Backend Development with Flask

Contents

[Backend Development with Flask 3](#_Toc112676990)

[Flask 3](#_Toc112676991)

[Managing Multiple Python Versions With pyenv 3](#_Toc112676992)

[pyenv Commands 4](#_Toc112676993)

[Installing Flask 5](#_Toc112676994)

[Setting up a new Python env environment for a new Project 5](#_Toc112676995)

[Running Flask 5](#_Toc112676996)

[Waking up Flask 6](#_Toc112676997)

[Render Template Function 6](#_Toc112676998)

[Introduction to Jinja2 6](#_Toc112676999)

[Python and Flask 7](#_Toc112677000)

[app.py 7](#_Toc112677001)

[Reference: 7](#_Toc112677002)

[Appendix: 8](#_Toc112677003)

[~/.profile: 8](#_Toc112677004)

[bashrc 8](#_Toc112677005)

# Backend Development with Flask

# Flask

* WSGI - Web Server Gateway Interface application framework
* Framework – code library to write code more easily
* Reliable, scalable and most importantly maintainable web applications
* Micro-framework – doesn’t do much
  + Receive user data
  + Send user data back
  + Can write pure python
  + Few rules, like other frameworks

# [Managing Multiple Python Versions With pyenv](https://realpython.com/intro-to-pyenv/#exploring-pyenv-commands)

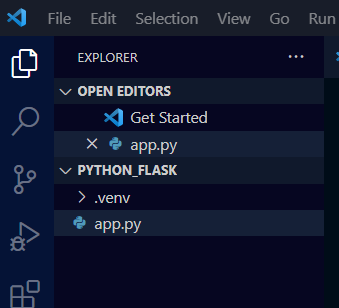
>> $ pyenv exec python –m venv .venv

|  |  |
| --- | --- |
| **pyenv**   * can easily try out new language features * contribute to a project that is on a different version of Python * “System Python” is the Python that comes installed on your operating system. * To install a package into your system Python, you have to run > sudo pip install. * Problems with multiple versions of the same package tend to creep up on you and bite you when you least expect it. * One common way this problem presents itself is a popular and stable package suddenly misbehaving on your system. After hours of troubleshooting and Googling, you may find that you’ve installed the wrong version of a dependency, and it’s ruining your day. | **Python REPL - Read Evaluate Print Loop**   * CLI interactive MicroPython shell that is accessible on the Pycom devices. Using the REPL is by far the easiest way to test out Python code and run commands. You can use the REPL in addition to writing scripts in main.py * In Python 3 mode, the REPL uses Python to control your computer. This version of the REPL is the most powerful. For example, as you type, the REPL will show you hints for the code you're using. * Repl.it is a free IDE (integrated development environment) that allows users to write their own programs and code in dozens of different languages. * To start the Python interpreter, type the command python without any parameter and hit the “return” key. |
| **pyenv advantages**  Python versions easily and flexibly:   1. Install Python in your user space 2. Install multiple versions of Python 3. Specify the exact Python version you want 4. Switch between the installed versions | **Dependency and Dependency Tests**   * package managers tend to install their packages into the global system space instead of the user space. Again, these system level packages pollute your development environment and make it hard to share a workspace with others * install Python from a package manager, consider what would happen if you’re writing a package and want to support and test on Python 3.4 - 3.7. |

# pyenv Commands

|  |  |
| --- | --- |
| **Command** | **Description** |
| $ pyenv commands | complete list of all available commands. Each command has a --help flag |
| $ pyenv install 3.6.8 |  |
| $ pyenv install -l  $ pyenv install --list | Lists all available Python versions for installation |
| $ pyenv install -g  $ pyenv install –debug | Builds a debug version of Python |
| $ pyenv install -v  $ pyenv install –verbose | Verbose mode: print compilation status to stdout |
| $ pyenv versions | displays all currently installed Python versions |
| $ python -v | Display system python |
| $ which python | Confirms physical location |
| $ pyenv which python | Confirms logical location |
| $ pyenv global 3.8-dev  $ python -m test | Tests version of Python you just installed to determine it is working properly using a built-in test suite. Kicks off lots of internal Python tests that will verify your installation. You can just kick back and watch the tests pass. |
| $ pyenv global 3.6.8 | The global command sets the global Python version. This can be overridden with other commands, but is useful for ensuring you use a particular Python version by default. If you wanted to use 3.6.8 by default |
| $ pyenv local 2.7.15 | The local command is often used to set an application-specific Python version. You could use it to set the version to 2.7.15 |
| $ pyenv shell 3.8-dev | set a shell-specific Python version. For example, if you wanted to test out the 3.8-dev version of Python |
| $ pyenv exec python -m venv .venv | Sets up the virtual environment named .venv for a project |
| $ pyenv root | Where the root directory for pyenv lives |
| $ pyenv versions | Versions of python installed |

# Installing Flask



1. PS C:\AWS\_FLASK\Python\_Flask> wsl
2. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ pyenv python exec -m venv .venv
   1. sets up >.venv
3. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ source .venv/bin/activate
   1. Activates .venv
4. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ pip install Flask
   1. Installs Flask in .venv
5. You can close the terminal

# Setting up a new Python env environment for a new Project

1. PS C:\AWS\_FLASK\Python\_Web\microblog> wsl  
     
   Check for local python
2. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Web/microblog$ pyenv versions
3. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Web/microblog$ pyenv local 3.9.12
4. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Web/microblog$ pyenv exec python  
     
   Create a new virtual environment for this project called .venv
5. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Web/microblog$ pyenv exec python -m venv .venv
6. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Web/microblog$ pip install flask
7. See “Wake up Flask”

# Running Flask

Tell Flask what the application to run is:

1. PS C:\AWS\_FLASK\Python\_Flask> wsl
2. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ export FLASK\_APP=app.py
   1. PS> $env:FLASK\_APP = app.py
   2. cmd> set FLASK\_APP = app.py
3. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ export FLASK\_ENV=development
4. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ flask run

\* Serving Flask app 'app.py' (lazy loading)

\* Environment: development

\* Debug mode: on

\* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 912-246-493

127.0.0.1 - - [16/May/2022 20:15:56] "GET / HTTP/1.1" 200 -

127.0.0.1 - - [16/May/2022 20:15:56] "GET /favicon.ico HTTP/1.1" 404 -

127.0.0.1 - - [16/May/2022 20:16:11] "GET / HTTP/1.1" 200 -

# Waking up Flask

1. PS C:\AWS\_FLASK\Python\_Flask> wsl
2. frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ source .venv/bin/activate
3. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ export FLASK\_APP=app.py
4. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ export FLASK\_ENV=development
5. (.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ flask run

# Render Template Function

* templates folder for static content
  + html

# Introduction to Jinja2

(.venv) frankt@IT-017969-FL:/mnt/c/AWS\_FLASK/Python\_Flask$ python

>>> from jinja2 import Template

>>> template = Template("Hello, {{name}}")

>>> template.render(name="Jack")

'Hello, Jack'

>>> template.render(name="Goerge")

'Hello, Goerge'

>>>

# Python and Flask

## app.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

The variable \_\_name\_\_ is passed as first argument when creating an instance of the Flask object (a Python Flask application). In this case \_\_name\_\_ represents the name of the application package and it’s used by Flask to identify resources like templates, static assets and the instance folder.

# Reference:

* [Python Virtual Environments: A Primer](https://realpython.com/python-virtual-environments-a-primer/#avoid-installing-pip)

# Appendix:

# ~/.profile:

~/.profile: executed by the command interpreter for login shells.

# This file is not read by bash(1), if ~/.bash\_profile or ~/.bash\_login

# exists.usrsa

# see /usr/share/doc/bash/examples/startup-files for examples.

# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask

# for ssh logins, install and configure the libpam-umask package.

#umask

# if running bash

if [ -n "$BASH\_VERSION" ]; then

# include .bashrc if it exists

if [ -f "$HOME/.bashrc" ]; then

. "$HOME/.bashrc"

fi

fi

# set PATH so it includes user's private bin if it exists

if [ -d "$HOME/bin" ] ; then

PATH="$HOME/bin:$PATH"

fi

# set PATH so it includes user's private bin if it exists

if [ -d "$HOME/.local/bin" ] ; then

PATH="$HOME/.local/bin:$PATH"

fi

# bashrc

# ~/.bashrc: executed by bash(1) for non-login shells.

# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)

# for examples

# If not running interactively, don't do anything

case $- in

\*i\*) ;;

\*) return;;

esac

# don't put duplicate lines or lines starting with space in the history.

# See bash(1) for more options

HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it

shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)

HISTSIZE=1000

HISTFILESIZE=2000

# check the window size after each command and, if necessary,

# update the values of LINES and COLUMNS.

shopt -s checkwinsize

# If set, the pattern "\*\*" used in a pathname expansion context will

# match all files and zero or more directories and subdirectories.

#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)

[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)

if [ -z "${debian\_chroot:-}" ] && [ -r /etc/debian\_chroot ]; then

debian\_chroot=$(cat /etc/debian\_chroot)

fi

# set a fancy prompt (non-color, unless we know we "want" color)

case "$TERM" in

xterm-color|\*-256color) color\_prompt=yes;;

esac

# uncomment for a colored prompt, if the terminal has the capability; turned

# off by default to not distract the user: the focus in a terminal window

# should be on the output of commands, not on the prompt

#force\_color\_prompt=yes

if [ -n "$force\_color\_prompt" ]; then

if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then

# We have color support; assume it's compliant with Ecma-48

# (ISO/IEC-6429). (Lack of such support is extremely rare, and such

# a case would tend to support setf rather than setaf.)

color\_prompt=yes

else

color\_prompt=

fi

fi

if [ "$color\_prompt" = yes ]; then

PS1='${debian\_chroot:+($debian\_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '

else

PS1='${debian\_chroot:+($debian\_chroot)}\u@\h:\w\$ '

fi

unset color\_prompt force\_color\_prompt

# If this is an xterm set the title to user@host:dir

case "$TERM" in

xterm\*|rxvt\*)

PS1="\[\e]0;${debian\_chroot:+($debian\_chroot)}\u@\h: \w\a\]$PS1"

;;

\*)

;;

esac

# enable color support of ls and also add handy aliases

if [ -x /usr/bin/dircolors ]; then

test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"

alias ls='ls --color=auto'

#alias dir='dir --color=auto'

#alias vdir='vdir --color=auto'

alias grep='grep --color=auto'

alias fgrep='fgrep --color=auto'

alias egrep='egrep --color=auto'

fi