputfile = os.path.join(dirname, 'output.txt')

        result = check(filename, inputfile, outputfile)

        self.nvim.command('echo "%s"' % result)
```

커맨드를 정의해줍니다.

커맨드 만들기

```
:CheckSolution solution.py
```

커맨드를 실행해봅니다.

커맨드 만들기

```
:CheckSolution solution.py
```

커맨드를 실행해봅니다.

```
=====
input
=====
2
1 1
2 2
=====
expected output
=====
2
4
=====
output
=====
2
4
=====
run time: 0.040526
=====
succeed!
```

잘됩니다.

그런데...

그런데...



- 하지만 매번 인자를 넣어줘야하는 것이 귀찮다네요.
- 인자를 생략하면 현재 파일의 경로가 들어가게 합시다.

그런데 API를 모르다보니, 어디서 시작을 해야할지 모르겠네요.

그런데 API를 모르다보니, 어디서 시작을 해야할지 모르겠네요.

Hello World!에서 했던 것처럼 Python 쉘을 사용해봅시다.

Python 쉘

```
>>> help(nvim.current)
# nvim.current에 buffer가 있는 것을 확인함.
```

Python 쉘

```
>>> help(nvim.current)
# nvim.current에 buffer가 있는 것을 확인함.
```

```
>>> help(nvim.current.buffer)
# buffer에 name이 있는 것을 확인함.
```

Python 쉘

```
>>> help(nvim.current)
# nvim.current에 buffer가 있는 것을 확인함.
```

```
>>> help(nvim.current.buffer)
# buffer에 name이 있는 것을 확인함.
```

```
>>> nvim.current.buffer.name
'/Users/jaehak/Projects/algotest/sample/solution.py'
```

Python 쉘

```
>>> help(nvim.current)
# nvim.current에 buffer가 있는 것을 확인함.
```

```
>>> help(nvim.current.buffer)
# buffer에 name이 있는 것을 확인함.
```

```
>>> nvim.current.buffer.name
'/Users/jaehak/Projects/algotest/sample/solution.py'
```

같은 폴더의 input.txt를 열고 확인해봅니다.

```
>> nvim.current.buffer.name
'/Users/jaehak/Projects/algotest/sample/input.txt'
```

이 API를 사용하면 될 것 같습니다!

바뀐 커맨드 코드

```
@neovim.command('CheckSolution', nargs='*')
def command(self, args):
+
+    if len(args) == 0:
+        args = [self.nvim.current.buffer.name]
+
    for filename in args:
        dirname = os.path.dirname(filename)
        inputfile = os.path.join(dirname, 'input.txt')
        outputfile = os.path.join(dirname, 'output.txt')

        result = check(filename, inputfile, outputfile)

        self.nvim.command('echo "%s"' % result)
```

바뀐 커マン드 코드

```
@neovim.command('CheckSolution', nargs='*')
def command(self, args):
+
+    if len(args) == 0:
+        args = [self.nvim.current.buffer.name]
+
+    for filename in args:
+        dirname = os.path.dirname(filename)
+        inputfile = os.path.join(dirname, 'input.txt')
+        outputfile = os.path.join(dirname, 'output.txt')

        result = check(filename, inputfile, outputfile)

        self.nvim.command('echo "%s"' % result)
```

```
:CheckSolution
```

```
=====
input
=====
3
...
```

잘 되네요.



그래도 귀찮다고 하네요.

문제는 어떻게 푸나 몰라.



그래도 귀찮다고 하네요.

문제는 어떻게 푸나 몰라.

- 현재까지는 :CheckSolution 커맨드를 실행해줘야합니다.
- 파일을 저장할 때, 자동으로 실행되었으면 좋겠네요.

플러그인 만들기 - `autocmd` 등록하기

저장할 때, 자동으로 실행되는 `autocmd`를 등록해봅시다.

Neovim

```
# autocmd 이벤트 리스트 확인하기  
:help autocommand-events
```

버퍼를 파일로 저장한 이후 실행되는 'BufWritePost' 이벤트를 발견했습니다.

BufWritePost autocmd

'BufWritePost' autocmd를 등록합시다.

BufWritePost autocmd

'BufWritePost' autocmd를 등록합시다.

```
@neovim.autocmd("BufWritePost", pattern="*.py", sync=True)
def on_bufwrite_post(self):
    filename = self.nvim.current.buffer.name

    dirname = os.path.dirname(filename)
    inputfile = os.path.join(dirname, 'input.txt')
    outputfile = os.path.join(dirname, 'output.txt')

    # input/output 파일이 있는 경우에만 실행
    if all([os.path.exists(path) for path in [inputfile, outputfile]]):
        self.nvim.command('CheckSolution %s' % filename)
```

Neovim

```
:w
```

```
=====
input
=====
3
1 1
2 2
3 3

=====
expected output
=====
2
4
6
...
```

역시 잘 동작합니다.

A screenshot of a terminal window with a dark background. On the left, there is a code editor pane titled "solution.py" containing the following Python code:

```
1 import time
2 def add(a, b):
3     return a + b
4
5 def main():
6     count = int(raw_input())
7     for i in xrange(count):
8         a, b = map(int, raw_input().split(' '))
9         print(add(a, b))
10
11 if __name__ == "__main__":
12     main()
```

The code defines a function `add` that adds two integers. It then defines a `main` function that reads an integer `count` from standard input, loops `count` times, reads two integers `a` and `b` from standard input separated by a space, and prints the result of `add(a, b)`. Finally, it checks if the script is run directly and calls `main`.

At the bottom of the terminal window, the status bar shows the following information:

- NORMAL
- +0 ~0 -1
- sample/solution.py
- python
- utf-8[unix]
- 8%
- 1/12 ≡ : 1

그런데, 저장할 때마다 이러네요.



똥을 만들어 버렸습니다.

플러그인 만들기 - 출력용 버퍼 만들기

출력용 버퍼를 만들어서, 그 버퍼에 출력해봅시다.

API 파악하기

python 쉘을 활용해서 API를 파악해봅시다.

API 파악하기

python 쉘을 활용해서 API를 파악해봅시다.

버퍼 생성 테스트해보기

```
>>> nvim.command('set splitright')
>>> nvim.command('vnew')
# 오른쪽에 빈 버퍼가 생성됩니다.
# vsplit되서 포커싱 됨
```

API 파악하기

python 쉘을 활용해서 API를 파악해봅시다.

버퍼 생성 테스트해보기

```
>>> nvim.command('set splitright')
>>> nvim.command('vnew')
# 오른쪽에 빈 버퍼가 생성됩니다.
# vsplit되서 포커싱 됨
```

버퍼 가져오기

```
>>> b = nvim.current.buffer
>>> b.name
''
```

API 파악하기

python 쉘을 활용해서 API를 파악해봅시다.

버퍼 생성 테스트해보기

```
>>> nvim.command('set splitright')
>>> nvim.command('vnew')
# 오른쪽에 빈 버퍼가 생성됩니다.
# vsplit되서 포커싱 됨
```

버퍼 가져오기

```
>>> b = nvim.current.buffer
>>> b.name
''
```

모든 버퍼 리스트 가져오기

```
>>> names = [b.name for b in nvim.buffers]
>>> print(names)
['', '/Users/jaehak/Projects/algotest/sample/solution.py']
```

플러그인 만들기 - 출력용 버퍼 만들기

버퍼 설정하기

```
# 이름 설정  
>>> b.name = "__algotest_result"  
  
# 저장 안되고, swap파일 생성 안되게 설정  
>>> nvim.command("setlocal buftype=nofile noswapfile")
```

플러그인 만들기 - 출력용 버퍼 만들기

버퍼 내용 조작해보기

```
>>> b[0]  
'  
>>> b[0] = 'Hello World!'  
# 1번째 라인이 'Hello World!'로 바뀜  
  
>>> b.append('Hello World2')  
# 2번째 라인에 'Hello World2'가 추가됨  
  
>>> del b[0]  
# 1번째 라인을 제거함
```

플러그인 만들기 - 출력용 버퍼 만들기

버퍼 내용 조작해보기

```
>>> b[0]  
'  
>>> b[0] = 'Hello World!'  
# 1번째 라인이 'Hello World!'로 바뀜  
  
>>> b.append('Hello World2')  
# 2번째 라인에 'Hello World2'가 추가됨  
  
>>> del b[0]  
# 1번째 라인을 제거함
```

vim 화면

```
1  
~  
~  
~ | 1 Hello World2  
| ~  
| ~  
| ~
```

플러그인 만들기 - 출력용 버퍼 만들기

필요한 API는 모두 파악했습니다.

플러그인 만들기 - 출력용 버퍼 만들기

필요한 API는 모두 파악했습니다.

이제 플러그인 코드만 짜면 됩니다.

플러그인 만들기 - 출력용 버퍼 만들기

일단 필요한 메소드들을 정의해줍시다.

플러그인 만들기 - 출력용 버퍼 만들기

일단 필요한 메소드들을 정의해줍시다.

1. 버퍼를 생성하거나 가져오는 메소드

```
def create_or_get_buffer(self, name):
    # 이미 있는 버퍼 중, 이름이 일치하는 버퍼가 있으면 리턴
    for b in self.nvim.buffers:
        bname = os.path.basename(b.name)

        if bname == name:
            return b

    # 새 버퍼 생성
    self.nvim.command('set splitright')
    self.nvim.command('vnew')

    b = self.nvim.current.buffer
    b.name = name
    self.nvim.command("setlocal buftype=nofile noswapfile")
    return b
```

플러그인 만들기 - 출력용 버퍼 만들기

2. 버퍼를 초기화는 메소드

```
def clear_buffer(self, buffer):  
    buffer[:] = []
```

3. 버퍼에 출력을 추가해주는 메소드

```
def append_text(self, buffer, text):  
    lines = text.splitlines()  
    buffer[len(buffer):] = lines
```

플러그인 만들기 - 출력용 버퍼 만들기

그리고 command에서 출력을 echo 대신 버퍼를 사용하도록 합시다.

```
@neovim.command('CheckSolution', nargs='*')
def command(self, args):
    if len(args) == 0:
        args = [self.nvim.current.buffer.name]
+
+    buffer = self.create_or_get_buffer('__algotest_result')
+    self.clear_buffer(buffer)
+
    for filename in args:
        dirname = os.path.dirname(filename)
        infile = os.path.join(dirname, 'input.txt')
        outfile = os.path.join(dirname, 'output.txt')

        result = check(filename, infile, outfile)

-
-    self.nvim.command('echo "%s"' % result)
+    self.append_text(buffer, result)
```

결과를 확인해봅시다!

A screenshot of a terminal window titled "solution.py[+]" in the title bar. The window shows a Python script with syntax highlighting. The code defines a function `add` that takes two arguments `a` and `b` and returns their sum. It also defines a `main` function that reads an integer `count` from standard input, then loops `count` times, reading pairs of integers `a` and `b` from standard input, summing them, and printing the result. The script concludes with a check for the `__name__` variable. The status bar at the bottom indicates the file is in "NORMAL" mode, has changes (+1), and is at line -1. The encoding is "utf-8[unix]" and the cursor is at position 15% of the screen width. The buffer tab is visible on the right.

```
solution.py+
 1 import time
+ 2
 3 def add(a, b):
 4     return a + b
 5
- 6 def main():
- 7     count = int(raw_input())
 8     for i in xrange(count):
 9         a, b = map(int, raw_input().split(' '))
10         print(add(a, b))
11
12 if __name__ == "__main__":
13     main()
```

완성!



덕룡이도 만족할꺼예요.

그런데... 좀 허전하네요.

그런데... 좀 허전하네요.

제목이 'Python으로 만드는 NEOVIM ASYNC PLUGIN' 아니었나요?

그런데... 좀 허전하네요.

제목이 'Python으로 만드는 NEOVIM ASYNC PLUGIN' 아니었나요?

비동기는 어디갔죠?

그런데... 좀 허전하네요.

제목이 'Python으로 만드는 NEOVIM ASYNC PLUGIN' 아니었나요?

비동기는 어디갔죠?

이런식이면 저장할 때 엄청 느려지는 것 아닌가요?

실은....

이제까지 만든 것이 비동기 플러그인입니다.

실은....

이제까지 만든 것이 비동기 플러그인입니다.

저장할 때 느려지지 않아요.

neovim의 python-client 모듈을 사용하면

기본적으로 비동기로 동작합니다.

- function call
- command
- autocmd

필요하다면 sync로 동작하게 할 수는 있어요.

```
import neovim

@neovim.plugin
class SimplePlugin(object):

    def __init__(self, nvim):
        self.nvim = nvim

    - @neovim.function('SimpleFunc')
    + @neovim.function('SimpleFunc', sync=False)
        def func(self, args):
            self.nvim.command('echo "simple func"')

    - @neovim.command('SimpleCommand', range='', nargs='*')
    + @neovim.command('SimpleCommand', range='', nargs='*', sync=False)
        def command(self, args, range):
            self.nvim.command('echo "simple command"')

    - @neovim.autocmd('BufEnter', pattern="*.py")
    + @neovim.autocmd('BufEnter', pattern="*.py", sync=False)
        def autocmd(self):
            self.nvim.command('echo "simple autocmd"')
```

SYNC vs ASYNC

이제까지 만들어본 플러그인으로 비교해봅시다.

SYNC vs ASYNC

이제까지 만들어본 플러그인으로 비교해봅시다.

```
time.sleep(5)
```

이 들어가있는 코드에서 플러그인 사용해봅시다.

SYNC 버전

```
solution.py ➜ 1 import time
+ 2
3 def add(a, b):
4     return a + b
5
- 6 def main():
7     count = int(raw_input())
+ 8     time.sleep(5)
9     for i in xrange(count):
10         a, b = map(int, raw_input().split(' '))
11         print(add(a, b))
12
13 if __name__ == "__main__":
14     main()

NORMAL ➜ +2 ~0 -1 ➜ solution.py ➜ python ➜ utf-8[unix] ➜ 100% ⌂ 14/14 ≡ : 10
:w
```

SYNC 버전



이 얼마나 끔찍하고
무시무시한 플러그인이니?

끔찍하게도 실행시간 5.03초동안 vim이 프리징 됩니다.

ASYNC 버전

```
solution.py
1 import time
2
3 def add(a, b):
4     return a + b
5
6 def main():
7     count = int(raw_input())
8     time.sleep(5)
9     for i in xrange(count):
10        a, b = map(int, raw_input().split(' '))
11        print(add(a, b))
12
13 if __name__ == "__main__":
14     main()

NORMAL +2 ~0 -1 solution.py      python   utf-8[unix] 7% ↵ 1/14☰ : 1 ↶
```

ASYNC 버전



반면, ASYNC버전은 vim이 프리징되지 않습니다.

Neovim 플러그인 소개

- deoplete (<https://github.com/Shougo/deoplete.nvim>)
 - 비동기 자동완성 시스템
- Neomake (<https://github.com/neomake/neomake>)
 - 비동기 :make
- vim-grepper (<https://github.com/mhinz/vim-grepper>)
 - 쉽게 vim 상에서 Grep 사용할 수 있도록 도와주는 플러그인

최종 정리

1. neovim 좋아요.
 - 로고에 그라데이션도 들어가 있어요.

최종 정리

1. neovim 좋아요.
 - 로고에 그라데이션도 들어가 있어요.
2. neovim 플러그인은 python으로 쉽게 만들 수 있어요.

최종 정리

1. neovim 좋아요.
 - 로고에 그라데이션도 들어가 있어요.
2. neovim 플러그인은 python으로 쉽게 만들 수 있어요.
3. python REPL을 활용하면 neovim API를 쉽게 테스트해볼 수 있어요.



없으면 만들어서 씁시다.

우린 코딩할 수 있으니까요.

QnA

이 프레젠테이션 자료는
[remark.js](#)를 사용하여 작성되었으며,
[GitHub Pages](#)로 호스팅됩니다.

감사합니다.