What is the use of “assert” in Python?

Ask Question

Asked 9 years, 5 months ago Active 1 month ago Viewed 754k times

989

I have been reading some source code and in several places I have seen the usage of `assert`.

What does it mean exactly? What is its usage?

python assert assertions

268

share improve this question follow

edited Dec 3 '14 at 3:13

APerson

6,946 ● 4 ● 30 ● 47

asked Feb 28 '11 at 13:11

Hossein

30.2k ● 52 ● 122 ● 167

3 stackoverflow.com/questions/944592/… – Russell Dias Feb 28 '11 at 13:13

add a comment

21 Answers

Active | Oldest | Votes

1123

The `assert` statement exists in almost every programming language. It helps detect problems early in your program, where the cause is clear, rather than later as a side-effect of some other operation.

When you do...

```
assert condition
```

... you're telling the program to test that condition, and immediately trigger an error if the condition is false.

In Python, it's roughly equivalent to this:

```
if not condition:
    raise AssertionError()
```

Try it in the Python shell:

```
>>> assert True # nothing happens
>>> assert False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

Assertions can include an optional message, and you can disable them when running the interpreter.

To print a message if the assertion fails:

```
assert False, "Oh no! This assertion failed!"
```

Do **not** use parenthesis to call `assert` like a function. It is a statement. If you do `assert(condition, message)` you'll be running the `assert` with a `(condition, message)` tuple as first parameter.

As for disabling them, when running `python` in optimized mode, where `__debug__` is `False`, assert statements will be ignored. Just pass the `-O` flag:

```
python -O script.py
```

See here for the relevant documentation.

share improve this answer follow

edited Apr 19 '19 at 15:14

answered Feb 28 '11 at 13:15

slezica

58.1k ● 18 ● 85 ● 148

95 Nit: assert is a statement and not a function. And unlike print, in Python 3 it's still a statement. — Rob Stein Sep 16 '14 at

95 Nit: assert is a statement and not a function. And unlike print, in Python 3 it's still a statement. – Bob Stein Sep 16 '14 at 16:45

2 @Chaine assert means "make sure that *something" is True". So assert a == 3 will make sure that a is equal to 3; if a is not equal to 3 (i.e. a==3 is False) then it will raise an error – Ant Jul 18 '17 at 14:55

6 If I can just use the `if not condition: raise AssertError()`, why should I use assert? Are there any conditions under which assert is better other than just being a shorter form of `if not condition` statement? – alpha_989 Jan 14 '18 at 17:33

6 @alpha_989 a) it's shorter and more readable, b) you can disable assert statements when running the interpreter (not so with the manual `if`). Read the docs for more info :) – slezica Jan 17 '18 at 15:07

10 totally cannot get how does this answer get so many up votes, actually others answers also. the question is "What is the use of “assert” in Python? ", so it is asking: when to use, or more exactly: what is the usage scenario of `assert`, but after reading all answers, i totally got nothing i want! – lnshi Feb 6 '18 at 3:10

show **2** more comments

438

Watch out for the parentheses. As has been pointed out above, in Python 3, `assert` is still a statement, so by analogy with `print(..)`, one may extrapolate the same to `assert(..)` or `raise(..)` but you shouldn't.

This is important because:

```
assert(2 + 2 == 5, "Houston we've got a problem")
```

won't work, unlike

```
assert 2 + 2 == 5, "Houston we've got a problem"
```

The reason the first one will not work is that `bool( (False, "Houston we've got a problem") )` evaluates to `True`.

In the statement `assert(False)`, these are just redundant parentheses around `False`, which evaluate to their contents. But with `assert(False,)` the parentheses are now a tuple, and a non-empty tuple evaluates to `True` in a boolean context.

answered Jun 11 '15 at 2:15

share improve this answer follow

edited Apr 6 '16 at 19:31

kmario23

33.7k ● 9 ● 105 ● 116

answered Jun 11 '15 at 2:15

Evgeni Sergeev

16.9k ● 15 ● 89 ● 103

19 I came here looking for this exact info about parens and the follow message. Thanks. – superbeck Jul 11 '16 at 18:41

6 But `assert (2 + 2 = 5), "Houston we've got a problem"` should be ok, yes? – SherylHohman Apr 30 '17 at 22:30

4 @SherylHohman you can also try to run that yourself and see if it works or not – DarkCygnus May 8 '17 at 22:30

3 Don't forget that people often use parentheses for PEP 8-compliant implicit line continuation Also Also don't forget that tuples are not defined by parentheses but by the existence of the comma (tuples have nothing to do with parens except for the purposes of operator precedence). – cowbert Aug 27 '17 at 20:42

4 `assert (2 + 2 = 5), "Houston we've got a problem"` won't work... but it has nothing to do with the assert statement, which is fine. Your condition won't work because it isn't a condition. Missing a second `=`. – n1k31t4 Oct 20 '17 at 23:50

show **5** more comments

137

As other answers have noted, `assert` is similar to throwing an exception if a given condition isn't true. An important difference is that assert statements get ignored if you compile your code with the optimization option `-O`. The documentation says that `assert expression` can better be described as being equivalent to

```
if __debug__:
   if not expression: raise AssertionError
```

This can be useful if you want to thoroughly test your code, then release an optimized version when you're happy that none of your assertion cases fail - when optimization is on, the `__debug__` variable becomes False and the conditions will stop getting evaluated. This feature can also catch you out if you're relying on the asserts and don't realize they've disappeared.

share improve this answer follow

edited Dec 27 '19 at 3:59

Ricardo Stuven

4,179 ● 2 ● 32 ● 36

answered Feb 28 '11 at 14:10

Neil Vass

4,253 ● 2 ● 18 ● 25

Does this mean, that if a certain variable or correct input (according to the contract by which the program is written) could lead to crashing the program, when its run by the user (assuming that -O flag is used when the user runs the program), you should instead use the `if Not Error: raise Exception(" this is a error")`? That way, the program will still show the source of the error, when the user runs it.. – alpha_989 Jan 14 '18 at 19:36

On the other hand, if you expect that the program could error out because of incorrect logic/implementation of the code (but not due to an input which is according to the contract to the user of the program), you should use the `assert` statement? The assumption here is that when the program is released to the end user, you are using the -O flag, thus assuming that all the bugs have been removed. Hence, any error or program crash is due to input to the program which is valid as per the contract, but cant be handled by the program. So it should alert the user as such. – alpha_989 Jan 14 '18 at 19:36

@alpha_989 that's exactly right. I like to think of assertions as sanity checks that are only to help you as a developer to make sure that what you think is true is actually true while you develop. – Christopher Shroba Mar 20 '18 at 18:32

add a comment

55

The goal of an assertion in Python is to inform developers about **unrecoverable** errors in a program.

Assertions are not intended to signal expected error conditions, like “file not found”, where a user can take corrective action (or just try again).

Another way to look at it is to say that assertions are **internal self-checks** in your code. They work by declaring some conditions as *impossible* in your code. If these conditions don’t hold that means there’s a bug in the program.

If your program is bug-free, these conditions will never occur. But if one of them *does* occur the program will crash with an assertion error telling you exactly which “impossible” condition was triggered. This makes it much easier to track down and fix bugs in your programs.

Here’s a summary from a tutorial on Python’s assertions I wrote:

> **Python’s assert statement is a debugging aid, not a mechanism for handling run-time errors.** The goal of using assertions is to let developers find the likely root cause of a bug more quickly. An assertion error should never be raised unless there’s a bug in your program.

share improve this answer follow

answered Jan 18 '17 at 14:04

dbader

7,207 ● 1 ● 18 ● 17

Thanks for the article. Very helpful to understand `assert` statement and when to use this. I am trying to understand a number of terms that you introduced in the article. – alpha_989 Jan 14 '18 at 18:51

I thought I would post the comments here so a lot more people might be benefited from the clarifications. Sorry if the questions are too naive. – alpha_989 Jan 14 '18 at 18:51

In your blog that you linked, you give an example where you mentioned that ` assert 0 <= price <= product['price']` is correct, but using ` assert user.is_admin(), 'Must have admin privileges to delete'` and `assert store.product_exists(product_id), 'Unknown product id'` is not a good practice, because if the debug is turned off then the `user` even if not an `admin` will be able to delete the product. Do you consider `assert user.is_admin()` as a `unrecoverable` error? Why is this not a `self-check`? – alpha_989 Jan 14 '18 at 18:54

If you consider that 'user.is_admin()` is a user input and hence shouldn't be used in an `assert statement`, cant `price` also be considered a user input? Why do you consider `assert user.is_admin()` as data validation but not `assert price`? – alpha_989 Jan 14 '18 at 18:54

1 @LaryxDecidua Nope, you can just read it on my website, the tutorial is publicly available. Just hit escape or click the little "x" symbol if you're not interested in the newsletter. Hope this helps :-) – dbader Apr 14 '18 at 20:36

show **1** more comment

52

Others have already given you links to documentation.

You can try the following in a interactive shell:

```
>>> assert 5 > 2
>>> assert 2 > 5
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.AssertionError:
```

The first statement does nothing, while the second raises an exception. This is the first hint: asserts are useful to check conditions that should be true in a given position of your code (usually, the beginning (preconditions) and the end of a function (postconditions)).

Asserts are actually highly tied to programming by contract, which is a very useful engineering practice:

http://en.wikipedia.org/wiki/Design_by_contract.

share improve this answer follow

edited Sep 19 '17 at 18:53

pyrrhic
1,329 ● 2 ● 13 ● 25

answered Feb 28 '11 at 13:18

Baltasarq
10.6k ● 1 ● 32 ● 51

So does that mean we can check in code in a situation like assert(2 > 5) and raise error else continue ? – user1176501 Oct 17 '13 at 6:25

21 Lose the parens, assert is not a function. – pillmuncher Feb 17 '14 at 15:27

2 Losing the parens is more important than it seems. See below. – Evgeni Sergeev Jul 16 '15 at 11:50

6 Assert actually dates back (long before "contracts") to Turing, when he wrote one of the earliest papers on how programmers might tackle the rather daunting task of creating correct programs. Finding that paper is left as an exercise for the reader, since all programmers can benefit from becoming familiar with his work. :-) turingarchive.org – Ron Burk Oct 31 '16 at 23:41

add a comment

19

From docs:

> Assert statements are a convenient way to insert debugging assertions into a program

You can read more here: http://docs.python.org/release/2.5.2/ref/assert.html

share improve this answer follow

edited Jun 17 at 10:51

binaryfunt
4,016 ● 2 ● 22 ● 40

answered Feb 28 '11 at 13:16

gruszczy
35.6k ● 26 ● 111 ● 163

i like this comment since it just explains what it is very plainly. my question is "if i've written a proper unit test, why would i need an assertion"? that stuff doesn't run in production anyways. – dtc Feb 10 at 7:06

add a comment

The assert statement has two forms.

18

The simple form, `assert <expression>`, is equivalent to

```
if __debug__:
    if not <expression>: raise AssertionError
```

The extended form, `assert <expression1>, <expression2>`, is equivalent to

```
if __debug__:
    if not <expression1>: raise AssertionError, <expression2>
```

share improve this answer follow

edited Oct 15 '14 at 20:54

Colin D Bennett

7,885 ● 3 ● 37 ● 59

answered Jul 10 '13 at 1:21

Bohdan

12.5k ● 11 ● 65 ● 64

add a comment

17

Assertions are a systematic way to check that the internal state of a program is as the programmer expected, with the goal of catching bugs. See the example below.

```
>>> number = input('Enter a positive number:')
Enter a positive number:-1
>>> assert (number > 0), 'Only positive numbers are allowed!'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: Only positive numbers are allowed!
>>>
```

share improve this answer follow

edited Feb 19 '14 at 16:58

answered Feb 19 '14 at 16:52

Jacob Abraham

833 ● 9 ● 8

1 Also, assertions can often be used in unit testing programs. stackoverflow.com/questions/1383/what-is-unit-testing – panofish Oct 2 '14 at 18:29

add a comment

8

Here is a simple example, save this in file (let's say b.py)

```
def chkassert(num):
    assert type(num) == int


chkassert('a')
```

and the result when `$python b.py`

```
Traceback (most recent call last):
  File "b.py", line 5, in <module>
    chkassert('a')
  File "b.py", line 2, in chkassert
    assert type(num) == int
AssertionError
```

share improve this answer follow

answered Feb 17 '14 at 14:54

Gaurav Agarwal

16.4k ● 27 ● 94 ● 152

add a comment

7

if the statement after assert is true then the program continues , but if the statement after assert is false then the program gives an error. Simple as that.

e.g.:

```
assert 1>0   #normal execution
assert 0>1   #Traceback (most recent call last):
             #File "<pyshell#11>", line 1, in <module>
             #assert 0>1
             #AssertionError
```

share improve this answer follow

edited Jun 29 '17 at 21:52

Shawn Mehan

4,026 ● 9 ● 25 ● 46

answered Nov 1 '14 at 5:05

abe312

3,781 ● 28 ● 21

add a comment

5

The `assert` statement exists in almost every programming language. It helps detect problems early in your program, where the cause is clear, rather than later as a side-effect of some other operation. They always expect a `True` condition.

When you do something like:

```
assert condition
```

You're telling the program to test that condition and immediately trigger an error if it is false.

In Python, `assert` expression, is equivalent to:

```
if __debug__:
    if not <expression>: raise AssertionError
```

You can use the extended expression to pass an **optional message**:

```
if __debug__:
    if not (expression_1): raise AssertionError(expression_2)
```

Try it in the Python interpreter:

```
>>> assert True # Nothing happens because the condition returns a True value.
>>> assert False # A traceback is triggered because this evaluation did not yield an expected value
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

There are some caveats to seen before using them mainly for those who deem to toggles between the `assert` and `if` statements. The aim to use `assert` is on occasions when the program verifies a condition and return a value that should stop the program immediately instead of taking some alternative way to bypass the error:

1. Parentheses

As you may have noticed, the `assert` statement uses two conditions. Hence, do **not** use parentheses to englobe them as one for obvious advice. If you do such as:

```
assert (condition, message)
```

Example:

```
>>> assert (1==2, 1==1)
<stdin>:1: SyntaxWarning: assertion is always true, perhaps remove parentheses?
```

You will be running the `assert` with a `(condition, message)` which represents a tuple as the first parameter, and this happens cause non-empty tuple in Python is **always** `True`. However, you can do separately without problem:

```
assert (condition), "message"
```

Example:

```
>>> assert (1==2), ("This condition returns a %s value.") % "False"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: This condition returns a False value.
```

2. Debug purpose

If you are wondering regarding when use `assert` statement. Take an example used in real life:

* When your program tends to control each parameter entered by the user or whatever else:

```
def loremipsum(**kwargs):
    kwargs.pop('bar') # return 0 if "bar" isn't in parameter
    kwargs.setdefault('foo', type(self)) # returns `type(self)` value by default
    assert (len(kwargs) == 0), "unrecognized parameter passed in %s" % ', '.join(kwargs.keys())
```

* Another case is on math when 0 or non-positive as a coefficient or constant on a certain equation:

```
def discount(item, percent):
    price = int(item['price'] * (1.0 - percent))
    print(price)
    assert (0 <= price <= item['price']),\
            "Discounted prices cannot be lower than 0 "\
            "and they cannot be higher than the original price."

    return price
```

* or even a simple example of a boolean implementation:

```
def true(a, b):
    assert (a == b), "False"
    return 1

def false(a, b):
    assert (a != b), "True"
    return 0
```

3. Data processing or data validation

The utmost importance is to not rely on the `assert` statement to execute data processing or data validation because this statement can be turned off on the Python initialization with `-O` or `-OO` flag – meaning value 1, 2, and 0 (as default), respectively – or `PYTHONOPTIMIZE` environment variable.

Value 1:

* asserts are disabled;

* bytecode files are generated using `.pyo` extension instead of `.pyc`;

* `sys.flags.optimize` is set to 1 (`True`);

* and, `__debug__` is set to `False`;

Value 2: disables one more stuff

* docstrings are disabled;

Therefore, using the `assert` statement to validate a sort of expected data is extremely dangerous, implying even to some security issues. Then, if you need to validate some permission I recommend you `raise AuthError` instead. As a preconditional effective, an `assert` is commonly used by programmers on libraries or modules that do not have a user interact directly.

share improve this answer follow

answered Apr 20 '19 at 13:21

slayer

158 ● 2 ● 8

add a comment

4

As summarized concisely on the C2 Wiki:

> An assertion is a boolean expression at a specific point in a program which will be true *unless there is a bug in the program.*

You can use an `assert` statement to document your understanding of the code at a particular program point. For example, you can document assumptions or guarantees about inputs (preconditions), program state (invariants), or outputs (postconditions).

Should your assertion ever fail, this is an alert for you (or your successor) that your understanding of the program was wrong when you wrote it, and that it likely contains a bug.

For more information, John Regehr has a wonderful blog post on the Use of Assertions, which applies to the Python `assert` statement as well.

share improve this answer follow

answered Dec 8 '18 at 10:14

avandeursen

7,398 ● 2 ● 34 ● 46

add a comment

3

If you ever want to know exactly what a reserved function does in python, type in `help(enter_keyword)`

Make sure if you are entering a reserved keyword that you enter it as a string.

share improve this answer follow

edited Jan 7 '16 at 3:36

answered Jul 16 '15 at 3:51

ytpillai

3,056 ● 22 ● 41

add a comment

3

Python **assert** is basically a debugging aid which test condition for internal self-check of your code. Assert makes debugging really easy when your code gets into impossible edge cases. Assert check those impossible cases.

Let's say there is a function to calculate price of item after discount :

```
def calculate_discount(price, discount):
    discounted_price = price - [discount*price]
    assert 0 <= discounted_price <= price
    return discounted_price
```

here, discounted_price can never be less than 0 and greater than actual price. So, in case the above condition is violated assert raises an Assertion Error, which helps the developer to identify that something impossible had happened.

Hope it helps :)

share improve this answer follow

answered Apr 15 '18 at 17:13

Nitish Chauhan

159 ● 1 ● 7

2 `assert` is useful in a debugging context, but should not be relied outside of a debugging context. – FluxIX Sep 27 '18 at 3:25

add a comment

3

My short explanation is:

- `assert` raises `AssertionError` if expression is false, otherwise just continues the code, and if there's a comma whatever it is it will be `AssertionError: whatever after comma`, and to code is like: `raise AssertionError(whatever after comma)`

A related tutorial about this:

> https://www.tutorialspoint.com/python/assertions_in_python.htm

share improve this answer follow

answered Sep 23 '18 at 5:55

U10-Forward

36.3k ● 7 ● 38 ● 63

The answer provides *how* to use an `assert`, but not *when* to use (or not use) an `assert`; also noting that an `assert` can be disabled if `__debug__` is `False` would be useful. – FluxIX Sep 27 '18 at 3:23

add a comment

2

In Pycharm, if you use `assert` along with `isinstance` to declare an object's type, it will let you access the methods and attributes of the parent object while you are coding, it will auto-complete automatically.

For example, let's say `self.object1.object2` is a `MyClass` object.

```
import MyClasss

def code_it(self):
    testObject = self.object1.object2 # at this point, program doesn't know that testObject  is a M
    assert isinstance(testObject , MyClasss) # now the program knows testObject is a MyClass object
    testObject.do_it() # from this point on, PyCharm will be able to auto-complete when you are wor
```

share improve this answer follow

answered May 11 at 17:05

alwbtc

21.3k 48 114 167

add a comment

1

As written in other answers, `assert` statements are used to check the state of the program at a given point.

I won't repeat what was said about associated message, parentheses, or `-O` option and `__debug__` constant. Check also the doc for first hand information. I will focus on your question: what is the use of `assert`? More precisely, when (and when not) should one use `assert`?

The `assert` statements are useful to debug a program, but discouraged to check user input. I use the following rule of thumb: keep assertions to detect a *this should not happen* situation. A user input may be incorrect, e.g. a password too short, but this is not a *this should not happen* case. If the diameter of a circle is not twice as large as its radius, you are in a *this should not happen* case.

The most interesting, in my mind, use of `assert` is inspired by the programming by contract as described by B. Meyer in [Object-Oriented Software Construction](https://www.eiffel.org/doc/eiffel/Object-Oriented_Software_Construction%2C_2nd_Edition) and implemented in the [Eiffel programming language](https://en.wikipedia.org/wiki/Eiffel_(programming_language)). You can't fully emulate programming by contract using the `assert` statement, but it's interesting to keep the intent.

Here's an example. Imagine you have to write a `head` function (like the [`head` function in Haskell](http://www.zvon.org/other/haskell/Outputprelude/head_f.html)). The specification you are given is: "if the list is not empty, return the first item of a list". Look at the following implementations:

```
>>> def head1(xs): return xs[0]
```

And

```
>>> def head2(xs):
...     if len(xs) > 0:
...         return xs[0]
...     else:
...         return None
```

(Yes, this can be written as `return xs[0] if xs else None`, but that's not the point)·

If the list is not empty, both functions have the same result and this result is correct:

```
>>> head1([1, 2, 3]) == head2([1, 2, 3]) == 1
True
```

Hence, both implementations are (I hope) correct. They differ when you try to take the head item of an empty list:

```
>>> head1([])
Traceback (most recent call last):
...
IndexError: list index out of range
```

But:

```
>>> head2([]) is None
True
```

Again, both implementations are correct, because no one should pass an empty list to these functions (we are *out of the specification*). That's an incorrect call, but if you do such a call, anything can happen. One function raises an exception, the other returns a special value. The most important is: **we can't rely on this behavior**. If `xs` is empty, this will work:

```
print(head2(xs))
```

But this will crash the program:

```
print(head1(xs))
```

To avoid some surprises, I would like to know when I'm passing some unexpected argument to a function. In other words: I would like to know when the observable behavior is not reliable, because it depends on the implementation, not on the specification. Of course, I can read the specification, but programmers do not always read carefully the docs.

Imagine if I had a way to insert the specification into the code to get the following effect: when I violate the specification, e.g by passing an empty list to `head`, I get a warning. That would be a great help to write a correct (i.e. compliant with the specification) program. And that's where `assert` enters on the scene:

```
>>> def head1(xs):
...     assert len(xs) > 0, "The list must not be empty"
...     return xs[0]
```

And

```
>>> def head2(xs):
...     assert len(xs) > 0, "The list must not be empty"
...     if len(xs) > 0:
...         return xs[0]
...     else:
...         return None
```

Now, we have:

```
>>> head1([])
Traceback (most recent call last):
...
AssertionError: The list must not be empty
```

And:

```
>>> head2([])
Traceback (most recent call last):
...
AssertionError: The list must not be empty
```

Note that `head1` throws an `AssertionError`, not an `IndexError`. That's important because an `AssertionError` is not *any* runtime error: it signals a violation of the specification. I wanted a warning, but I get an error. Fortunately, I can disable the check (using the `-O` option), but at my own risks. I will do it a crash is really expensive, and hope for the best. Imagine my program is embedded in a spaceship that travels through a black hole. I will disable assertions and hope the program is robust enough to not crash as long as possible.

This example was only about preconditions, be you can use `assert` to check postconditions (the return value and/or the state) and invariants (state of a class). Note that checking postconditions and invariants with `assert` can be cumbersome:

- for postconditions, you need to assign the return value to a variable, and maybe to store the iniial state of the object if you are dealing with a method;
- for invariants, you have to check the state before and after a method call.

You won't have something as sophisticated as Eiffel, but you can however improve the overall quality of a program.

To summarize, the `assert` statement is a convenient way to detect a *this should not happen* situation. Violations of the specification (e.g. passing an empty list to `head`) are first class *this should not happen* situations. Hence, while the `assert` statement may be used to detect any unexpected situation, it is a privilegied way to ensure that the specification is fulfilled. Once you have inserted `assert` statements into the code to represent the specification, we can hope you have improved the quality of the program because incorrect arguments, incorrect return values, incorrect states of a class..., will be reported.

answered Apr 29 at 18:26

jferard
5,639 ● 1 ● 8 ● 21

-2

format : assert Expression[,arguments] When assert encounters a statement,Python evaluates the expression.If the statement is not true,an exception is raised(assertionError). If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError. AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback. Example:

```
def KelvinToFahrenheit(Temperature):
    assert (Temperature >= 0),"Colder than absolute zero!"
    return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

When the above code is executed, it produces the following result:

```
32.0
451
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print KelvinToFahrenheit(-5)
  File "test.py", line 4, in KelvinToFahrenheit
    assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

share improve this answer follow

edited Dec 26 '14 at 15:41

answered Dec 26 '14 at 15:16

bhavya joshi

677 ● 7 ● 17

add a comment

-2

```
def getUser(self, id, Email):

    user_key = id and id or Email

    assert user_key
```

Can be used to ensure parameters are passed in the function call.

share improve this answer follow

edited Feb 19 '16 at 21:00

Cleb

17.7k ● 13 ● 75 ● 112

answered Mar 24 '15 at 11:54

user2725012

45 ● 1 ● 5

1 This will work, but from what I understand, asserts shouldn't be used for checking user-input, because they can be turned off at run-time. If you really want to enforce or validate user-input use the `if not user_key: raise ValueError()` Check last 2 paragraphs here: wiki.python.org/moin/UsingAssertionsEffectively – alpha_989 Jan 14 '18 at 17:53

`assert` should not be used for input validation because either the validation will be stripped out if `__debug__` is `False`. Also using assertions for non-debug purposes can cause people to catch the resulting `AssertionError`s, which can make debugging more difficult instead of less. – FluxIX Aug 26 '18 at 0:33

add a comment

-4

```
>>>this_is_very_complex_function_result = 9
>>>c = this_is_very_complex_function_result
>>>test_us = (c < 4)

>>> #first we try without assert
>>>if test_us == True:
    print("YES! I am right!")
else:
    print("I am Wrong, but the program still RUNS!")

I am Wrong, but the program still RUNS!


>>> #now we try with assert
>>> assert test_us
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    assert test_us
AssertionError
>>>
```

share improve this answer follow

answered Apr 25 '17 at 5:04

rianhariadi.com
1

add a comment

-4

Basically the assert keyword meaning is that if the condition is not true then it through an assertionerror else it continue for example in python.

code-1

```
a=5

b=6

assert a==b
```

OUTPUT:

```
assert a==b

AssertionError
```

code-2

```
a=5

b=5

assert a==b
```

OUTPUT:

```
Process finished with exit code 0
```

share improve this answer follow

edited Jul 27 '17 at 7:22
Dinesh Pundkar
3,402 ● 1 ● 17 ● 33

answered Jul 26 '17 at 17:19
ujjwal_bansal
15 ● 2

2 please format your code properly. also, how does this improve on previous answers? – c2huc2hu Jul 26 '17 at 17:31

is there any problem in my explanation? – ujjwal_bansal Aug 2 '17 at 22:18

your explanation doesn't add anything to the existing answers, and the poor grammar makes it hard to read. if you're looking for questions to answer, consider browsing the new questions feed. – c2huc2hu Aug 3 '17 at 1:46

The provided answer does answer *how* to use an `assert`, but does not answer *when* to use (or not use) an `assert`. – FluxIX Sep 27 '18 at 3:20

add a comment

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.

Not the answer you're looking for? Browse other questions tagged python assert assertions or ask your own question.

The Overflow Blog

- The key components for building a React community
- Podcast 258: why are you coding in bed?

Featured on Meta

- Improved experience for users with review suspensions
- CEO Blog: Some exciting news about

fundraising

2020 Community Moderator Election Results

Linked

0 Use of assert statement in test cases written in python

-1 How can I do it assertion on plain text in Python?

488 Best practice for Python assert

211 What is unit testing?

48 Is there a way to use Python unit test assertions outside of a TestCase?

2 Loop not working in socket.connect_ex()

1 Comparison in Python assert statement

-3 Convert a range score into a letter grade in Python

0 How to check if text is present using selenium webdriver python?

2 JSON Data validation in python

See more linked questions

Related

4967 Calling an external command from Python

5796 What are metaclasses in Python?

2028 How do you assert that a certain exception is thrown in JUnit 4 tests?

3267 Finding the index of an item in a list

10382 What does the “yield” keyword do?

6161 Does Python have a ternary conditional operator?

6199 What does if __name__ == “__main__”: do?

3209 Iterating over dictionaries using 'for' loops

3599 Does Python have a string 'contains' substring method?

2167 Why is “1000000000000000 in range(1000000000000001)” so fast in Python 3?

Hot Network Questions

- Who made the first derivation of the angle to maximise projectile range, which turned out to be wrong?
- Are there any spells that affect multiple targets at the same time in Harry Potter?
- Counterexamples against all odds
- Would mail in voting disproportionately help Republicans?
- "I was no longer working for X" vs. "I no longer worked for X"
- Is it important to maintain a work-life balance as a PhD student?

- Saving one file format with a different file extension. JPG - PNG; MOV - MP4
- What is the interaction between Green-Flame Blade and Absorb elements
- Why is this regular expression so slow in Java?
- The F-16s that were scrambled to intercept United 93 did not have time to arm the jets, so the pilots were going to “ram it” how does that work?
- I'm translating a play and in the midst of it there is a German sentence I cannot understand
- What is the highest possible altitude of a sub-orbital flight?
- What are the criteria for degree revocation?
- Too many class what the alternatives?
- How languages compare with the number of different syllables from all words?
- If "bro splits" don't work, why is literally every muscular guy in my gym doing them?
- Is it good practice to write unit tests when features are changed frequently?
- Transcribing by ear, is it done with the root as reference, or the previous note?
- Did/does the US use statute or nautical miles to decide who gets astronaut wings?
- Is it okay to hit wrong notes?
- Famous connect wall
- Would swapping Sacred Flame to Toll the Dead for Evil NPCs affect balance?
- If I add a copyright disclaimer to my app am I exempt of charges?

How did bank transactions (or "data" transactions) work when it took people weeks to travel vast distances?

Question feed