Contents

[Python Ternary conditional operator 1](#_Toc505591388)

[How to RUN ansible module in Python 2](#_Toc505591389)

[Check if List is Empty 3](#_Toc505591390)

[Other methods don't work for numpy arrays 3](#_Toc505591391)

[Check if Word is in string 5](#_Toc505591392)

[String.join(list) on object array rather than string array 5](#_Toc505591393)

[Turn List value to string 6](#_Toc505591394)

[Exit without Exception Raised from sys.exit() 6](#_Toc505591395)

[Check if key in list 7](#_Toc505591396)

[Formatting Pyaml dump 8](#_Toc505591397)

[Creating a new Dictionary 9](#_Toc505591398)

[Editing YAML file 9](#_Toc505591399)

[Deleting item from list in loop 9](#_Toc505591400)

[Flatten a array 10](#_Toc505591401)

[Check Empty String 11](#_Toc505591402)

[TypeError: a float is required in time.sleep() 12](#_Toc505591403)

# Python Ternary conditional operator

Yes, it was [added](https://mail.python.org/pipermail/python-dev/2005-September/056846.html) in version 2.5.  
The syntax is:

a if condition else b

First condition is evaluated, then either a or b is returned based on the [Boolean](https://en.wikipedia.org/wiki/Boolean_data_type) value of condition  
If condition evaluates to *True* a is returned, else b is returned.

For example:

>>> 'true' if True else 'false'

'true'

>>> 'true' if False else 'false'

'false'

Keep in mind that it's frowned upon by some Pythonistas for several reasons:

* The order of the arguments is different from many other languages (such as C, Ruby, Java, etc.), which may lead to bugs when people unfamiliar with Python's "surprising" behaviour use it (they may reverse the order).
* Some find it "unwieldy", since it goes contrary to the normal flow of thought (thinking of the condition first and then the effects).
* Stylistic reasons.

If you're having trouble remembering the order, then remember that if you read it out loud, you (almost) say what you mean. For example, x = 4 if b > 8 else 9 is read aloud as x will be 4 if b is greater than 8 otherwise 9.

You can index into a tuple:

(falseValue, trueValue)[test]

test needs to return *True* or *False*.  
It might be safer to always implement it as:

(falseValue, trueValue)[test == True]

or you can use the built-in [bool()](https://docs.python.org/3.3/library/functions.html#bool) to assure a [Boolean](https://en.wikipedia.org/wiki/Boolean_data_type) value:

(falseValue, trueValue)[bool(<expression>)]

# How to RUN ansible module in Python

Here is my code: I am not able to run a ansible module using python. How to pass a inventory file for which this command is running. I am not able to run it for my inventory. Do I need to do something else ? Here is my ansible command:

ansible all -i /home/ubuntu/extra -m 'debug' -a 'var=hostvars'

Here is my code:

import json

import ansible.runner

import ansible.playbook

import ansible.inventory

example\_inventory = ansible.inventory.Inventory('path/to/your/inventory')

pm = ansible.runner.Runner( module\_name = 'debug', module\_args = 'var=hostvars', timeout = 5, inventory = example\_inventory, subset = 'all')

out = pm.run()

print json.dumps(out, sort\_keys=True, indent=4, separators=(',', ': '))

|  |  |
| --- | --- |
| You can pass the inventory file path to ansible.runner.Runner()  And for getting group-names and host-names , you should pass var=hostvars, not vars=hostvars  Your code would look like this,  import json  import ansible.runner  import ansible.playbook  import ansible.inventory  example\_inventory = ansible.inventory.Inventory('path/to/your/inventory')  pm = ansible.runner.Runner( module\_name = 'debug', module\_args = 'var=hostvars', timeout = 5, inventory = example\_inventory, subset = 'all')  out = pm.run()  print json.dumps(out, sort\_keys=True, indent=4, separators=(',', ': '))  and your output  {'contacted': {'ip-address': {'invocation': {'module\_args': u'var=hostvars',  'module\_complex\_args': {},  'module\_name': 'debug'},  'var': {u'hostvars': {'group\_names': ['group1', 'group2', 'group3'],  'groups': {'group1': ['ip-address'],  'all': ['ip-address'],  'group2': ['ip-address'],  'group3': ['ip-address'],  'ungrouped': []},  'inventory\_hostname': 'ip/hostname',  'inventory\_hostname\_short': 'hostname-short'}},  'verbose\_always': True}},  'dark': {}}   |  | | --- | | [share](https://stackoverflow.com/a/33543150/7579266)[edit](https://stackoverflow.com/posts/33543150/edit)[flag](https://stackoverflow.com/questions/33538483/how-to-run-ansible-module-in-python) | |

# Check if List is Empty

if not a:

print("List is empty")

Using the implicit booleanness of the empty list is quite pythonic.

I prefer it explicitly:

if len(li) == 0:

print('the list is empty')

This way it's 100% clear that li is a sequence (list) and we want to test its size. My problem with if not li: ... is that it gives the false impression that li is a boolean variable.

# Other methods don't work for numpy arrays

You need to be careful with numpy arrays, because other methods that work fine for lists or other standard containers fail for numpy arrays. I explain why below, but in short, the [preferred method](http://www.scipy.org/scipylib/faq.html#what-is-the-preferred-way-to-check-for-an-empty-zero-element-array) is to use size.

**The "pythonic" way doesn't work: Part 1**

The "pythonic" way fails with numpy arrays because numpy tries to cast the array to an array of bools, and if x tries to evaluate all of those bools at once for some kind of aggregate truth value. But this doesn't make any sense, so you get a ValueError:

>>> x = numpy.array([0,1])

>>> if x: print("x")

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()

**The "pythonic" way doesn't work: Part 2**

But at least the case above tells you that it failed. If you happen to have a numpy array with exactly one element, the if statement will "work", in the sense that you don't get an error. However, if that one element happens to be 0 (or 0.0, or false, ...), the if statement will incorrectly result in false:

>>> x = numpy.array([0,])

>>> if x: print("x")

... else: print("No x")

No x

But clearly x exists and is not empty! This result is not what you wanted.

**Using len can give unexpected results**

For example,

len( numpy.zeros((1,0)) )

returns 1, even though the array has zero elements.

**The numpythonic way**

As explained in the [scipy FAQ](http://www.scipy.org/scipylib/faq.html" \l "what-is-the-preferred-way-to-check-for-an-empty-zero-element-array), the correct method in all cases where you know you have a numpy array is to use if x.size:

>>> x = numpy.array([0,1])

>>> if x.size: print("x")

x

>>> x = numpy.array([0,])

>>> if x.size: print("x")

... else: print("No x")

x

>>> x = numpy.zeros((1,0))

>>> if x.size: print("x")

... else: print("No x")

No x

If you're not sure whether it might be a list, a numpy array, or something else, you could combine this approach with [the answer @dubiousjim gives](https://stackoverflow.com/a/10835703/1194883) to make sure the right test is used for each type. Not very "pythonic", but it turns out that numpy intentionally broke pythonicity in at least this sense.

If you need to do more than just check if the input is empty, and you're using other numpy features like indexing or math operations, it's probably more efficient (and certainly more common) to force the input to be a numpy array. There are a few nice functions for doing this quickly — most importantly [numpy.asarray](https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.asarray.html). This takes your input, does nothing if it's already an array, or wraps your input into an array if it's a list, tuple, etc., and optionally converts it to your chosen dtype. So it's very quick whenever it can be, and it ensures that you just get to assume the input is a numpy array. We usually even just use the same name, as the conversion to an array won't make it back outside of the current [scope](http://python-textbok.readthedocs.io/en/1.0/Variables_and_Scope.html):

x = numpy.asarray(x, dtype=numpy.double)

This will make the x.size check work in all cases I see on this page.

# Check if Word is in string

if string.find(word):

print 'success'

if word in mystring:

print 'success'

Thanks for any help.

import re

def findWholeWord(w):

return re.compile(r'\b({0})\b'.format(w), flags=re.IGNORECASE).search

findWholeWord('seek')('those who seek shall find') # -> <match object>

findWholeWord('word')('swordsmith')

haystack = 'asdf'

haystack.find('a') # result: 0

haystack.find('s') # result: 1

haystack.find('g') # result: -1

if haystack.find(needle) >= 0:

print 'Needle found.'

else:

print 'Needle not found.'

# String.join(list) on object array rather than string array

In Python, I can do:

>>> list = ['a', 'b', 'c']

>>> ', '.join(list)

'a, b, c'

Is there any easy way to do the same when I have a list of objects?

>>> class Obj:

... def \_\_str\_\_(self):

... return 'name'

...

>>> list = [Obj(), Obj(), Obj()]

>>> ', '.join(list)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: sequence item 0: expected string, instance found

Or do I have to resort to a for loop?

ou could use a list comprehension or a generator expression instead:

', '.join([str(x) for x in list]) # list comprehension

', '.join(str(x) for x in list) # generator expression

The built-in string constructor will automatically call obj.\_\_str\_\_:

''.join(map(str,list))

Use [join](http://docs.python.org/library/stdtypes.html#str.join):

>>> sentence = ['this','is','a','sentence']

>>> '-'.join(sentence)

'this-is-a-sentence'

>>> my\_lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> my\_lst\_str = ''.join(map(str, my\_lst))

>>> print(my\_lst\_str)

'12345678910'

# Turn List value to string

Apply str() on the element not on the list:

In [206]: country = [u'USA']

In [207]: country[0] = str(country[0])

In [208]: country

Out[208]: ['USA']

or may be you meant this:

In [217]: country = [u'USA']

In [218]: country = str(country[0])

In [219]: country

Out[219]: 'USA'

# Exit without Exception Raised from sys.exit()

You can call [os.\_exit()](https://docs.python.org/2/library/os.html" \l "os._exit) to directly exit, without throwing an exception:

import os

os.\_exit(1)

As Jerub said, os.\_exit(1) is your answer. But, considering it bypasses all cleanup procedures, including finally: blocks, closing files, etc, and should really be avoided at all costs, may I present a "safer-ish" way of using it?

If you problem is SystemExit being caught at outer levels (ie, unittest), then **be the outer level yourself!** Wrap your main code in a try/except block, catch SystemExit, and call os.\_exit there, and ***only*** there! This way you may call sys.exit normally anywhere in the code, let it bubble out to the top level, gracefully closing all files and running all cleanups, and then calling os.\_exit.

You can even choose which exits are the "emergency" ones. The code below is an example of such approach:

import sys, os

EMERGENCY = 255 # can be any number actually

try:

# wrap your whole code here ...

# ... some code

if x: sys.exit()

# ... some more code

if y: sys.exit(EMERGENCY) # use only for emergency exits

# ...

except SystemExit as e:

if e.code != EMERGENCY:

raise # normal exit, let unittest catch it

else:

os.\_exit(EMERGENCY) # try to stop \*that\*, sucker!

# Check if key in list

in is definitely more pythonic.

In fact [has\_key() was removed in Python 3.x](http://docs.python.org/3.1/whatsnew/3.0.html" \l "builtins).

According to python [docs](http://docs.python.org/library/stdtypes.html#dict.has_key):

has\_key() is deprecated in favor of key in d.

ages = {}

# Add a couple of names to the dictionary

ages['Sue'] = 23

ages['Peter'] = 19

ages['Andrew'] = 78

ages['Karren'] = 45

# use of 'in' in if condition instead of function\_name.has\_key(key-name).

if 'Sue' in ages:

print "Sue is in the dictionary. She is", ages['Sue'], "years old"

else:

print "Sue is not in the dictionary"

in is the intended way to test for the existence of a key in a dict.

d = dict()

for i in xrange(100):

key = i % 10

if key in d:

d[key] += 1

else:

d[key] = 1

If you wanted a default, you can always use dict.get():

d = dict()

for i in xrange(100):

key = i % 10

d[key] = d.get(key, 0) + 1

... and if you wanted to always ensure a default value for any key you can use defaultdict from the collections module, like so:

from collections import defaultdict

d = defaultdict(lambda: 0)

for i in xrange(100):

d[i % 10] += 1

... but in general, the in keyword is the best way to do it.

# Formatting Pyaml dump

There's no easy way to do this with the library (Node objects in yaml dumper syntax tree are passive and can't emit this info), so I ended up with

stream = yaml.dump(list\_of\_dicts, default\_flow\_style = False)

file.write(stream.replace('\n- ', '\n\n- '))

Before anything else: **never use yaml.load() if you don't have to**, as it is in principle unsafe to do so. For this kind of simple structure (without tags) you should use yaml.safe\_load() (and the corresponding safe\_dump() that will complain if your data cannot be safe\_loaded after dumping).

yaml.dump() has the following signature:

def dump(documents, stream=None, Dumper=Dumper,

default\_style=None, default\_flow\_style=None,

canonical=None, indent=None, width=None,

allow\_unicode=None, line\_break=None,

encoding='utf-8', explicit\_start=None, explicit\_end=None,

version=None, tags=None)

Of this only the first one needs to be given, that should be your doc variable. If you don't specify a stream, then dump() writes data structure to an in memory file-object (like StringIO) and after writing returns the value as string.

So although you could do:

with open("file\_to\_edit.yaml", 'w') as f:

f.write(yaml.safe\_dump(doc))

this is inefficient and shows little understanding of how yaml.safe\_dump() works.

If you want to open the file for reading and writing you have to make sure you both reset the index in the file **and** truncate its content. This is usually not worth the effort so it is safer to re-open the file for writing:

def set\_state(state):

file\_name = "file\_to\_edit.yaml"

with open(file\_name) as f:

doc = yaml.safe\_load(f)

doc['state'] = state

with open(file\_name, 'w') as f:

yaml.safe\_dump(doc, f, default\_flow\_style=False)

(of course you make the filename a variable when you want to make sure you overwrite the original, so you cannot mistype it).

If you don't specify default\_flow\_style=False, your output will look like:

{state: deleted}

The output will not include the superfluous quotes around present in your input. You can specify default\_style="'" as well, but this will also put quotes around state.  
If losing the quotes is a problem and you really want the output to look like the input, you should be using ruamel.yaml (disclaimer I am the author of that package), which can preserve the quotes on individual strings, handles YAML 1.2 (instead of YAML 1.1) and also preserves comments in your file.

# Creating a new Dictionary

Call dict with no parameters

new\_dict = dict()

or simply write

new\_dict = {}

d = dict()

or

d = {}

or

import types

d = types.DictType.\_\_new\_\_(types.DictType, (), {})

# Editing YAML file

with open("my\_file.yaml") as f:

list\_doc = yaml.load(f)

for sense in list\_doc:

if sense["name"] == "sense\_2":

sense["value"] = 1234

with open("my\_file.yaml", "w") as f:

yaml.dump(list\_doc, f)

# Deleting item from list in loop

I have checked all of the other questions with the same error yet found no helpful solution =/

I have a dictionary of lists:

d = {'a': [1], 'b': [1, 2], 'c': [], 'd':[]}

in which some of the values are empty. At the end of creating these lists, I want to remove these empty lists before returning my dictionary. Current I am attempting to do this as follows:

for i in d:

if not d[i]:

d.pop(i)

however, this is giving me the runtime error. I am aware that you cannot add/remove elements in a dictionary while iterating through it...what would be a way around this then?

In Python 2.x calling keys makes a copy of the key that you can iterate over while modifying the dict:

for i in d.keys():

Note that this doesn't work in Python 3.x because keys returns an iterator instead of a list.

Another way is to use list to force a copy of the keys to be made. This one also works in Python 3.x:

for i in list(d):

# Flatten a array

|  |
| --- |
|  |
|  | # http://stackoverflow.com/questions/11264684/flatten-list-of-lists |
|  | list\_of\_lists = [[180.0], [173.8], [164.2], [156.5], [147.2], [138.2]] |
|  | flattened = [val for sublist in list\_of\_lists for val in sublist] |

**import** unittest

**from** flatten\_array **import** flatten

*# Tests adapted from `problem-specifications//canonical-data.json` @ v1.1.0*

**class** **FlattenArrayTests**(unittest**.**TestCase):

**def** **test\_no\_nesting**(self):

self**.**assertEqual(flatten([0, 1, 2]), [0, 1, 2])

**def** **test\_flatten\_integers**(self):

inputs **=** [1, [2, 3, 4, 5, 6, 7], 8]

expected **=** [1, 2, 3, 4, 5, 6, 7, 8]

self**.**assertEqual(flatten(inputs), expected)

**def** **test\_five\_level\_nesting**(self):

inputs **=** [0, 2, [[2, 3], 8, 100, 4, [[[50]]]], **-**2]

expected **=** [0, 2, 2, 3, 8, 100, 4, 50, **-**2]

self**.**assertEqual(flatten(inputs), expected)

**def** **test\_six\_level\_nesting**(self):

inputs **=** [1, [2, [[3]], [4, [[5]]], 6, 7], 8]

expected **=** [1, 2, 3, 4, 5, 6, 7, 8]

self**.**assertEqual(flatten(inputs), expected)

**def** **test\_with\_none\_values**(self):

inputs **=** [0, 2, [[2, 3], 8, [[100]], None, [[None]]], **-**2]

expected **=** [0, 2, 2, 3, 8, 100, **-**2]

self**.**assertEqual(flatten(inputs), expected)

**def** **test\_all\_values\_are\_none**(self):

inputs **=** [None, [[[None]]], None, None, [[None, None], None], None]

expected **=** []

self**.**assertEqual(flatten(inputs), expected)

*# Additional tests for this track*

**def** **test\_empty\_nested\_lists**(self):

self**.**assertEqual(flatten([[()]]), [])

**def** **test\_strings**(self):

self**.**assertEqual(flatten(['0', ['1', '2']]), ['0', '1', '2'])

**if** \_\_name\_\_ **==** '\_\_main\_\_':

unittest**.**main()

# Check Empty String

From [PEP 8](http://www.python.org/dev/peps/pep-0008/), in the [“Programming Recommendations” section](https://www.python.org/dev/peps/pep-0008/#programming-recommendations):

For sequences, (strings, lists, tuples), use the fact that empty sequences are false.

So you should use:

if not some\_string:

or:

if some\_string:

I would test noneness before stripping. Also, I would use the fact that empty strings are False (or Falsy). This approach is similar to [Apache's StringUtils.isBlank](https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html#isBlank-java.lang.CharSequence-) or [Guava's Strings.isNullOrEmpty](https://google.github.io/guava/releases/19.0/api/docs/com/google/common/base/Strings.html#isNullOrEmpty(java.lang.String))

**This is what I would use to test if a string is either None OR Empty OR Blank:**

def isBlank (myString):

if myString and myString.strip():

#myString is not None AND myString is not empty or blank

return False

#myString is None OR myString is empty or blank

return True

**And, the exact opposite to test if a string is not None NOR Empty NOR Blank:**

def isNotBlank (myString):

if myString and myString.strip():

#myString is not None AND myString is not empty or blank

return True

#myString is None OR myString is empty or blank

return False

**More concise forms of the above code:**

def isBlank (myString):

return not (myString and myString.strip())

def isNotBlank (myString):

return bool(myString and myString.strip())

I once wrote something similar to Bartek's answer and javascript inspired:

def isNotEmpty(s):

return bool(s and s.strip())

Test:

print isNotEmpty("") # False

print isNotEmpty(" ") # False

print isNotEmpty("ok") # True

print isNotEmpty(None) # False

Test empty or blank string (shorter way):

if myString.strip():

print("it's not an empty or blank string")

else:

print("it's an empty or blank string")

a = ''

b = ' '

a.isspace() -> False

b.isspace() -> True

# [TypeError: a float is required in time.sleep()](https://stackoverflow.com/questions/27223653/typeerror-a-float-is-required-in-time-sleep)

print "Delay entered until next cluster:",

print delay.rstrip("\n"),

print "seconds\n"

delay = delay.rstrip('\n')

time.sleep(delay)

Everything works absolutely fine! Except for that last part, every time I run it, it gives me the following error:

sleep() needs a float and you're giving it a string. You need an explicit conversion:

time.sleep(float(delay))