Module: tf.compat.v1.test / tf.test

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test#functions)

Testing.

See the [Testing](https://tensorflow.org/api_guides/python/test) guide.

**Note:** **tf.compat.v1.test.mock** is an alias to the python **mock** or **unittest.mock** depending on the python version.

Classes

[class Benchmark](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark): Abstract class that provides helpers for TensorFlow benchmarks.

[class StubOutForTesting](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting): Support class for stubbing methods out for unit testing.

[class TestCase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase): Base class for tests that need to test TensorFlow.

Functions

[assert\_equal\_graph\_def(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/assert_equal_graph_def): Asserts that two GraphDefs are (mostly) the same.

[benchmark\_config(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/benchmark_config): Returns a tf.compat.v1.ConfigProto for disabling the dependency optimizer.

[compute\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/compute_gradient): Computes and returns the theoretical and numerical Jacobian.

[compute\_gradient\_error(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/compute_gradient_error): Computes the gradient error.

[create\_local\_cluster(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/create_local_cluster): Create and start local servers and return the associated Serverobjects.

[get\_temp\_dir(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/get_temp_dir): Returns a temporary directory for use during tests.

[gpu\_device\_name(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/gpu_device_name): Returns the name of a GPU device if available or the empty string.

[is\_built\_with\_cuda(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_built_with_cuda): Returns whether TensorFlow was built with CUDA (GPU) support.

[is\_gpu\_available(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_gpu_available): Returns whether TensorFlow can access a GPU.

[main(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/main): Runs all unit tests.

[test\_src\_dir\_path(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/test_src_dir_path): Creates an absolute test srcdir path given a relative path.

# tf.compat.v1.test.assert\_equal\_graph\_def

Asserts that two GraphDefs are (mostly) the same.

tf.compat.v1.test.assert\_equal\_graph\_def(  
    actual,  
    expected,  
    checkpoint\_v2=False,  
    hash\_table\_shared\_name=False  
)

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

Compares two GraphDef protos for equality, ignoring versions and ordering of nodes, attrs, and control inputs. Node names are used to match up nodes between the graphs, so the naming of nodes must be consistent.

#### Args:

* **actual**: The GraphDef we have.
* **expected**: The GraphDef we expected.
* **checkpoint\_v2**: boolean determining whether to ignore randomized attribute values that appear in V2 checkpoints.
* **hash\_table\_shared\_name**: boolean determining whether to ignore randomized shared\_names that appear in HashTableV2 op defs.

#### Raises:

* **AssertionError**: If the GraphDefs do not match.
* **TypeError**: If either argument is not a GraphDef.

# tf.compat.v1.test.compute\_gradient

Computes and returns the theoretical and numerical Jacobian.

tf.compat.v1.test.compute\_gradient(  
    x,  
    x\_shape,  
    y,  
    y\_shape,  
    x\_init\_value=None,  
    delta=0.001,  
    init\_targets=None,  
    extra\_feed\_dict=None  
)

Defined in [python/ops/gradient\_checker.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradient_checker.py).

If x or y is complex, the Jacobian will still be real but the corresponding Jacobian dimension(s) will be twice as large. This is required even if both input and output is complex since TensorFlow graphs are not necessarily holomorphic, and may have gradients not expressible as complex numbers. For example, if x is complex with shape [m] and y is complex with shape [n], each Jacobian J will have shape [m \* 2, n \* 2] with

J[:m, :n] = d(Re y)/d(Re x)  
J[:m, n:] = d(Im y)/d(Re x)  
J[m:, :n] = d(Re y)/d(Im x)  
J[m:, n:] = d(Im y)/d(Im x)

#### Args:

* **x**: a tensor or list of tensors
* **x\_shape**: the dimensions of x as a tuple or an array of ints. If x is a list, then this is the list of shapes.
* **y**: a tensor
* **y\_shape**: the dimensions of y as a tuple or an array of ints.
* **x\_init\_value**: (optional) a numpy array of the same shape as "x" representing the initial value of x. If x is a list, this should be a list of numpy arrays. If this is none, the function will pick a random tensor as the initial value.
* **delta**: (optional) the amount of perturbation.
* **init\_targets**: list of targets to run to initialize model params.
* **extra\_feed\_dict**: dict that allows fixing specified tensor values during the Jacobian calculation.

#### Returns:

Two 2-d numpy arrays representing the theoretical and numerical Jacobian for dy/dx. Each has "x\_size" rows and "y\_size" columns where "x\_size" is the number of elements in x and "y\_size" is the number of elements in y. If x is a list, returns a list of two numpy arrays.

# tf.compat.v1.test.compute\_gradient\_error

Computes the gradient error.

tf.compat.v1.test.compute\_gradient\_error(  
    x,  
    x\_shape,  
    y,  
    y\_shape,  
    x\_init\_value=None,  
    delta=0.001,  
    init\_targets=None,  
    extra\_feed\_dict=None  
)

Defined in [python/ops/gradient\_checker.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradient_checker.py).

Computes the maximum error for dy/dx between the computed Jacobian and the numerically estimated Jacobian.

This function will modify the tensors passed in as it adds more operations and hence changing the consumers of the operations of the input tensors.

This function adds operations to the current session. To compute the error using a particular device, such as a GPU, use the standard methods for setting a device (e.g. using with sess.graph.device() or setting a device function in the session constructor).

#### Args:

* **x**: a tensor or list of tensors
* **x\_shape**: the dimensions of x as a tuple or an array of ints. If x is a list, then this is the list of shapes.
* **y**: a tensor
* **y\_shape**: the dimensions of y as a tuple or an array of ints.
* **x\_init\_value**: (optional) a numpy array of the same shape as "x" representing the initial value of x. If x is a list, this should be a list of numpy arrays. If this is none, the function will pick a random tensor as the initial value.
* **delta**: (optional) the amount of perturbation.
* **init\_targets**: list of targets to run to initialize model params.
* **extra\_feed\_dict**: dict that allows fixing specified tensor values during the Jacobian calculation.

#### Returns:

The maximum error in between the two Jacobians.

# tf.compat.v1.test.get\_temp\_dir

Returns a temporary directory for use during tests.

tf.compat.v1.test.get\_temp\_dir()

Defined in [python/platform/test.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/test.py).

There is no need to delete the directory after the test.

#### Returns:

The temporary directory.

# tf.compat.v1.test.StubOutForTesting

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting#top_of_page)
* [Class StubOutForTesting](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting#class_stuboutfortesting)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting#methods)
  + [CleanUp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test/StubOutForTesting#CleanUp)

## Class StubOutForTesting

Support class for stubbing methods out for unit testing.

Defined in [python/platform/googletest.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/googletest.py).

#### Sample Usage:

You want os.path.exists() to always return true during testing.

stubs = StubOutForTesting() stubs.Set(os.path, 'exists', lambda x: 1) ... stubs.CleanUp()

The above changes os.path.exists into a lambda that returns 1. Once the ... part of the code finishes, the CleanUp() looks up the old value of os.path.exists and restores it.

## \_\_init\_\_

\_\_init\_\_()

## Methods

### CleanUp

CleanUp()

Undoes all SmartSet() & Set() calls, restoring original definitions.

### Set

Set(  
    parent,  
    child\_name,  
    new\_child  
)

In parent, replace child\_name's old definition with new\_child.

The parent could be a module when the child is a function at module scope. Or the parent could be a class when a class' method is being replaced. The named child is set to new\_child, while the prior definition is saved away for later, when UnsetAll() is called.

This method supports the case where child\_name is a staticmethod or a classmethod of parent.

#### Args:

* **parent**: The context in which the attribute child\_name is to be changed.
* **child\_name**: The name of the attribute to change.
* **new\_child**: The new value of the attribute.

### SmartSet

SmartSet(  
    obj,  
    attr\_name,  
    new\_attr  
)

Replace obj.attr\_name with new\_attr.

This method is smart and works at the module, class, and instance level while preserving proper inheritance. It will not stub out C types however unless that has been explicitly allowed by the type.

This method supports the case where attr\_name is a staticmethod or a classmethod of obj.

#### Notes:

* If obj is an instance, then it is its class that will actually be stubbed. Note that the method Set() does not do that: if obj is an instance, it (and not its class) will be stubbed.
* The stubbing is using the builtin getattr and setattr. So, the **get** and **set** will be called when stubbing (TODO: A better idea would probably be to manipulate obj.**dict** instead of getattr() and setattr()).

#### Args:

* **obj**: The object whose attributes we want to modify.
* **attr\_name**: The name of the attribute to modify.
* **new\_attr**: The new value for the attribute.

#### Raises:

* **AttributeError**: If the attribute cannot be found.

### SmartUnsetAll

SmartUnsetAll()

Reverses SmartSet() calls, restoring things to original definitions.

This method is automatically called when the StubOutForTesting() object is deleted; there is no need to call it explicitly.

It is okay to call SmartUnsetAll() repeatedly, as later calls have no effect if no SmartSet() calls have been made.

### UnsetAll

UnsetAll()

Reverses Set() calls, restoring things to their original definitions.

This method is automatically called when the StubOutForTesting() object is deleted; there is no need to call it explicitly.

It is okay to call UnsetAll() repeatedly, as later calls have no effect if no Set() calls have been made.

### \_\_enter\_\_

\_\_enter\_\_()

### \_\_exit\_\_

\_\_exit\_\_(  
    unused\_exc\_type,  
    unused\_exc\_value,  
    unused\_tb  
)

# tf.compat.v1.test.test\_src\_dir\_path

Creates an absolute test srcdir path given a relative path.

tf.compat.v1.test.test\_src\_dir\_path(relative\_path)

Defined in [python/platform/test.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/test.py).

#### Args:

* **relative\_path**: a path relative to tensorflow root. e.g. "core/platform".

#### Returns:

An absolute path to the linked in runfiles.

# tf.test.assert\_equal\_graph\_def

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/assert_equal_graph_def#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/assert_equal_graph_def#aliases)

Asserts that two GraphDefs are (mostly) the same.

### Aliases:

* tf.compat.v2.test.assert\_equal\_graph\_def
* tf.test.assert\_equal\_graph\_def

tf.test.assert\_equal\_graph\_def(  
    expected,  
    actual  
)

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

Compares two GraphDef protos for equality, ignoring versions and ordering of nodes, attrs, and control inputs. Node names are used to match up nodes between the graphs, so the naming of nodes must be consistent. This function ignores randomized attribute values that may appear in V2 checkpoints.

#### Args:

* **expected**: The GraphDef we expected.
* **actual**: The GraphDef we have.

#### Raises:

* **AssertionError**: If the GraphDefs do not match.
* **TypeError**: If either argument is not a GraphDef.

# tf.test.Benchmark

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark#top_of_page)
* [Class Benchmark](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark#class_benchmark)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/Benchmark#methods)

## Class Benchmark

Abstract class that provides helpers for TensorFlow benchmarks.

### Aliases:

* Class tf.compat.v1.test.Benchmark
* Class tf.compat.v2.test.Benchmark
* Class tf.test.Benchmark

Defined in [python/platform/benchmark.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/benchmark.py).

## \_\_init\_\_

\_\_init\_\_()

## Methods

### evaluate

evaluate(tensors)

Evaluates tensors and returns numpy values.

#### Args:

* **tensors**: A Tensor or a nested list/tuple of Tensors.

#### Returns:

tensors numpy values.

### is\_abstract

@classmethod  
is\_abstract(cls)

### report\_benchmark

report\_benchmark(  
    iters=None,  
    cpu\_time=None,  
    wall\_time=None,  
    throughput=None,  
    extras=None,  
    name=None,  
    metrics=None  
)

Report a benchmark.

#### Args:

* **iters**: (optional) How many iterations were run
* **cpu\_time**: (optional) Median or mean cpu time in seconds.
* **wall\_time**: (optional) Median or mean wall time in seconds.
* **throughput**: (optional) Throughput (in MB/s)
* **extras**: (optional) Dict mapping string keys to additional benchmark info. Values may be either floats or values that are convertible to strings.
* **name**: (optional) Override the BenchmarkEntry name with name. Otherwise it is inferred from the top-level method name.
* **metrics**: (optional) A list of dict, where each dict has the keys below name (required), string, metric name value (required), double, metric value min\_value (optional), double, minimum acceptable metric value max\_value (optional), double, maximum acceptable metric value

### run\_op\_benchmark

run\_op\_benchmark(  
    sess,  
    op\_or\_tensor,  
    feed\_dict=None,  
    burn\_iters=2,  
    min\_iters=10,  
    store\_trace=False,  
    store\_memory\_usage=True,  
    name=None,  
    extras=None,  
    mbs=0  
)

Run an op or tensor in the given session. Report the results.

#### Args:

* **sess**: Session object to use for timing.
* **op\_or\_tensor**: Operation or Tensor to benchmark.
* **feed\_dict**: A dict of values to feed for each op iteration (see the feed\_dict parameter of Session.run).
* **burn\_iters**: Number of burn-in iterations to run.
* **min\_iters**: Minimum number of iterations to use for timing.
* **store\_trace**: Boolean, whether to run an extra untimed iteration and store the trace of iteration in returned extras. The trace will be stored as a string in Google Chrome trace format in the extras field "full\_trace\_chrome\_format". Note that trace will not be stored in test\_log\_pb2.TestResults proto.
* **store\_memory\_usage**: Boolean, whether to run an extra untimed iteration, calculate memory usage, and store that in extras fields.
* **name**: (optional) Override the BenchmarkEntry name with name. Otherwise it is inferred from the top-level method name.
* **extras**: (optional) Dict mapping string keys to additional benchmark info. Values may be either floats or values that are convertible to strings.
* **mbs**: (optional) The number of megabytes moved by this op, used to calculate the ops throughput.

#### Returns:

A dict containing the key-value pairs that were passed to report\_benchmark. If store\_traceoption is used, then full\_chrome\_trace\_format will be included in return dictionary even though it is not passed to report\_benchmark with extras.

# tf.test.benchmark\_config

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/benchmark_config#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/benchmark_config#aliases)

Returns a tf.compat.v1.ConfigProto for disabling the dependency optimizer.

### Aliases:

* tf.compat.v1.test.benchmark\_config
* tf.compat.v2.test.benchmark\_config
* tf.test.benchmark\_config

tf.test.benchmark\_config()

Defined in [python/platform/benchmark.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/benchmark.py).

#### Returns:

A TensorFlow ConfigProto object.

# tf.test.compute\_gradient

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/compute_gradient#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/compute_gradient#aliases)

Computes the theoretical and numeric Jacobian of f.

### Aliases:

* tf.compat.v2.test.compute\_gradient
* tf.test.compute\_gradient

tf.test.compute\_gradient(  
    f,  
    x,  
    delta=0.001  
)

Defined in [python/ops/gradient\_checker\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradient_checker_v2.py).

With y = f(x), computes the theoretical and numeric Jacobian dy/dx.

#### Args:

* **f**: the function.
* **x**: a list arguments for the function
* **delta**: (optional) perturbation used to compute numeric Jacobian.

#### Returns:

A pair of lists, where the first is a list of 2-d numpy arrays representing the theoretical Jacobians for each argument, and the second list is the numerical ones. Each 2-d array has "x\_size" rows and "y\_size" columns where "x\_size" is the number of elements in the corresponding argument and "y\_size" is the number of elements in f(x).

#### Raises:

* **ValueError**: If result is empty but the gradient is nonzero.
* **ValueError**: If x is not list, but any other type.

#### Example:

@tf.function  
def test\_func(x):  
  return x\*x  
  
theoretical, numerical = tf.test.compute\_gradient(test\_func, [1.0])  
theoretical, numerical  
# ((array([[2.]], dtype=float32),), (array([[2.000004]], dtype=float32),))

# tf.test.create\_local\_cluster

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/create_local_cluster#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/create_local_cluster#aliases)

Create and start local servers and return the associated Server objects.

### Aliases:

* tf.compat.v1.test.create\_local\_cluster
* tf.compat.v2.test.create\_local\_cluster
* tf.test.create\_local\_cluster

tf.test.create\_local\_cluster(  
    num\_workers,  
    num\_ps,  
    protocol='grpc',  
    worker\_config=None,  
    ps\_config=None  
)

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

"PS" stands for "parameter server": a task responsible for storing and updating the model's parameters. Other tasks send updates to these parameters as they work on optimizing the parameters. This particular division of labor between tasks is not required, but is common for distributed training.

Read more at https://www.tensorflow.org/guide/extend/architecture

Figure illustrates the interaction of these components. "/job:worker/task:0" and "/job:ps/task:0" are both tasks with worker services.

#### Example:

workers, \_ = tf.test.create\_local\_cluster(num\_workers=2, num\_ps=2)  
  
worker\_sessions = [tf.compat.v1.Session(w.target) for w in workers]  
  
with tf.device("/job:ps/task:0"):  
  ...  
with tf.device("/job:ps/task:1"):  
  ...  
with tf.device("/job:worker/task:0"):  
  ...  
with tf.device("/job:worker/task:1"):  
  ...  
  
worker\_sessions[0].run(...)

#### Args:

* **num\_workers**: Number of worker servers to start.
* **num\_ps**: Number of PS servers to start.
* **protocol**: Communication protocol. Allowed values are documented in the documentation of [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server).
* **worker\_config**: (optional) tf.ConfigProto to initialize workers. Can be used to instantiate multiple devices etc.
* **ps\_config**: (optional) tf.ConfigProto to initialize PS servers.

#### Returns:

A tuple (worker\_servers, ps\_servers). worker\_servers is a list of num\_workers objects of type [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) (all running locally); and ps\_servers is a list of num\_ps objects of similar type.

#### Raises:

* **ImportError**: if portpicker module was not found at load time

# tf.test.gpu\_device\_name

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/gpu_device_name#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/gpu_device_name#aliases)

Returns the name of a GPU device if available or the empty string.

### Aliases:

* tf.compat.v1.test.gpu\_device\_name
* tf.compat.v2.test.gpu\_device\_name
* tf.test.gpu\_device\_name

tf.test.gpu\_device\_name()

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

# tf.test.is\_built\_with\_cuda

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_built_with_cuda#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_built_with_cuda#aliases)

Returns whether TensorFlow was built with CUDA (GPU) support.

### Aliases:

* tf.compat.v1.test.is\_built\_with\_cuda
* tf.compat.v2.test.is\_built\_with\_cuda
* tf.test.is\_built\_with\_cuda

tf.test.is\_built\_with\_cuda()

Defined in [python/platform/test.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/test.py).

# tf.test.is\_gpu\_available

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_gpu_available#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_gpu_available#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_gpu_available#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_gpu_available#used_in_the_tutorials)

Returns whether TensorFlow can access a GPU.

### Aliases:

* tf.compat.v1.test.is\_gpu\_available
* tf.compat.v2.test.is\_gpu\_available
* tf.test.is\_gpu\_available

tf.test.is\_gpu\_available(  
    cuda\_only=False,  
    min\_cuda\_compute\_capability=None  
)

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

### Used in the tutorials:

* [Tensors and Operations](https://www.tensorflow.org/beta/tutorials/eager/basics)
* [Text classification of movie reviews with Keras and TensorFlow Hub](https://www.tensorflow.org/beta/tutorials/keras/basic_text_classification_with_tfhub)

**Warning:** if a non-GPU version of the package is installed, the function would also return False. Use [**tf.test.is\_built\_with\_cuda**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/is_built_with_cuda) to validate if TensorFlow was build with CUDA support.

#### Args:

* **cuda\_only**: limit the search to CUDA GPUs.
* **min\_cuda\_compute\_capability**: a (major,minor) pair that indicates the minimum CUDA compute capability required, or None if no requirement.

#### Returns:

True if a GPU device of the requested kind is available.

# tf.test.main

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/main#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/main#aliases)

Runs all unit tests.

### Aliases:

* tf.compat.v1.test.main
* tf.compat.v2.test.main
* tf.test.main

tf.test.main(argv=None)

Defined in [python/platform/test.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/test.py).

# tf.test.TestCase

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase#top_of_page)
* [Class TestCase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase#class_testcase)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase#__init__)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase#child_classes)

## Class TestCase

Base class for tests that need to test TensorFlow.

### Aliases:

* Class tf.compat.v1.test.TestCase
* Class tf.compat.v2.test.TestCase
* Class tf.test.TestCase

Defined in [python/framework/test\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/test_util.py).

## \_\_init\_\_

\_\_init\_\_(methodName='runTest')

## Child Classes

[class failureException](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase/failureException)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    \*args,  
    \*\*kwds  
)

### \_\_eq\_\_

\_\_eq\_\_(other)

### addCleanup

addCleanup(  
    function,  
    \*args,  
    \*\*kwargs  
)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

### addTypeEqualityFunc

addTypeEqualityFunc(  
    typeobj,  
    function  
)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

#### Args:

* **typeobj**: The data type to call this function on when both values are of the same type in assertEqual().
* **function**: The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

### assertAllClose

assertAllClose(  
    a,  
    b,  
    rtol=1e-06,  
    atol=1e-06,  
    msg=None  
)

Asserts that two structures of numpy arrays or Tensors, have near values.

a and b can be arbitrarily nested structures. A layer of a nested structure can be a dict, namedtuple, tuple or list.

#### Args:

* **a**: The expected numpy ndarray, or anything that can be converted into a numpy ndarray(including Tensor), or any arbitrarily nested of structure of these.
* **b**: The actual numpy ndarray, or anything that can be converted into a numpy ndarray(including Tensor), or any arbitrarily nested of structure of these.
* **rtol**: relative tolerance.
* **atol**: absolute tolerance.
* **msg**: Optional message to report on failure.

#### Raises:

* **ValueError**: if only one of a[p] and b[p] is a dict or a[p] and b[p] have different length, where [p] denotes a path to the nested structure, e.g. given a = [(1, 1), {'d': (6, 7)}]and [p] = [1]['d'], then a[p] = (6, 7).

### assertAllCloseAccordingToType

assertAllCloseAccordingToType(  
    a,  
    b,  
    rtol=1e-06,  
    atol=1e-06,  
    float\_rtol=1e-06,  
    float\_atol=1e-06,  
    half\_rtol=0.001,  
    half\_atol=0.001,  
    bfloat16\_rtol=0.01,  
    bfloat16\_atol=0.01,  
    msg=None  
)

Like assertAllClose, but also suitable for comparing fp16 arrays.

In particular, the tolerance is reduced to 1e-3 if at least one of the arguments is of type float16.

#### Args:

* **a**: the expected numpy ndarray or anything can be converted to one.
* **b**: the actual numpy ndarray or anything can be converted to one.
* **rtol**: relative tolerance.
* **atol**: absolute tolerance.
* **float\_rtol**: relative tolerance for float32.
* **float\_atol**: absolute tolerance for float32.
* **half\_rtol**: relative tolerance for float16.
* **half\_atol**: absolute tolerance for float16.
* **bfloat16\_rtol**: relative tolerance for bfloat16.
* **bfloat16\_atol**: absolute tolerance for bfloat16.
* **msg**: Optional message to report on failure.

### assertAllEqual

assertAllEqual(  
    a,  
    b,  
    msg=None  
)

Asserts that two numpy arrays or Tensors have the same values.

#### Args:

* **a**: the expected numpy ndarray or anything can be converted to one.
* **b**: the actual numpy ndarray or anything can be converted to one.
* **msg**: Optional message to report on failure.

### assertAllGreater

assertAllGreater(  
    a,  
    comparison\_target  
)

Assert element values are all greater than a target value.

#### Args:

* **a**: The numpy ndarray, or anything that can be converted into a numpy ndarray (including Tensor).
* **comparison\_target**: The target value of comparison.

### assertAllGreaterEqual

assertAllGreaterEqual(  
    a,  
    comparison\_target  
)

Assert element values are all greater than or equal to a target value.

#### Args:

* **a**: The numpy ndarray, or anything that can be converted into a numpy ndarray (including Tensor).
* **comparison\_target**: The target value of comparison.

### assertAllInRange

assertAllInRange(  
    target,  
    lower\_bound,  
    upper\_bound,  
    open\_lower\_bound=False,  
    open\_upper\_bound=False  
)

Assert that elements in a Tensor are all in a given range.

#### Args:

* **target**: The numpy ndarray, or anything that can be converted into a numpy ndarray(including Tensor).
* **lower\_bound**: lower bound of the range
* **upper\_bound**: upper bound of the range
* **open\_lower\_bound**: (bool) whether the lower bound is open (i.e., > rather than the default >=)
* **open\_upper\_bound**: (bool) whether the upper bound is open (i.e., < rather than the default <=)

#### Raises:

* **AssertionError**: if the value tensor does not have an ordered numeric type (float\* or int\*), or if there are nan values, or if any of the elements do not fall in the specified range.

### assertAllInSet

assertAllInSet(  
    target,  
    expected\_set  
)

Assert that elements of a Tensor are all in a given closed set.

#### Args:

* **target**: The numpy ndarray, or anything that can be converted into a numpy ndarray(including Tensor).
* **expected\_set**: (list, tuple or set) The closed set that the elements of the value of target are expected to fall into.

#### Raises:

* **AssertionError**: if any of the elements do not fall into expected\_set.

### assertAllLess

assertAllLess(  
    a,  
    comparison\_target  
)

Assert element values are all less than a target value.

#### Args:

* **a**: The numpy ndarray, or anything that can be converted into a numpy ndarray (including Tensor).
* **comparison\_target**: The target value of comparison.

### assertAllLessEqual

assertAllLessEqual(  
    a,  
    comparison\_target  
)

Assert element values are all less than or equal to a target value.

#### Args:

* **a**: The numpy ndarray, or anything that can be converted into a numpy ndarray (including Tensor).
* **comparison\_target**: The target value of comparison.

### assertAlmostEqual

assertAlmostEqual(  
    first,  
    second,  
    places=None,  
    msg=None,  
    delta=None  
)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

### assertAlmostEquals

assertAlmostEquals(  
    \*args,  
    \*\*kwargs  
)

### assertArrayNear

assertArrayNear(  
    farray1,  
    farray2,  
    err,  
    msg=None  
)

Asserts that two float arrays are near each other.

Checks that for all elements of farray1 and farray2 |f1 - f2| < err. Asserts a test failure if not.

#### Args:

* **farray1**: a list of float values.
* **farray2**: a list of float values.
* **err**: a float value.
* **msg**: Optional message to report on failure.

### assertBetween

assertBetween(  
    value,  
    minv,  
    maxv,  
    msg=None  
)

Asserts that value is between minv and maxv (inclusive).

### assertCommandFails

assertCommandFails(  
    command,  
    regexes,  
    env=None,  
    close\_fds=True,  
    msg=None  
)

Asserts a shell command fails and the error matches a regex in a list.

#### Args:

* **command**: List or string representing the command to run.
* **regexes**: the list of regular expression strings.
* **env**: Dictionary of environment variable settings. If None, no environment variables will be set for the child process. This is to make tests more hermetic. NOTE: this behavior is different than the standard subprocess module.
* **close\_fds**: Whether or not to close all open fd's in the child after forking.
* **msg**: Optional message to report on failure.

### assertCommandSucceeds

assertCommandSucceeds(  
    command,  
    regexes=(b'',),  
    env=None,  
    close\_fds=True,  
    msg=None  
)

Asserts that a shell command succeeds (i.e. exits with code 0).

#### Args:

* **command**: List or string representing the command to run.
* **regexes**: List of regular expression byte strings that match success.
* **env**: Dictionary of environment variable settings. If None, no environment variables will be set for the child process. This is to make tests more hermetic. NOTE: this behavior is different than the standard subprocess module.
* **close\_fds**: Whether or not to close all open fd's in the child after forking.
* **msg**: Optional message to report on failure.

### assertContainsExactSubsequence

assertContainsExactSubsequence(  
    container,  
    subsequence,  
    msg=None  
)

Asserts that "container" contains "subsequence" as an exact subsequence.

Asserts that "container" contains all the elements of "subsequence", in order, and without other elements interspersed. For example, [1, 2, 3] is an exact subsequence of [0, 0, 1, 2, 3, 0] but not of [0, 0, 1, 2, 0, 3, 0].

#### Args:

* **container**: the list we're testing for subsequence inclusion.
* **subsequence**: the list we hope will be an exact subsequence of container.
* **msg**: Optional message to report on failure.

### assertContainsInOrder

assertContainsInOrder(  
    strings,  
    target,  
    msg=None  
)

Asserts that the strings provided are found in the target in order.

This may be useful for checking HTML output.

#### Args:

* **strings**: A list of strings, such as [ 'fox', 'dog' ]
* **target**: A target string in which to look for the strings, such as 'The quick brown fox jumped over the lazy dog'.
* **msg**: Optional message to report on failure.

### assertContainsSubsequence

assertContainsSubsequence(  
    container,  
    subsequence,  
    msg=None  
)

Asserts that "container" contains "subsequence" as a subsequence.

Asserts that "container" contains all the elements of "subsequence", in order, but possibly with other elements interspersed. For example, [1, 2, 3] is a subsequence of [0, 0, 1, 2, 0, 3, 0] but not of [0, 0, 1, 3, 0, 2, 0].

#### Args:

* **container**: the list we're testing for subsequence inclusion.
* **subsequence**: the list we hope will be a subsequence of container.
* **msg**: Optional message to report on failure.

### assertContainsSubset

assertContainsSubset(  
    expected\_subset,  
    actual\_set,  
    msg=None  
)

Checks whether actual iterable is a superset of expected iterable.

### assertCountEqual

assertCountEqual(  
    first,  
    second,  
    msg=None  
)

An unordered sequence comparison asserting that the same elements, regardless of order. If the same element occurs more than once, it verifies that the elements occur the same number of times.

self.assertEqual(Counter(list(first)),  
                 Counter(list(second)))

Example: - [0, 1, 1] and [1, 0, 1] compare equal. - [0, 0, 1] and [0, 1] compare unequal.

### assertDTypeEqual

assertDTypeEqual(  
    target,  
    expected\_dtype  
)

Assert ndarray data type is equal to expected.

#### Args:

* **target**: The numpy ndarray, or anything that can be converted into a numpy ndarray(including Tensor).
* **expected\_dtype**: Expected data type.

### assertDeviceEqual

assertDeviceEqual(  
    device1,  
    device2,  
    msg=None  
)

Asserts that the two given devices are the same.

#### Args:

* **device1**: A string device name or TensorFlow DeviceSpec object.
* **device2**: A string device name or TensorFlow DeviceSpec object.
* **msg**: Optional message to report on failure.

### assertDictContainsSubset

assertDictContainsSubset(  
    subset,  
    dictionary,  
    msg=None  
)

Checks whether dictionary is a superset of subset.

### assertDictEqual

assertDictEqual(  
    a,  
    b,  
    msg=None  
)

Raises AssertionError if a and b are not equal dictionaries.

#### Args:

* **a**: A dict, the expected value.
* **b**: A dict, the actual value.
* **msg**: An optional str, the associated message.

#### Raises:

* **AssertionError**: if the dictionaries are not equal.

### assertEmpty

assertEmpty(  
    container,  
    msg=None  
)

Asserts that an object has zero length.

#### Args:

* **container**: Anything that implements the collections.Sized interface.
* **msg**: Optional message to report on failure.

### assertEndsWith

assertEndsWith(  
    actual,  
    expected\_end,  
    msg=None  
)

Asserts that actual.endswith(expected\_end) is True.

#### Args:

* **actual**: str
* **expected\_end**: str
* **msg**: Optional message to report on failure.

### assertEqual

assertEqual(  
    first,  
    second,  
    msg=None  
)

Fail if the two objects are unequal as determined by the '==' operator.

### assertEquals

assertEquals(  
    \*args,  
    \*\*kwargs  
)

### assertFalse

assertFalse(  
    expr,  
    msg=None  
)

Check that the expression is false.

### assertGreater

assertGreater(  
    a,  
    b,  
    msg=None  
)

Just like self.assertTrue(a > b), but with a nicer default message.

### assertGreaterEqual

assertGreaterEqual(  
    a,  
    b,  
    msg=None  
)

Just like self.assertTrue(a >= b), but with a nicer default message.

### assertIn

assertIn(  
    member,  
    container,  
    msg=None  
)

Just like self.assertTrue(a in b), but with a nicer default message.

### assertIs

assertIs(  
    expr1,  
    expr2,  
    msg=None  
)

Just like self.assertTrue(a is b), but with a nicer default message.

### assertIsInstance

assertIsInstance(  
    obj,  
    cls,  
    msg=None  
)

Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

### assertIsNone

assertIsNone(  
    obj,  
    msg=None  
)

Same as self.assertTrue(obj is None), with a nicer default message.

### assertIsNot

assertIsNot(  
    expr1,  
    expr2,  
    msg=None  
)

Just like self.assertTrue(a is not b), but with a nicer default message.

### assertIsNotNone

assertIsNotNone(  
    obj,  
    msg=None  
)

Included for symmetry with assertIsNone.

### assertItemsEqual

assertItemsEqual(  
    first,  
    second,  
    msg=None  
)

An unordered sequence comparison asserting that the same elements, regardless of order. If the same element occurs more than once, it verifies that the elements occur the same number of times.

self.assertEqual(Counter(list(first)),  
                 Counter(list(second)))

Example: - [0, 1, 1] and [1, 0, 1] compare equal. - [0, 0, 1] and [0, 1] compare unequal.

### assertJsonEqual

assertJsonEqual(  
    first,  
    second,  
    msg=None  
)

Asserts that the JSON objects defined in two strings are equal.

A summary of the differences will be included in the failure message using assertSameStructure.

#### Args:

* **first**: A string contining JSON to decode and compare to second.
* **second**: A string contining JSON to decode and compare to first.
* **msg**: Additional text to include in the failure message.

### assertLen

assertLen(  
    container,  
    expected\_len,  
    msg=None  
)

Asserts that an object has the expected length.

#### Args:

* **container**: Anything that implements the collections.Sized interface.
* **expected\_len**: The expected length of the container.
* **msg**: Optional message to report on failure.

### assertLess

assertLess(  
    a,  
    b,  
    msg=None  
)

Just like self.assertTrue(a < b), but with a nicer default message.

### assertLessEqual

assertLessEqual(  
    a,  
    b,  
    msg=None  
)

Just like self.assertTrue(a <= b), but with a nicer default message.

### assertListEqual

assertListEqual(  
    list1,  
    list2,  
    msg=None  
)

A list-specific equality assertion.

#### Args:

* **list1**: The first list to compare.
* **list2**: The second list to compare.
* **msg**: Optional message to use on failure instead of a list of differences.

### assertLogs

assertLogs(  
    logger=None,  
    level=None  
)

Fail unless a log message of level level or higher is emitted on logger\_name or its children. If omitted, level defaults to INFO and logger defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: output and records. At the end of the context manager, the output attribute will be a list of the matching formatted log messages and the records attribute will be a list of the corresponding LogRecord objects.

Example::

with self.assertLogs('foo', level='INFO') as cm:  
    logging.getLogger('foo').info('first message')  
    logging.getLogger('foo.bar').error('second message')  
self.assertEqual(cm.output, ['INFO:foo:first message',  
                             'ERROR:foo.bar:second message'])

### assertMultiLineEqual

assertMultiLineEqual(  
    first,  
    second,  
    msg=None,  
    \*\*kwargs  
)

Asserts that two multi-line strings are equal.

### assertNDArrayNear

assertNDArrayNear(  
    ndarray1,  
    ndarray2,  
    err,  
    msg=None  
)

Asserts that two numpy arrays have near values.

#### Args:

* **ndarray1**: a numpy ndarray.
* **ndarray2**: a numpy ndarray.
* **err**: a float. The maximum absolute difference allowed.
* **msg**: Optional message to report on failure.

### assertNear

assertNear(  
    f1,  
    f2,  
    err,  
    msg=None  
)

Asserts that two floats are near each other.

Checks that |f1 - f2| < err and asserts a test failure if not.

#### Args:

* **f1**: A float value.
* **f2**: A float value.
* **err**: A float value.
* **msg**: An optional string message to append to the failure message.

### assertNoCommonElements

assertNoCommonElements(  
    expected\_seq,  
    actual\_seq,  
    msg=None  
)

Checks whether actual iterable and expected iterable are disjoint.

### assertNotAllClose

assertNotAllClose(  
    a,  
    b,  
    \*\*kwargs  
)

Assert that two numpy arrays, or Tensors, do not have near values.

#### Args:

* **a**: the first value to compare.
* **b**: the second value to compare.
* **\*\*kwargs**: additional keyword arguments to be passed to the underlying assertAllClose call.

#### Raises:

* **AssertionError**: If a and b are unexpectedly close at all elements.

### assertNotAlmostEqual

assertNotAlmostEqual(  
    first,  
    second,  
    places=None,  
    msg=None,  
    delta=None  
)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

### assertNotAlmostEquals

assertNotAlmostEquals(  
    \*args,  
    \*\*kwargs  
)

### assertNotEmpty

assertNotEmpty(  
    container,  
    msg=None  
)

Asserts that an object has non-zero length.

#### Args:

* **container**: Anything that implements the collections.Sized interface.
* **msg**: Optional message to report on failure.

### assertNotEndsWith

assertNotEndsWith(  
    actual,  
    unexpected\_end,  
    msg=None  
)

Asserts that actual.endswith(unexpected\_end) is False.

#### Args:

* **actual**: str
* **unexpected\_end**: str
* **msg**: Optional message to report on failure.

### assertNotEqual

assertNotEqual(  
    first,  
    second,  
    msg=None  
)

Fail if the two objects are equal as determined by the '!=' operator.

### assertNotEquals

assertNotEquals(  
    \*args,  
    \*\*kwargs  
)

### assertNotIn

assertNotIn(  
    member,  
    container,  
    msg=None  
)

Just like self.assertTrue(a not in b), but with a nicer default message.

### assertNotIsInstance

assertNotIsInstance(  
    obj,  
    cls,  
    msg=None  
)

Included for symmetry with assertIsInstance.

### assertNotRegex

assertNotRegex(  
    text,  
    unexpected\_regex,  
    msg=None  
)

Fail the test if the text matches the regular expression.

### assertNotStartsWith

assertNotStartsWith(  
    actual,  
    unexpected\_start,  
    msg=None  
)

Asserts that actual.startswith(unexpected\_start) is False.

#### Args:

* **actual**: str
* **unexpected\_start**: str
* **msg**: Optional message to report on failure.

### assertProtoEquals

assertProtoEquals(  
    expected\_message\_maybe\_ascii,  
    message,  
    msg=None  
)

Asserts that message is same as parsed expected\_message\_ascii.

Creates another prototype of message, reads the ascii message into it and then compares them using self.\_AssertProtoEqual().

#### Args:

* **expected\_message\_maybe\_ascii**: proto message in original or ascii form.
* **message**: the message to validate.
* **msg**: Optional message to report on failure.

### assertProtoEqualsVersion

assertProtoEqualsVersion(  
    expected,  
    actual,  
    producer=versions.GRAPH\_DEF\_VERSION,  
    min\_consumer=versions.GRAPH\_DEF\_VERSION\_MIN\_CONSUMER,  
    msg=None  
)

### assertRaises

assertRaises(  
    excClass,  
    callableObj=None,  
    \*args,  
    \*\*kwargs  
)

Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this::

 with self.assertRaises(SomeException):  
     do\_something()

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion::

with self.assertRaises(SomeException) as cm:  
    do\_something()  
the\_exception = cm.exception  
self.assertEqual(the\_exception.error\_code, 3)

### assertRaisesOpError

assertRaisesOpError(expected\_err\_re\_or\_predicate)

### assertRaisesRegex

assertRaisesRegex(  
    expected\_exception,  
    expected\_regex,  
    callable\_obj=None,  
    \*args,  
    \*\*kwargs  
)

Asserts that the message in a raised exception matches a regex.

#### Args:

* **expected\_exception**: Exception class expected to be raised.
* **expected\_regex**: Regex (re pattern object or string) expected to be found in error message.
* **callable\_obj**: Function to be called.
* **msg**: Optional message used in case of failure. Can only be used when assertRaisesRegex is used as a context manager.
* **args**: Extra args.
* **kwargs**: Extra kwargs.

### assertRaisesRegexp

assertRaisesRegexp(  
    expected\_exception,  
    expected\_regex,  
    callable\_obj=None,  
    \*args,  
    \*\*kwargs  
)

Asserts that the message in a raised exception matches a regex.

#### Args:

* **expected\_exception**: Exception class expected to be raised.
* **expected\_regex**: Regex (re pattern object or string) expected to be found in error message.
* **callable\_obj**: Function to be called.
* **msg**: Optional message used in case of failure. Can only be used when assertRaisesRegex is used as a context manager.
* **args**: Extra args.
* **kwargs**: Extra kwargs.

### assertRaisesWithLiteralMatch

assertRaisesWithLiteralMatch(  
    expected\_exception,  
    expected\_exception\_message,  
    callable\_obj=None,  
    \*args,  
    \*\*kwargs  
)

Asserts that the message in a raised exception equals the given string.

Unlike assertRaisesRegex, this method takes a literal string, not a regular expression.

with self.assertRaisesWithLiteralMatch(ExType, 'message'): DoSomething()

#### Args:

* **expected\_exception**: Exception class expected to be raised.
* **expected\_exception\_message**: String message expected in the raised exception. For a raise exception e, expected\_exception\_message must equal str(e).
* **callable\_obj**: Function to be called, or None to return a context.
* **\*args**: Extra args.
* **\*\*kwargs**: Extra kwargs.

#### Returns:

A context manager if callable\_obj is None. Otherwise, None.

#### Raises:

self.failureException if callable\_obj does not raise a matching exception.

### assertRaisesWithPredicateMatch

assertRaisesWithPredicateMatch(  
    \*args,  
    \*\*kwds  
)

Returns a context manager to enclose code expected to raise an exception.

If the exception is an OpError, the op stack is also included in the message predicate search.

#### Args:

* **exception\_type**: The expected type of exception that should be raised.
* **expected\_err\_re\_or\_predicate**: If this is callable, it should be a function of one argument that inspects the passed-in exception and returns True (success) or False (please fail the test). Otherwise, the error message is expected to match this regular expression partially.

#### Returns:

A context manager to surround code that is expected to raise an exception.

### assertRegex

assertRegex(  
    text,  
    expected\_regex,  
    msg=None  
)

Fail the test unless the text matches the regular expression.

### assertRegexMatch

assertRegexMatch(  
    actual\_str,  
    regexes,  
    message=None  
)

Asserts that at least one regex in regexes matches str.

If possible you should use assertRegex, which is a simpler version of this method. assertRegextakes a single regular expression (a string or re compiled object) instead of a list.

#### Notes:

1. This function uses substring matching, i.e. the matching succeeds if any substring of the error message matches any regex in the list. This is more convenient for the user than full-string matching.
2. If regexes is the empty list, the matching will always fail.
3. Use regexes=[''] for a regex that will always pass.
4. '.' matches any single character except the newline. To match any character, use '(.|\n)'.
5. '^' matches the beginning of each line, not just the beginning of the string. Similarly, '$' matches the end of each line.
6. An exception will be thrown if regexes contains an invalid regex.

#### Args:

* **actual\_str**: The string we try to match with the items in regexes.
* **regexes**: The regular expressions we want to match against str. See "Notes" above for detailed notes on how this is interpreted.
* **message**: The message to be printed if the test fails.

### assertRegexpMatches

assertRegexpMatches(  
    \*args,  
    \*\*kwargs  
)

### assertSameElements

assertSameElements(  
    expected\_seq,  
    actual\_seq,  
    msg=None  
)

Asserts that two sequences have the same elements (in any order).

This method, unlike assertCountEqual, doesn't care about any duplicates in the expected and actual sequences.

assertSameElements([1, 1, 1, 0, 0, 0], [0, 1]) # Doesn't raise an AssertionError

If possible, you should use assertCountEqual instead of assertSameElements.

#### Args:

* **expected\_seq**: A sequence containing elements we are expecting.
* **actual\_seq**: The sequence that we are testing.
* **msg**: The message to be printed if the test fails.

### assertSameStructure

assertSameStructure(  
    a,  
    b,  
    aname='a',  
    bname='b',  
    msg=None  
)

Asserts that two values contain the same structural content.

The two arguments should be data trees consisting of trees of dicts and lists. They will be deeply compared by walking into the contents of dicts and lists; other items will be compared using the == operator. If the two structures differ in content, the failure message will indicate the location within the structures where the first difference is found. This may be helpful when comparing large structures.

Mixed Sequence and Set types are supported. Mixed Mapping types are supported, but the order of the keys will not be considered in the comparison.

#### Args:

* **a**: The first structure to compare.
* **b**: The second structure to compare.
* **aname**: Variable name to use for the first structure in assertion messages.
* **bname**: Variable name to use for the second structure.
* **msg**: Additional text to include in the failure message.

### assertSequenceAlmostEqual

assertSequenceAlmostEqual(  
    expected\_seq,  
    actual\_seq,  
    places=None,  
    msg=None,  
    delta=None  
)

An approximate equality assertion for ordered sequences.

Fail if the two sequences are unequal as determined by their value differences rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between each value in the two sequences is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two sequences compare equal then they will automatically compare almost equal.

#### Args:

* **expected\_seq**: A sequence containing elements we are expecting.
* **actual\_seq**: The sequence that we are testing.
* **places**: The number of decimal places to compare.
* **msg**: The message to be printed if the test fails.
* **delta**: The OK difference between compared values.

### assertSequenceEqual

assertSequenceEqual(  
    seq1,  
    seq2,  
    msg=None,  
    seq\_type=None  
)

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

#### Args:

* **seq1**: The first sequence to compare.
* **seq2**: The second sequence to compare.
* **seq\_type**: The expected datatype of the sequences, or None if no datatype should be enforced.
* **msg**: Optional message to use on failure instead of a list of differences.

### assertSequenceStartsWith

assertSequenceStartsWith(  
    prefix,  
    whole,  
    msg=None  
)

An equality assertion for the beginning of ordered sequences.

If prefix is an empty sequence, it will raise an error unless whole is also an empty sequence.

If prefix is not a sequence, it will raise an error if the first element of whole does not match.

#### Args:

* **prefix**: A sequence expected at the beginning of the whole parameter.
* **whole**: The sequence in which to look for prefix.
* **msg**: Optional message to report on failure.

### assertSetEqual

assertSetEqual(  
    set1,  
    set2,  
    msg=None  
)

A set-specific equality assertion.

#### Args:

* **set1**: The first set to compare.
* **set2**: The second set to compare.
* **msg**: Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

### assertShapeEqual

assertShapeEqual(  
    np\_array,  
    tf\_tensor,  
    msg=None  
)

Asserts that a Numpy ndarray and a TensorFlow tensor have the same shape.

#### Args:

* **np\_array**: A Numpy ndarray or Numpy scalar.
* **tf\_tensor**: A Tensor.
* **msg**: Optional message to report on failure.

#### Raises:

* **TypeError**: If the arguments have the wrong type.

### assertStartsWith

assertStartsWith(  
    actual,  
    expected\_start,  
    msg=None  
)

Assert that actual.startswith(expected\_start) is True.

#### Args:

* **actual**: str
* **expected\_start**: str
* **msg**: Optional message to report on failure.

### assertTotallyOrdered

assertTotallyOrdered(  
    \*groups,  
    \*\*kwargs  
)

Asserts that total ordering has been implemented correctly.

For example, say you have a class A that compares only on its attribute x. Comparators other than **lt**are omitted for brevity.

class A(object): def **init**(self, x, y): self.x = x self.y = y

def **hash**(self): return hash(self.x)

def **lt**(self, other): try: return self.x < other.x except AttributeError: return NotImplemented

assertTotallyOrdered will check that instances can be ordered correctly. For example,

self.assertTotallyOrdered( [None], # None should come before everything else. [1], # Integers sort earlier. [A(1, 'a')], [A(2, 'b')], # 2 is after 1. [A(3, 'c'), A(3, 'd')], # The second argument is irrelevant. [A(4, 'z')], ['foo']) # Strings sort last.

#### Args:

* **\*groups**: A list of groups of elements. Each group of elements is a list of objects that are equal. The elements in each group must be less than the elements in the group after it. For example, these groups are totally ordered: [None], [1], [2, 2], [3]. \*\*kwargs: optional msg keyword argument can be passed.

### assertTrue

assertTrue(  
    expr,  
    msg=None  
)

Check that the expression is true.

### assertTupleEqual

assertTupleEqual(  
    tuple1,  
    tuple2,  
    msg=None  
)

A tuple-specific equality assertion.

#### Args:

* **tuple1**: The first tuple to compare.
* **tuple2**: The second tuple to compare.
* **msg**: Optional message to use on failure instead of a list of differences.

### assertUrlEqual

assertUrlEqual(  
    a,  
    b,  
    msg=None  
)

Asserts that urls are equal, ignoring ordering of query params.

### assertWarns

assertWarns(  
    expected\_warning,  
    callable\_obj=None,  
    \*args,  
    \*\*kwargs  
)

Fail unless a warning of class warnClass is triggered by callable\_obj when invoked with arguments args and keyword arguments kwargs. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with callable\_obj omitted or None, will return a context object used like this::

 with self.assertWarns(SomeWarning):  
     do\_something()

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion::

with self.assertWarns(SomeWarning) as cm:  
    do\_something()  
the\_warning = cm.warning  
self.assertEqual(the\_warning.some\_attribute, 147)

### assertWarnsRegex

assertWarnsRegex(  
    expected\_warning,  
    expected\_regex,  
    callable\_obj=None,  
    \*args,  
    \*\*kwargs  
)

Asserts that the message in a triggered warning matches a regexp. Basic functioning is similar to assertWarns() with the addition that only warnings whose messages also match the regular expression are considered successful matches.

#### Args:

* **expected\_warning**: Warning class expected to be triggered.
* **expected\_regex**: Regex (re pattern object or string) expected to be found in error message.
* **callable\_obj**: Function to be called.
* **msg**: Optional message used in case of failure. Can only be used when assertWarnsRegex is used as a context manager.
* **args**: Extra args.
* **kwargs**: Extra kwargs.

### assert\_

assert\_(  
    \*args,  
    \*\*kwargs  
)

### cached\_session

cached\_session(  
    \*args,  
    \*\*kwds  
)

Returns a TensorFlow Session for use in executing tests.

This method behaves differently than self.session(): for performance reasons cached\_session will by default reuse the same session within the same test. The session returned by this function will only be closed at the end of the test (in the TearDown function).

Use the use\_gpu and force\_gpu options to control where ops are run. If force\_gpu is True, all ops are pinned to /device:GPU:0. Otherwise, if use\_gpu is True, TensorFlow tries to run as many ops on the GPU as possible. If both force\_gpu anduse\_gpu` are False, all ops are pinned to the CPU.

#### Example:

class MyOperatorTest(test\_util.TensorFlowTestCase):  
  def testMyOperator(self):  
    with self.cached\_session(use\_gpu=True) as sess:  
      valid\_input = [1.0, 2.0, 3.0, 4.0, 5.0]  
      result = MyOperator(valid\_input).eval()  
      self.assertEqual(result, [1.0, 2.0, 3.0, 5.0, 8.0]  
      invalid\_input = [-1.0, 2.0, 7.0]  
      with self.assertRaisesOpError("negative input not supported"):  
        MyOperator(invalid\_input).eval()

#### Args:

* **graph**: Optional graph to use during the returned session.
* **config**: An optional config\_pb2.ConfigProto to use to configure the session.
* **use\_gpu**: If True, attempt to run as many ops as possible on GPU.
* **force\_gpu**: If True, pin all ops to /device:GPU:0.

#### Yields:

A Session object that should be used as a context manager to surround the graph building and execution code in a test case.

### captureWritesToStream

captureWritesToStream(  
    \*args,  
    \*\*kwds  
)

A context manager that captures the writes to a given stream.

This context manager captures all writes to a given stream inside of a CapturedWrites object. When this context manager is created, it yields the CapturedWrites object. The captured contents can be accessed by calling .contents() on the CapturedWrites.

For this function to work, the stream must have a file descriptor that can be modified using os.dupand os.dup2, and the stream must support a .flush() method. The default python sys.stdout and sys.stderr are examples of this. Note that this does not work in Colab or Jupyter notebooks, because those use alternate stdout streams.

#### Example:

class MyOperatorTest(test\_util.TensorFlowTestCase):  
  def testMyOperator(self):  
    input = [1.0, 2.0, 3.0, 4.0, 5.0]  
    with self.captureWritesToStream(sys.stdout) as captured:  
      result = MyOperator(input).eval()  
    self.assertStartsWith(captured.contents(), "This was printed.")

#### Args:

* **stream**: The stream whose writes should be captured. This stream must have a file descriptor, support writing via using that file descriptor, and must have a .flush() method.

#### Yields:

A CapturedWrites object that contains all writes to the specified stream made during this context.

### checkedThread

checkedThread(  
    target,  
    args=None,  
    kwargs=None  
)

Returns a Thread wrapper that asserts 'target' completes successfully.

This method should be used to create all threads in test cases, as otherwise there is a risk that a thread will silently fail, and/or assertions made in the thread will not be respected.

#### Args:

* **target**: A callable object to be executed in the thread.
* **args**: The argument tuple for the target invocation. Defaults to ().
* **kwargs**: A dictionary of keyword arguments for the target invocation. Defaults to {}.

#### Returns:

A wrapper for threading.Thread that supports start() and join() methods.

### countTestCases

countTestCases()

### create\_tempdir

create\_tempdir(  
    name=None,  
    cleanup=None  
)

Create a temporary directory specific to the test.

NOTE: The directory and its contents will be recursively cleared before creation. This ensures that there is no pre-existing state.

This creates a named directory on disk that is isolated to this test, and will be properly cleaned up by the test. This avoids several pitfalls of creating temporary directories for test purposes, as well as makes it easier to setup directories and verify their contents.

See also: create\_tempfile() for creating temporary files.

#### Args:

* **name**: Optional name of the directory. If not given, a unique name will be generated and used.
* **cleanup**: Optional cleanup policy on when/if to remove the directory (and all its contents) at the end of the test. If None, then uses self.tempfile\_cleanup.

#### Returns:

A \_TempDir representing the created directory.

### create\_tempfile

create\_tempfile(  
    file\_path=None,  
    content=None,  
    mode='w',  
    encoding='utf8',  
    errors='strict',  
    cleanup=None  
)

Create a temporary file specific to the test.

This creates a named file on disk that is isolated to this test, and will be properly cleaned up by the test. This avoids several pitfalls of creating temporary files for test purposes, as well as makes it easier to setup files, their data, read them back, and inspect them when a test fails.

NOTE: This will zero-out the file. This ensures there is no pre-existing state.

See also: create\_tempdir() for creating temporary directories.

#### Args:

* **file\_path**: Optional file path for the temp file. If not given, a unique file name will be generated and used. Slashes are allowed in the name; any missing intermediate directories will be created. NOTE: This path is the path that will be cleaned up, including any directories in the path, e.g., 'foo/bar/baz.txt' will rm -r foo.
* **content**: Optional string or bytes to initially write to the file. If not specified, then an empty file is created.
* **mode**: Mode string to use when writing content. Only used if content is non-empty.
* **encoding**: Encoding to use when writing string content. Only used if content is text.
* **errors**: How to handle text to bytes encoding errors. Only used if content is text.
* **cleanup**: Optional cleanup policy on when/if to remove the directory (and all its contents) at the end of the test. If None, then uses self.tempfile\_cleanup.

#### Returns:

A \_TempFile representing the created file.

### debug

debug()

Run the test without collecting errors in a TestResult

### defaultTestResult

defaultTestResult()

### doCleanups

doCleanups()

Execute all cleanup functions. Normally called for you after tearDown.

### evaluate

evaluate(tensors)

Evaluates tensors and returns numpy values.

#### Args:

* **tensors**: A Tensor or a nested list/tuple of Tensors.

#### Returns:

tensors numpy values.

### fail

fail(  
    msg=None,  
    prefix=None  
)

Fail immediately with the given message, optionally prefixed.

### failIf

failIf(  
    \*args,  
    \*\*kwargs  
)

### failIfAlmostEqual

failIfAlmostEqual(  
    \*args,  
    \*\*kwargs  
)

### failIfEqual

failIfEqual(  
    \*args,  
    \*\*kwargs  
)

### failUnless

failUnless(  
    \*args,  
    \*\*kwargs  
)

### failUnlessAlmostEqual

failUnlessAlmostEqual(  
    \*args,  
    \*\*kwargs  
)

### failUnlessEqual

failUnlessEqual(  
    \*args,  
    \*\*kwargs  
)

### failUnlessRaises

failUnlessRaises(  
    \*args,  
    \*\*kwargs  
)

### get\_temp\_dir

get\_temp\_dir()

Returns a unique temporary directory for the test to use.

If you call this method multiple times during in a test, it will return the same folder. However, across different runs the directories will be different. This will ensure that across different runs tests will not be able to pollute each others environment. If you need multiple unique directories within a single test, you should use tempfile.mkdtemp as follows: tempfile.mkdtemp(dir=self.get\_temp\_dir()):

#### Returns:

string, the path to the unique temporary directory created for this test.

### id

id()

### run

run(result=None)

### session

session(  
    \*args,  
    \*\*kwds  
)

Returns a TensorFlow Session for use in executing tests.

Note that this will set this session and the graph as global defaults.

Use the use\_gpu and force\_gpu options to control where ops are run. If force\_gpu is True, all ops are pinned to /device:GPU:0. Otherwise, if use\_gpu is True, TensorFlow tries to run as many ops on the GPU as possible. If both force\_gpu anduse\_gpu` are False, all ops are pinned to the CPU.

#### Example:

class MyOperatorTest(test\_util.TensorFlowTestCase):  
  def testMyOperator(self):  
    with self.session(use\_gpu=True):  
      valid\_input = [1.0, 2.0, 3.0, 4.0, 5.0]  
      result = MyOperator(valid\_input).eval()  
      self.assertEqual(result, [1.0, 2.0, 3.0, 5.0, 8.0]  
      invalid\_input = [-1.0, 2.0, 7.0]  
      with self.assertRaisesOpError("negative input not supported"):  
        MyOperator(invalid\_input).eval()

#### Args:

* **graph**: Optional graph to use during the returned session.
* **config**: An optional config\_pb2.ConfigProto to use to configure the session.
* **use\_gpu**: If True, attempt to run as many ops as possible on GPU.
* **force\_gpu**: If True, pin all ops to /device:GPU:0.

#### Yields:

A Session object that should be used as a context manager to surround the graph building and execution code in a test case.

### setUp

setUp()

### setUpClass

setUpClass(cls)

Hook method for setting up class fixture before running tests in the class.

### shortDescription

shortDescription()

Formats both the test method name and the first line of its docstring.

If no docstring is given, only returns the method name.

This method overrides unittest.TestCase.shortDescription(), which only returns the first line of the docstring, obscuring the name of the test upon failure.

#### Returns:

* **desc**: A short description of a test method.

### skipTest

skipTest(reason)

Skip this test.

### subTest

subTest(  
    \*args,  
    \*\*kwds  
)

Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

### tearDown

tearDown()

### tearDownClass

tearDownClass(cls)

Hook method for deconstructing the class fixture after running all tests in the class.

### test\_session

test\_session(  
    graph=None,  
    config=None,  
    use\_gpu=False,  
    force\_gpu=False  
)

Use cached\_session instead. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use **self.session()** or **self.cached\_session()** instead.

## Class Members

* longMessage = True
* maxDiff = 1600
* tempfile\_cleanup

# tf.test.TestCase.failureException

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase/failureException#top_of_page)
* [Class failureException](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase/failureException#class_failureexception)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase/failureException#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test/TestCase/failureException#__init__)

## Class failureException

Assertion failed.

### Aliases:

* Class tf.compat.v1.test.TestCase.failureException
* Class tf.compat.v2.test.TestCase.failureException
* Class tf.test.TestCase.failureException

## \_\_init\_\_

\_\_init\_\_(  
    \*args,  
    \*\*kwargs  
)