How to determine if Python is running inside a virtualenv?

Asked 14 years, 7 months ago Modified 2 months ago Viewed 414k times

Is it possible to determine if the current script is running inside a virtualenv environment?

529

python virtualenv

Share Improve this question

Follow

edited Feb 5 at 5:24

Rob Bednark

27.2k • 24 • 84 • 126

asked Dec 9, 2009 at 4:18

miracle2k

31.2k • 21 • 66 • 66

3 Out of curiosity, why would you want to know that? – Jorge Leitao Sep 15, 2018 at 8:53

3 i.e. to be able to write custom script that generates prompt for your shell and you want that prompt to indicate if you are in venv or not, so you want to be able to detect that from within that code, best w/o calling external tools. – Marcin Orlowski Mar 2, 2019 at 0:19

Accepted answer didn't work for me (2020, python3.6.9 on Ubuntu 18.04) This answer worked: stackoverflow.com/a/42580137/1296044 – arielf Aug 9, 2020 at 3:04

Keep It Simple: (in Jupyter Notebook or Python 3.7.1 terminal on Windows 10) `import sys` `print(sys.executable)` # example output: >> `C:\Anaconda3\envs\quantecon\python.exe` OR `sys.base_prefix` # Example output: >> 'C:\\Anaconda3\\envs\\quantecon' – Rich Lysakowski PhD Aug 9, 2020 at 6:18

4 A potential time saver is going to ask *why* you need to find out how python is running. Rather than attempting to figure out how python is running after the fact, if your problem can be solved by understanding *how* python will be invoked before it happens, the solution becomes easy and much more universal. In that case, it becomes MrHetii's simple answer: stackoverflow.com/a/28388115/881411. i.e. if you control invoking python and you can check via shell prior to invoking for the presence of `${VIRTUAL_ENV}`, then you will know whether python will be running in a venv or not. – Jon Feb 16, 2021 at 19:36

20 Answers

Sorted by: Highest score (default)

362

The reliable and documented way is to compare `sys.prefix` and `sys.base_prefix`. If they're equal, you're not in a virtual environment, otherwise you are. Inside a venv, `sys.prefix` points to the directory of the virtual environment, and `sys.base_prefix` to the Python interpreter used to create the environment.

This is documented under *How venvs work*:

> It is sufficient to check `sys.prefix != sys.base_prefix` to determine if the current interpreter is running from a virtual environment.

This works for Python stdlib `venv` and for `virtualenv` (since version 20):

```
def in_venv():
    return sys.prefix != sys.base_prefix
```

Older versions of `virtualenv` used `sys.real_prefix` instead of `sys.base_prefix`, and `sys.real_prefix` did not exist outside a virtual environment. In Python 3.3 and earlier `sys.base_prefix` did not ever exist. So a check that also handles some legacy cases could look like this:

```
import sys


def get_base_prefix_compat():
    """Get base/real prefix, or sys.prefix if there is none."""
    return (
        getattr(sys, "base_prefix", None)
        or getattr(sys, "real_prefix", None)
        or sys.prefix
    )


def in_virtualenv():
    return sys.prefix != get_base_prefix_compat()
```

Using the `VIRTUAL_ENV` environment variable is not reliable. It is set by the virtualenv `activate` shell script, but a virtualenv can be used without activation by directly running an executable from the virtualenv's `bin/` (or `Scripts`) directory, in which case `$VIRTUAL_ENV` will not be set. Or a non-virtualenv Python binary can be executed directly while a virtualenv is activated in the shell, in which case `$VIRTUAL_ENV` may be set in a Python process that is not actually running in that virtualenv.

Share Improve this answer Follow

edited May 23, 2023 at 20:12

wim

353k ● 107 ● 648 ● 779

answered Dec 10, 2009 at 19:07

Carl Meyer

125k ● 20 ● 107 ● 117

61 If you are using virtualenv (github.com/pypa/virtualenv), this answer is equally correct for Python 2 or Python 3. If you are using pyvenv (legacy.python.org/dev/peps/pep-0405), a virtualenv-equivalent built into Python 3.3+ (but not the same thing as virtualenv), then it uses sys.base_prefix instead of sys.real_prefix, and sys.base_prefix always exists; outside a pyvenv it is equal to sys.prefix. – Carl Meyer May 1, 2014 at 19:21

3 @Kounavi I don't think it's likely that the Windows version would have any impact. This answer is a core part of how virtualenv works on any platform. Is it possible you are using Python 3 pyvenv, not virtualenv, on the Windows 2012 machine? Or that something is going on with the PATH and you are not in fact running in the virtualenv when you think you are? – Carl Meyer Jun 13, 2015 at 7:00

6 One-liner for bash scripts `PYTHON_ENV=$(python -c "import sys; sys.stdout.write('1') if hasattr(sys, 'real_prefix') else sys.stdout.write('0')")` – Sam Myers May 17, 2017 at 19:16

12 **This answer is obsolete,** unsurprisingly. Specifically, this answer returns false negatives for common use cases. That's bad. Instead, see either: hroncok's authoritative update correctly detecting all non-

Anaconda venvs or Victoria Stuart's authoritative answer correctly detecting all Anaconda venvs. (*All my upvotes for whoever combines those two answers.*) – Cecil Curry Nov 13, 2019 at 6:02

7 Updated the answer to account for stdlib `venv` module and changes in latest `virtualenv`. – Carl Meyer Sep 9, 2020 at 22:34

312

Try using `pip -V` (notice capital V)

If you are running the virtual env. it'll show the path to the env.'s location.

Share Improve this answer Follow

answered Aug 14, 2016 at 4:12

loki l

3,893 ● 2 ● 11 ● 10

If you've moved your `virtualenv` around a lot, it's possible this can fail or lie to you. If it's lying, you can do `find /path/to/venv/ -type f -exec sed -ie "s:/old/path/to/venv:/path/to/venv:g" {} \+`. If it's failing (I got "bad marshal data") you'll need to wipe the .pyc files with `find /path/to/venv -type f -name "*.pyc" -exec rm {} \+` (don't worry, they'll rebuild automatically). – jeremysprofile Oct 29, 2018 at 20:47

1 I just tested this on Windows 10 with Python 3.7. It prints the location of pip from the default install `...\lib\site-packages` in the `%PATH%`. So it will return a false positive in that case. – JamesThomasMoon Feb 12, 2019 at 5:16

9 On Ubuntu 18 with pip 21.0.1, `pip -V` returns path of user installation of pip, if not in a venv. Therefore, this doesn't tell you if a venv is activated or not. – Jon Feb 16, 2021 at 19:24

1 Works on Windows 10 if you are inside an env. – Gulzar Nov 7, 2021 at 14:52

1 The question is how to determine it from python script, not a command line. – Mikaelblomkvistsson Sep 5, 2022 at 7:44

103

Check the `$VIRTUAL_ENV` environment variable.

The `$VIRTUAL_ENV` environment variable contains the virtual environment's directory when in an active virtual environment.

```
>>> import os
>>> os.environ['VIRTUAL_ENV']
'/some/path/project/venv'
```

Once you run `deactivate` / leave the virtual environment, the `$VIRTUAL_ENV` variable will be cleared/empty. Python will raise a `KeyError` because the environment variable was unset.

```
>>> import os
>>> os.environ['VIRTUAL_ENV']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File
"/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/lib/python3
```

```
line 678, in __getitem__
    raise KeyError(key) from None
KeyError: 'VIRTUAL_ENV'
```

These same environment variable checks can of course also be done outside of the Python script, in the shell.

Share Improve this answer Follow edited Jun 19, 2019 at 1:23 answered Feb 7, 2015 at 22:17

George Pantazes
1,120 ● 3 ● 13 ● 26

MrHetii
1,425 ● 1 ● 11 ● 12

3 This works both for a `virtualenv` virtualenv and a `venv` virtualenv. – florisla Sep 19, 2019 at 7:36

@verboze: as it is supposed to do, right? A deactivated virtualenv means the user script is *not* running on one. – MestreLion Nov 8, 2019 at 21:12

4 This checks if a virtualenv is activated, but that doesn't necessarily mean that the Python process running is from that virtualenv. – erb Mar 19, 2020 at 17:00

6 This answer can be wrong in both directions. A Python process can be running in a virtual environment by direct execution of the virtualenv's Python binary (or a script with that binary as its shebang), without ever shell "activating" the virtualenv. In that case this answer will give a false negative. Alternatively, even with a virtualenv activated in the shell, a non-virtualenv Python binary or script could be directly executed, and this answer would give a false positive. – Carl Meyer Sep 9, 2020 at 22:38

1 It could be useful for edge use cases, like for instance if you want to test if the virtualenv is activated before running Python code from another language. In my case, my Node server spawns Python services here and there, and I want to check if the virtual env is activated before running Jest test. – Eric Burel Jul 7, 2021 at 13:30

102

This is an improvement of the accepted answer by Carl Meyer. It works with virtualenv for Python 3 and 2 and also for the venv module in Python 3:

```
import sys


def is_venv():
    return (hasattr(sys, 'real_prefix') or
            (hasattr(sys, 'base_prefix') and sys.base_prefix != sys.prefix))
```

The check for `sys.real_prefix` covers virtualenv, the equality of non-empty `sys.base_prefix` with `sys.prefix` covers venv.

Consider a script that uses the function like this:

```
if is_venv():
    print('inside virtualenv or venv')
else:
    print('outside virtualenv or venv')
```

And the following invocation:

```
$ python2 test.py
outside virtualenv or venv

$ python3 test.py
outside virtualenv or venv

$ python2 -m virtualenv virtualenv2
...
$ . virtualenv2/bin/activate
(virtualenv2) $ python test.py
inside virtualenv or venv
(virtualenv2) $ deactivate

$ python3 -m virtualenv virtualenv3
...
$ . virtualenv3/bin/activate
(virtualenv3) $ python test.py
inside virtualenv or venv
(virtualenv3) $ deactivate

$ python3 -m venv venv3
$ . venv3/bin/activate
(venv3) $ python test.py
inside virtualenv or venv
(venv3) $ deactivate
```

Share Improve this answer Follow edited Sep 29, 2017 at 14:49 answered Mar 3, 2017 at 13:26

hroncok
1,207 ● 1 ● 9 ● 12

8 Since Python 3.3 is no longer maintained or supported by most Python 3 frameworks and applications, this function now reduces to a trivial one-liner: `def is_venv(): return hasattr(sys, 'real_prefix') or sys.base_prefix != sys.prefix`. *Just sayin'*. – Cecil Curry Nov 13, 2019 at 6:36

Sadly this doesn't appear to work with `pipenv` created virtual environments. – dragon788 Feb 24, 2020 at 16:43

Works with classic venv and pipenv too (both through `pipenv run` and `pipenv shell`) for me (MacOS Ventura on Mac M1, Python 3.9.10) – Jens Roland Jan 19, 2023 at 19:12

58

There are multiple good answers here, and some less robust ones. Here's an overview.

How not to do it

Do not rely on on the location of Python or the `site-packages` folder.

If these are set to non-standard locations, that does *not* mean you're actually in a virtual environment. Users can have more than one Python version installed, and those are not always where you expect them to be.

Avoid looking at:

- `sys.executable`
- `sys.prefix`
- `pip -V`
- `which python`

Also, do not check for the presence of `venv`, `.venv` or `envs` in any of these paths. This will break for environments with a more unique location. For example, Pipenv uses hash values as the name for its environments.

`VIRTUAL_ENV` environment variable

Both `virtualenv` and `venv` set the environment variable `$VIRTUAL_ENV` when activating an environment. See PEP 405.

You can read out this variable in shell scripts, or use this Python code to determine if it's set.

```
import os
running_in_virtualenv = "VIRTUAL_ENV" in os.environ

# alternative ways to write this, also supporting the case where
# the variable is set but contains an empty string to indicate
# 'not in a virtual environment':
running_in_virtualenv = bool(os.environ.get("VIRTUAL_ENV"))
running_in_virtualenv = bool(os.getenv("VIRTUAL_ENV"))
```

The problem is, this only works when the environment is *activated* by the `activate` shell script.

You can start the environment's scripts *without activating the environment*, so if that is a concern, you have to use a different method.

`sys.base_prefix`

`virtualenv`, `venv` and `pyvenv` point `sys.prefix` to the Python installed inside of the virtualenv as you would expect.

At the same time, the *original* value of `sys.prefix` is also made available as `sys.base_prefix`.

We can use that to detect if we're in a virtualenv.

```
import sys
# note: Python versions before 3.3 don't have sys.base_prefix
# if you're not in virtual environment
running_in_virtualenv = sys.prefix != sys.base_prefix
```

Fallback: `sys.real_prefix`

Now watch out, `virtualenv` before version 20 did not set `sys.base_prefix` but it set `sys.real_prefix` instead.

So to be safe, check both as suggested in hroncok's answer:

```
import sys

real_prefix = getattr(sys, "real_prefix", None)
base_prefix = getattr(sys, "base_prefix", sys.prefix)

running_in_virtualenv = (base_prefix or real_prefix) != sys.prefix
```

Anaconda

If you're using Anaconda virtual environments, check Victoria Stuart's answer.

Share Improve this answer Follow edited Jun 19, 2020 at 16:15 answered Sep 20, 2019 at 10:51

florisla
13.3k ● 6 ● 42 ● 47

2 The OP is asking "How do I?", not "How NOT to?" This answer is overkill. It goes beyond the spirit of the question and obfuscates the answer with too many variations. Please keep your answers as simple as possible and answer the question directly. – Rich Lysakowski PhD Nov 20, 2019 at 6:21

24 I am summarizing the multiple answers here, and providing advice on which one might be right for specific circumstances. The original question does not provide enough context to select one of these techniques as the 'best' one -- it just isn't that simple. – florisla Nov 27, 2019 at 10:33

1 In the section for sys.base_prefix, shouldn't the test be: `running_in_virtualenv = sys.*base_*prefix != sys.prefix` – usonianhorizon Jun 19, 2020 at 14:31

1 Thank you, @florisia! I am in process of transitioning from virtualenvwrapper to the built-in venv for application deployment, and your explanation gave me the template on how to do it. I had been relying only on the `if hasattr(sys, 'real_prefix'):` test, which no longer worked. – usonianhorizon Jun 19, 2020 at 16:10

1 @florisla That is what I am asking. I thought `sys.prefix != sys.base_prefix` only suggests, but does not confirm, you are operating inside a virtualenv (or rather, any isolated env)? Basically, is an isolated env the only thing that can change `sys.prefix` to something other than `sys.base_prefix`? That *seems* to be what the docs suggest. But the python path system is zany, so I want to be 100% sure. Also I'm talking `python>=3.3`. – DeusXMachina Oct 18, 2022 at 18:42

29

According to the virtualenv pep at http://www.python.org/dev/peps/pep-0405/#specification you can just use `sys.prefix` instead `os.environ['VIRTUAL_ENV']`.

the `sys.real_prefix` does not exist in my virtualenv and same with `sys.base_prefix`.

Share Improve this answer Follow edited Nov 13, 2020 at 18:40 answered May 24, 2012 at 18:15


Delimitry
3,017 ● 4 ● 31 ● 39


chronossc
435 ● 4 ● 3

8 virtualenv is the standalone project that works on any Python version (github.com/pypa/virtualenv). The PEP you linked to is for pyvenv, which is based on virtualenv but is implemented differently (better) and is built-in to Python 3.3+. This question is about virtualenv, not pyvenv. You're correct that in a pyvenv there is no `sys.real_prefix`. – Carl Meyer May 23, 2013 at 19:18

6 A nice way to detect from bash using this answer is to run: `env |grep VIRTUAL_ENV |wc -l` which will return a 1 if in a venv or a 0 if not. – LISTERINE Nov 4, 2014 at 16:32

4 If you're in a shell you can simply use `[[ -n $VIRTUAL_ENV ]] && echo virtualenv` or `[[ -z $VIRTUAL_ENV ]] && echo not virtualenv` depending on your needs. – Six Dec 29, 2014 at 20:32

15

To check whether your inside Virtualenv:

```
import os

if os.getenv('VIRTUAL_ENV'):
    print('Using Virtualenv')
else:
    print('Not using Virtualenv')
```

You can also get more data on your environment:

```
import sys
import os

print(f'Python Executable: {sys.executable}')
print(f'Python Version: {sys.version}')
print(f'Virtualenv: {os.getenv("VIRTUAL_ENV")}')
```

Share Improve this answer Follow

answered Jul 9, 2018 at 12:04


Matt Harasymczuk
1,758 ● 2 ● 22 ● 29

1 This is the best cross-platform (Windows/Unix) approach. – Adi Unnithan Mar 5, 2019 at 4:48

So far, this is only cross platform, python 2 and python 3 compatible way. Thanks. – R J Sep 19, 2019 at 23:47

Works on Mac too (MacOS Ventura on Mac M1, Python 3.9.10) – Jens Roland Jan 19, 2023 at 19:13

This works ONLY if the venv had been ACTIVATED! If it hasn't it works nowhere. – ullix Dec 30, 2023 at 9:55

9

You can do `which python` and see if its pointing to the one in virtual env.

Share Improve this answer Follow

answered Jul 18, 2018 at 19:47


kunrazor
351 ● 1 ● 4 ● 10

3 `which` is not available by default on Windows. You could use `where` instead on Windows, or employ whichcraft. Or look at `sys.executable`. But still, there are better methods. – florisla Oct 31, 2019 at 7:45

- *Updated Nov 2019 (appended).*

9

I routinely use several Anaconda-installed virtual environments (venv). This code snippet/examples enables you to determine whether or not you are in a venv (or your system environment), and to also require a specific venv for your script.

Add to Python script (code snippet):

```
# ----------------------------------------------------------------------------
# Want script to run in Python 3.5 (has required installed OpenCV, imutils, ...
packages):
import os

# First, see if we are in a conda venv { py27: Python 2.7 | py35: Python 3.5 |
tf: TensorFlow | thee : Theano }
try:
   os.environ["CONDA_DEFAULT_ENV"]
except KeyError:
   print("\tPlease set the py35 { p3 | Python 3.5 } environment!\n")
   exit()

# If we are in a conda venv, require the p3 venv:
if os.environ['CONDA_DEFAULT_ENV'] != "py35":
    print("\tPlease set the py35 { p3 | Python 3.5 } environment!\n")
    exit()

# See also:
# Python: Determine if running inside virtualenv
# http://stackoverflow.com/questions/1871549/python-determine-if-running-inside-
virtualenv
# [ ... SNIP! ... ]
```

Example:

```
$ p2
  [Anaconda Python 2.7 venv (source activate py27)]

(py27) $ python  webcam_.py
    Please set the py35 { p3 | Python 3.5 } environment!

(py27) $ p3
  [Anaconda Python 3.5 venv (source activate py35)]

(py35) $ python  webcam.py -n50
```

```
    current env: py35
    processing (live): found 2 faces and 4 eyes in this frame
    threaded OpenCV implementation
    num_frames: 50
    webcam -- approx. FPS: 18.59
    Found 2 faces and 4 eyes!
(py35) $
```

Update 1 -- use in bash scripts:

You can also use this approach in bash scripts (e.g., those that must run in a specific virtual environment). Example (added to bash script):

```
if [ $CONDA_DEFAULT_ENV ]        ## << note the spaces (important in BASH)!
    then
        printf 'venv: operating in tf-env, proceed ...'
    else
        printf 'Note: must run this script in tf-env venv'
        exit
fi
```

Update 2 [Nov 2019]

- For simplicity, I like Matt's answer (https://stackoverflow.com/a/51245168/1904943).

Since my original post I've moved on from Anaconda venv (and Python itself has evolved *viz-a-viz* virtual environments).

Reexamining this issue, here is some updated Python code that you can insert to test that you are operating in a specific Python virtual environment (venv).

```
import os, re
try:
    if re.search('py37', os.environ['VIRTUAL_ENV']):
        pass
except KeyError:
    print("\n\tPlease set the Python3 venv [alias: p3]!\n")
    exit()
```

Here is some explanatory code.

```
[victoria@victoria ~]$ date; python --version
  Thu 14 Nov 2019 11:27:02 AM PST
  Python 3.8.0

[victoria@victoria ~]$ python
  Python 3.8.0 (default, Oct 23 2019, 18:51:26)
  [GCC 9.2.0] on linux
  Type "help", "copyright", "credits" or "license" for more information.

>>> import os, re

>>> re.search('py37', os.environ['VIRTUAL_ENV'])
<re.Match object; span=(20, 24), match='py37'>

>>> try:
...     if re.search('py37', os.environ['VIRTUAL_ENV']):
```

```
...       print('\n\tOperating in Python3 venv, please proceed!  :-)')
... except KeyError:
...     print("\n\tPlease set the Python3 venv [alias: p3]!\n")
...

    Please set the Python3 venv [alias: p3]!

>>> [Ctrl-d]
  now exiting EditableBufferInteractiveConsole...

[victoria@victoria ~]$ p3
  [Python 3.7 venv (source activate py37)]

(py37) [victoria@victoria ~]$ python --version
  Python 3.8.0

(py37) [victoria@victoria ~]$ env | grep -i virtual
  VIRTUAL_ENV=/home/victoria/venv/py37

(py37) [victoria@victoria ~]$ python
  Python 3.8.0 (default, Oct 23 2019, 18:51:26)
  [GCC 9.2.0] on linux
  Type "help", "copyright", "credits" or "license" for more information.

>>> import os, re
>>> try:
...     if re.search('py37', os.environ['VIRTUAL_ENV']):
...       print('\n\tOperating in Python3 venv, please proceed!  :-)')
... except KeyError:
...     print("\n\tPlease set the Python3 venv [alias: p3]!\n")
...

    Operating in Python3 venv, please proceed!  :-)
>>>
```

Share Improve this answer Follow edited Nov 14, 2019 at 21:44

answered Oct 18, 2016 at 3:35

Victoria Stuart
4,960 ● 2 ● 46 ● 40

6

Easiest way is to just run: `which python`, if you are in a virtualenv it will point to its python instead of the global one

Share Improve this answer Follow

answered Jul 19, 2019 at 9:11

Silvio Biasiol
924 ● 9 ● 14

2 I don't think this actually answers the question (which is concerned about the "current script"). However this answers my particular question, "how do I find out if I'm in a virtual environment from the command line." – ukrutt Sep 16, 2019 at 15:22

You can look for the 'signature' of whatever venv method you're trying to support. In my case I want to support:

2

```
python -m venv venv_dir
```

or

```
virtualenv venv_dir
```

So I can write:

```
import sys
from pathlib import Path

IS_VENV_ENVIRONMENT = (Path(sys.prefix) / "pyvenv.cfg").exists()
```

See: https://docs.python.org/3/library/venv.html

"Running this command creates the target directory (creating any parent directories that don't exist already) and places a pyvenv.cfg file in it"

NB: virtualenv also creates the same file but I couldn't see any docs about that, I only noticed by experiment.

Share Improve this answer Follow edited Feb 20, 2023 at 14:19

answered Feb 20, 2023 at 14:01

Keeely
965 ● 10 ● 22

1

(edited) I found that way, what do you think of it ? (it also returns the venv base path and works even for **readthedocs** where checking the **env** variable does not):

```
import os
import sys
from distutils.sysconfig import get_config_vars


def get_venv_basedir():
    """Returns the base directory of the virtualenv, useful to read configuration
and plugins"""

    exec_prefix = get_config_vars()['exec_prefix']

    if hasattr(sys, 'real_prefix') is False or
exec_prefix.startswith(sys.real_prefix):
        raise EnvironmentError('You must be in a virtual environment')

    return os.path.abspath(get_config_vars()['exec_prefix'] + '/../')
```

Share Improve this answer Follow edited Aug 21, 2017 at 12:10

answered Jul 16, 2017 at 7:26

user8314429

0

I wanted to see if a Python virtual was activated using conda. The following worked for what I was trying to accomplish:

```
import subprocess
result = subprocess.run(["conda", "info", "--envs"], capture_output=True)
output = [res.replace('\n', ' ').strip() for res in result.stdout.decode("utf-
8").split(" ") if res]

store_last = ""
for data in output:
    if '*' in data:
        print(f"Virutal Environment activated: {store_last}")
    store_last = data.split(' ')[-1]
```

Share Improve this answer Follow

answered Apr 22 at 12:50

denezt
11 ●2

-1

In windows OS you see something like this:

```
C:\Users\yourusername\virtualEnvName\Scripts>activate
(virtualEnvName) C:\Users\yourusername\virtualEnvName\Scripts>
```

Parentheses mean that you are actually in the virtual environment called "virtualEnvName".

Share Improve this answer Follow

answered Jul 7, 2017 at 14:58

getleft
73 ●1 ●2 ●9

2 You and I can read 'virtualEnvName' just fine. But the question is, how a Python module can read this. – florisla Sep 19, 2019 at 9:02

-1

There are a lot of great methods posted here already, but just adding one more:

```
import site
site.getsitepackages()
```

tells you where `pip` installed the packages.

Share Improve this answer Follow

answered Aug 12, 2019 at 7:33

flow2k
4,219 ●45 ●58

1 This does not tell if the Python is running inside a virtual environment or not. – florisla Sep 19, 2019 at 9:07

@florisla Could you elaborate? If `site.getsitepackages()` outputs a directory that's not the system one, then you can deduce you're in a virtual environment. – flow2k Sep 19, 2019 at 18:14

3 You can install Python in multiple locations. For example, on Windows you can install a 'system' Python and a WinPython distribution plus a Conda-based Python. These all have different site-packages folders but are not necessarily created by (or used in) a `virtualenv`. – florisla Sep 20, 2019 at 7:05

@florisla Good point - I just saw this (venv or not) is what the question is asking (I had written a similar answer for another question). I agree this might not give the surefire answer as to whether you're in a venv or not, but could help in telling you which Python or which `venv` you're using. – flow2k Sep 26, 2019 at 6:58

-1

If you are using Anaconda here is the solution. This command lists all the discoverable environments

```
conda info --envs
```

Share Improve this answer Follow

answered Dec 18, 2022 at 13:04

Mounesh
678 ● 7 ● 19

-1

I typically use a bash wrapper script to start my application and was running into the problem of imports not working because of pip packages not being installed outside the virtual environment.

Using the solution provided by Matt Harasymczuk, here are the few lines of bash script to check the virtual environment has been activated before starting the application.

```
#!/bin/bash

# check the virtual environment is active
if [ -z "$VIRTUAL_ENV" ]
then
    echo "[ERROR] Virtual environment not activated"
    exit 1
fi

# start the application
python3   # amend with application name and other details

# end of file
```

Share Improve this answer Follow

answered Feb 16, 2023 at 5:13

Ramon
97 ● 1 ● 3

-1

As not already mentioned, and time has moved on since this question was made, I thought I'd mention pyenv-virtualenv.

pyenv allows multiple python interpreters to exist on the same system at the same time without collision. The interpreter can automatically change when you chdir into a given path. https://github.com/pyenv/pyenv

pyenv-virtualenv is a pyenv plugin that, like venv/pipenv, creates independent virtualenvs each with different module configurations, but now that env also specifies which interpreter(s) can be used. https://github.com/pyenv/pyenv-virtualenv

If you are using pyenv or pyenv-virtualenv then this one line bash command will work.

```
test -x $(which pyenv) && { pyenv which python | grep -q "^$(pyenv root)"; } &&
echo "1"
```

This could be used to set an environment variable that Python can read via one of the hook scripts that pyenv uses, or updated via the shell prompt hook variable, $PROMPT_COMMAND. (Note: shell hook example may need some speedup shortcuts/tricks or risk being annoying)

e.g. Here IN_VENV is either 'system' or a version number or a virtualenv

```
export IN_VENV=$(test -x $(which pyenv) && { pyenv which python | grep -q
"^$(pyenv root)"; } && pyenv version-name)
```

If you need to detect this inside Python with no help from the os, then

```
from subprocess import run

# run cmd in shell
# return None if error else stdout or ""
def sh(cmd, **kw):
    res = run(cmd, capture_output=True, shell=True, **kw)
    return None if res.returncode != 0 else res.stdout.decode("ascii").strip()

pyenv = sh("which pyenv")
assert pyenv is not None, "No pyenv command"
assert sh(f"test -x {pyenv}") is not None, "pyenv is non-exe"
pycmd = sh(f"pyenv which python")
assert pycmd is not None, "no pyenv python"
root = sh(f"{pyenv} root")
assert root is not None, "'pyenv root' did not run"
in_pyenv = sh(f"grep -q ^{root}", input=pycmd.encode('utf-8') is not None
```

This not only finds the virtualenv but proves it is working.

Share Improve this answer Follow edited Mar 19, 2023 at 3:48

answered Mar 19, 2023 at 3:38

Jay M
4,084 ● 1 ● 26 ● 38

-1

If you want to directly check if you are in a virtualenv the above answers are the correct solutions.

However, if the real problem you are trying to solve is determining whether you can `pip install` a dependency without erroring (something that would be true if you were in a virtualenv), you can simply check if you have write permissions for the location of the Python executable:

```
os.access(sys.executable, os.W_OK)
```

This works in all versions of Python, and also returns `True` if you run the system Python with `sudo`. Here's an example of how this would be useful:

```
import os, sys
can_install_pip_packages = os.access(sys.executable, os.W_OK)

if can_install_pip_packages:
    import pip
    pip.main(['install', 'mypackage'])
```

Share Improve this answer Follow edited May 31, 2023 at 4:52

answered May 2, 2018 at 15:43

Matthew D. Scholefield
3,166 ● 3 ● 33 ● 44

-2

It's not bullet-proof but for UNIX environments simple test like

```
if run("which python3").find("venv") == -1:
    # something when not executed from venv
```

works great for me. It's simpler then testing existing of some attribute and, anyway, you should name your venv directory `venv`.

Share Improve this answer Follow

answered Oct 18, 2016 at 13:40

Matt Warrick
1,478 ● 1 ● 14 ● 23

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.