Quora Challenge Documentation

Release 1.0.1

['Tony Flury']

Oct 18, 2019

Contents

Welcome to the User and contributor documentation for the Quora Python Challenge Framework.

This framework supports the programming challenges as published from time to time in either Python Beginners or Python Programming spaces on Quora.

It is entirely possible to complete the challenges without using this framework, though the framework does provide a few useful features:

- A function *describe(challenge_name,..)* which provides easy to access description of the challenge, without needing to access the original post on Quora.
- A decorator *Autotest(challenge_name,..)* which will automatically test the function being decorated and report both test failures and any execptions raised by the function.

For a short tutorial of how to use the framework see *Gettng Started*

Note: The Quora Challenge Framework is only supported on Python 3.6 onwards. Versions for Python 3.5 and earlier, and Python 2 do not exist and are highly unlikely to be produced.

CHAPTER 1

Suggested reading order

For someone undertaking the challenges :

1.1 Installation

Installation is very simple :

```
$ pip install quorachallenge
```

1.2 Getting Started - Using the Framework

The Quora challenge framework is very easy to use and can do 3 main useful things :

- Get the challenge Description
- Automatically test your function

Note: The quorachallenge framework relies heavily on being able to download data and descriptions from the public internet, so you need to ensure you have a working network connection when using the framework.

Note: This documentation regularly uses the name '**challenge1**' as the name of the challenge being solved and tested. This name is just an example for the sake of the documentation. The challenge name you need to use for any given challenge will be stated clearly on the post on Quora that describes the challenge - so make sure you use the right name. When new challenges are posted there will be no need to install a new version of the framework - just use the new name.

1.2.1 To get the challenge Description

To get a description of the challenge (in this case 'challenge1') use the *describe function*.

```
>>> import quorachallenge
>>> quorachallenge.describe('challenge1')
```

This code snippet will open a new web browser tab or window and display the description of the challenge.

1.2.2 Automatically test your function

To test your function which you think solves a given challenge use the *AutoTest Decorator*.

Imagine we have written function `my_func` to solve 'challenge1'

```
import quorachallenge

@quorachallenge.autotest('challenge1')
def my_func(a, b):
    return a+b
```

This code will automatically test my_func against all of the test cases defined for this challenge, and report any errors or exceptions that occur during the testing. The displayed messages will identify the test case id that failed and some details of the failure.

The *AutoTest Decorator* can also be used in a more indirect way as shown in this example

```
>>> import quorachallenge as qc
        ...
>>> tester = qc.autotest('challenge1', defer_results=True)
>>> passed = tester(func)
>>> passed
False
>>> print('\n'.join(tester.errors))
Test 500 - Incorrect result : Expected (0) != Returned (1)
>>> tester.results('500')
'Test 500 - Incorrect result : Expected (0) != Returned (1)'
```

The *AutoTest Decorator* can also be passed a specific test id, so as to repeat just that single test case. This can be very useful in debugging failures.

1.2.3 Display testdata

To display the test data for a given challenge use the *testdata function*.

```
>>> import quorachallenge
>>> quorachallenge.testdata('challenge1',test_id='500')
------
Id : 500
Called as : your_function(0,0)
Expected to return : 0
```

This code snippet will display all the test data that exists for 'challenge1'. There is an optional 2nd argument to the `testdata` function that allows you to select the test data for just a single test case :

```
import quorachallenge

quorachallenge.testdata('challenge1', '1')
```

This code snippet will display just the test data for the test case with id ‘1’; this will be useful if your code is failing one particular test case.

When looking at the test data as displayed from this function :

Id The id of this tests case

called as how the function will be called for this test case

Raises The Exception that the function should raise under this test case

Returns The expected return value for this test case

For any given test case, there will be either an expected Return value, or an expected Exception listed but not both.

1.3 Framework Details

1.3.1 Overview

The quorachallenge module is a framework to assist developers to are taking part in the Quora Python challenges which will posted ocassionally in Python Beginners or Python Programming Quora spaces. The challenges will all require the implementation of a Python function which will meet the specific interface and functionality as required.

The framework provides:

- immediate access to the description of the challenge
- the ability to automatically test the function written as an entry to the challenge
- easy to read test data

1.3.2 Functions

`quorachallenge.`**`describe`**(*challenge_name: str*, *webpage: bool = True*, *_directory: str = None*)

Display the Description of this challenge

Parameters

- **`challenge_name`** (*`str`*) – The lower case name for the challenge.
- **`webpage`** (*`bool`*) – When True the description is displayed in a tab in the browser. When False the text is returned in a raw form - rst formatted.
- **`_directory`** (*`str`*) – The base local directory to search for this challenge in. Implemented to allow testing of challenges before publication. For Contributor use only.

In normal use the test data is downloaded from a remote site, so this function requires an active public internet connection.

`quorachallenge.`**`testdata`**(*challenge_name: str*, *test_id: str = None*, *_directory: str = None*)

Display the test data for the given challenge

Parameters

- **`challenge_name`** (*`str`*) – The lower case name for the challenge.

- **test_id** (*str*) – The test test_id. If left as the default value this function will display the data for all of the test cases. If it is not None, then this function will display the data for the test case with that test_id.
- **_directory** (*str*) – The base local directory to search for this challenge in. Implemented to allow testing of challenges before publication. For Contributor use only.

In normal use the test data is downloaded from a remote site, so this function requires an active public internet connection.

1.3.3 autotest class

class quorachallenge.**autotest**(*challenge_name*, *test_id: str = None*, *defer_results: bool = None*, *_directory: str = None*)

A callable class/decorator, which will automatically test the function against the challenge requirements. By default will immediately report any errors and unexpected exceptions that are raised by the function that is decorated.

Parameters

- **challenge_name** (*str*) – The case insensitive name for the challenge.
- **test_id** (*str*) – A specific test_id to execute.
- **defer_results** (*bool*) – When False the decorator will immediately report test errors and unexpected exceptions upon completion of the automatic testing. When True the function will be automatically tested but test failures and exceptions are recorded but not automatically reported.
- **_directory** (*str*) – The base local directory to search for this challenge in. Implemented to allow testing of challenges before publication. For Contributor use only.

When defer_results is True, the test failures and exceptions are accessed via the *errors* and *exceptions* properties. and the *passed property* provides a simple True/False report on whether the requested tests completed without errors or unexpected exceptions.

In normal use the test data for the challenge is downloaded from a remote site, so using this function requires an active public internet connection.

Examples :

As a decorator:

```
@quorachallenge.solves('dummy_challenge')
def dummy( a, b):
    return a +b
```

Using an explicit instance of the class

```
def dummy( a, b):
    return a +b

solver=quorachallenge.solves('dummy_challenge', defer_results=True)
passed = solver(dummy)

if not passed:
    print(solver.errors)
    print(solver.exceptions)
```

`autotest.`**`results`**`(`*test_id*`)`

Return the results for this test test_id if executed

Parameters **`test_id`** (*`str`*) – The test test_id to be queried

This methods returns a text string which indicates whether the given test test_id passed successfully, resulted in an return value error, or an unexpected exception.

Returns an empty string if this test test_id has not been executed.

This documentation set also describes how experienced developers can submit new challenges :

1.4 Developer Guide - Overview

1.4.1 Introduction

This Developer Guide documents how experienced developers can set their own challenges for either Python Beginners or Python Programming readers, such that the readers can use this framework to confirm that their code successfully completes the challenge.

Every challenge will require the user to implement a single function, which takes one or more inputs (either positional or keyword or both) and will return one or more values. As the person setting the challenge your role is the clearly described the requirements you expect the entrants to meet, and define a set of test data that is sufficient to show that the entrants code meets the requirments.

1.4.2 Required Contents

In order to provide a programming challenge under this framework you will need to provide

- description.rst - An English language description of the challenge; a clear definition of the functionality expected formatted in rST structure. The description should state any exceptions that the code should raise. See *Description Guidelines* for more information.
- testdata.json - A json file that defines the data sets which will be passed to the function in order to confirm that the function meets the description. See *Test Data Guidelines* for details on how to write this critical file. Note that errors in this file will be presented to the user during testing of their code, and those errors are likely to confuse challenge entrants.
- An examplar solution to the challenge - this is not published but is required to *test your challenge information* before publication.

Please read the documentation while developing this content.

1.4.3 Pre requisites

Apart from a good knowledge of Python, there is a short list of pre-requisites

- You are a known contributor to either Python Beginners or Python Programming spaces on Quora
- You have git on your developement machine
- you have an account on Git Hub - and are familiar with cloning, forking, branches and pull requests.

1.4.4 Process Overview

There is a simple 9 step process to provide your own challenges :

1. Create a GIT fork of QuoraChallengesTestData and clone that fork to your local computer.
2. Create a new developement branch in your local copy.
3. Create a new folder at the top level of the repository - the name of folder is the chosen name for the challenge. The folder name must be in lowercase.
4. Within the new folder create the following files *description.rst*, *testdata.json* and any optional files that your challenge requires.
5. Edit the Readme.rst in the top level directory to add your challenge to the list.
6. Write your exemplar solution, and test your solution and your testdata.
7. Once all of the requirements are complete, commit and push the branch to your git hub repository. Do not commit your exemplar solution.
8. Request a pull request to merge your branch into the main repository.
9. Once the merge has been completed - publicise your challenge in the appropriate space.

To make it clear - these are the changes and new files that are required in order to create a new challenge :

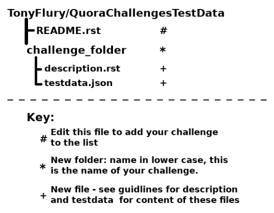
TonyFlury/QuoraChallengesTestData
README.rst #
challenge_folder *
description.rst +
testdata.json +
Key:
# Edit this file to add your challenge to the list
* New folder: name in lower case, this is the name of your challenge.
+ New file - see guidlines for description and testdata for content of these files

Fig. 1: File Changes required in the QuoraChallengesTestData repository.

Warning: Please do not publicise your challenge until the merge is complete. Until the merge completes, the testdata and description will not be available to the public.

1.4.5 Likely Audiences

When developing your challenge, please keep in mind at all times the target audience.

- The Python Beginners space is intended for those developers who are just starting out with Python and likely just starting out with development. It is likely that they wont be up to speed with most data structure and algorithms.

- Challenges aimed at the developers in Python Programming (who generally have skill level intermediate and beyond) will be much less common and the challenges should be open to multiple styles of solution, using data structures and algorithms

1.4.6 Key Additional Information

Make sure you are familiar with these additional guidelines in order to complete your challenge.

- *Writing your description*
- *Writing your test data*
- *Testing your challenge test data*

Providing a challenge Description

The description file must be a clear and unambiguous description of the challenge, written in rST format,

It is strongly recommended that :

- The challenge should have a clear title
- The remainder of the description should be structured as a Python docstring[1].

  Specifically that is :

  - Following the title, a one sentence summary of challenge.
  - After the short summary a full description should follow if neccessary.
  - Where required the arguments that the function should be listed as a bullet list
  - The description must clearly describe the return value that the function should produce, providing simple examples as neccessary to aid the description.
  - The description must list any exceptions that the function shall raise and under what conditions.

An example in rST format

```
==========================================
Quora Python Beginners - Example Challenge
==========================================

A function that takes two numbers as postional arguments and adds them␣
↪together, returning the result.

The function will take two numbers (integers or floats), and return the two␣
↪numbers added together, with the
following reservations :

 - If either value is zero the result returned should be zero
 - If either value is negative the result should be zero

There is no requirement for the function to test the type of either argument.

Exceptions
----------

The Function is not required to raise any exceptions.
```

[1] https://www.python.org/dev/peps/pep-0257/

This example is also available as the description of the example_challenge from the QuoraChallengesTestData repository.

The description must be provided within the challenge folder in a file named as 'description.rst'.

In order to test that the description is formatted correctly, use the *describe function* from the framework.

Providing test data

Overiew

As part of providing a challenge, you will provide a set of test cases which aims to confirm that the function provided meets the requirements of the challenge. The test data should boundary conditions, any exceptional arguments (Empty strings, None values etc), and of course any argument sets which should cause the function to raise an exception.

Structure

The testdata.json file is a simple array of dictionaries. If you are not familiar with JSON format it is human readable, and is very similar to Python formatting :

- JSON arrays are delimited by square brackets (like Python lists)
- JSON dictionaries are delimited by braces {like Python dictionaries

the main relevant differences are that:

- JSON only uses a double quote character to delimit strings (Python of course can use single or double quotes)
- Neither JSON lists or dictionaries can have trailing commas (unlike their Python equivalents).

Each dictionary within the JSON array represents a single test case, with it's own call arguments, expected return value and ability to catch expected exceptions from the function under test. These are provided by a set of keys within the dictionary.

JSON files do not have a format for comments, but if absolutely neccessary, there is nothing to stop you using the `comment` key in the dictionary to add notes and comments to each test case. The `comment` key is guaranteed to never be used by the framework, and will be displayed by the *test_data function* function

Dictionary Keys

Within each dictionary, the following keys are considered. All other keys are ignored, and the keys are case sensitive (all lower case).

"id" (Optional) The unique id for this for test case - either numeric or a string. If not provided, the id is assumed to be the index in the top-level array.

"arguments" (Mandatory) The arguments passed to the function being tested. A string (delimited by single quotes). This is the text that appears within the parentheses ( ) as the function is being called - see *Arguments in Detail* for more information and examples.

"return" (Mandatory) The return value expected from this function when called with the given arguments. A string delimited by single quotes. This is the repr of the return value. - see *Return value in Detail* for more information and examples. Although this value must be present it is ignored if an exception is raised by the function.

"raises" (Optional) The name of the exception that the function should raise given these arguments. A json string delimited by single quotes. Can be the name of any exception apart from SyntaxError.

Arguments in Detail

The value of the `arguments` key provides the arguments passed to the function being tested. This value is a string, the contents of which is the text that appears inside of the parentheses in a normal function call.

For example :

Single Numeric argument

```
# Actual function call to be executed
the_function(3)
```

Equivalent json arguments key :

```
"arguments" : "3"
```

Multiple Numeric Arguments

```
# Actual function call to be executed
the_function(5, 7.5)
```

Equivalent json arguments key :

```
"arguments" : "5, 7.5"
```

String Arguments

```
# Actual function call to be executed
the_function('Hello World')
```

Equivalent json arguments key :

```
"arguments" : "'Hello World'"
```

Positional and Keyword arguments

```
# Actual function call to be executed
the_function('Hello World', type='bold')
```

Equivalent json arguments key :

```
"arguments" : "'Hello World', type='bold'"
```

The `arguments` key can be used for any combination of possible positional and keyword arguments, passing in any builtin type, including strings, lists, tuples, sets, dictionaries, integers, floats and complex values.

Note: It is currently impossible to pass values which derive from a library that needs to be imported. This limitation might be removed in a future update.

Return value in Detail

The value of the `return` key provides the value which is expected to be returned from the function being tested for the given arguments value. This value is a string, the contents of which is the repr of the return value.

For example :

Single Numeric return value

```
the_function( <arguments> ) == 3
```

Equivalent json return key :

```
"return" : "3"
```

Multiple Numeric return value

```
the_function( <arguments> ) == (3,15)
```

Equivalent json return key :

```
"return" : "(3, 15)"
```

or alternatively

```
"return" : "3, 15"
```

String return value

```
# Actual function call to be executed
the_function( <arguments> ) == 'Hello World'
```

Equivalent json return key :

```
"return" : "'Hello World'"
```

The `return` key can be used for any combination of possible return value of a builtin type, including strings, lists, tuples, sets, dictionaries, integers, floats and complex values.

Note: It is currently impossible to test for return values which derive from a library that needs to be imported. This limitation might be removed in a future update.

Examples

The following is a fully set of test data for a the 'example_challenge'. The description is

```
A function that takes two numbers as postional arguments and adds them together,␣
↪returning the result.

The function will take two numbers (integers or floats), and return the two numbers␣
↪added together, with the
following reservations :

 - If either value is zero the result returned should be zero
 - If either value is negative the result should be zero
 - If both values are negative the function should raise a ValueError exception.

There is no requirement for the function to test the type of either argument.

Exceptions
----------

The Function is required to raise a ValueError exception when both arguments are␣
↪negative.
```

The example test data is to confirm that all conditions have been met (at least for the given test data is :

```
[
{"input":"3,5", "return":"8"},
{"input":"1,2", "return":"3"},
{"input":"9,11", "return":"20"},
{"input":"8,7", "return":"15"},
{"input":"3.5,7.6", "return":"11.1"},
{"input":"2.3,4.87", "return":"7.17"},
{"input":"0,1", "return":"0"},
{"input":"1,0", "return":"0"},
{"input":"-3,18", "return":"0"},
{"input":"12,-5", "return":"0"},
{"input":"-3,-5", "raises":"ValueError"},
]
```

Note: In many cases it would be recommended where possible to write a short script which produces testdata.json file with 100 or more test cases. An example of an auto generated testdata set is provided in example challenge test data. The script which produced this data is provided as testdata_producer.py in example_challenge.

Testing Your challenge Solution and TestData

By default the framework will download the description and data from the QuoraChallengesTestData repository, looking for the relevant folder in that repository. Clearly this only works when challenge information has been published which means that the pull request has been merged into the master branch.

Ideally you need to test your description formats correctly, and that the test data is both formatted correctly and will correctly prove that the Entrants function meets the requirements before attempting the pull request. You can accomplish this by using the `_directory` argument on the *describe function*, *autotest function* and *testdata function*.

The `_directory` argument allows the developer to specify the top-level directory of their challenges under development

- i.e. the parent of the challenge directory that contains the description.rst and testdata.json files.

For example if the directory structure looks like this :

```
~/Development/QuoraChallengesTestData/
  ├README.rst
  my_challenge
    ├description.rst
    └testdata.json
```

then to test your description :

```
>>> import quorachallenge as qc
>>> qc.describe('my_challenge', _directory='~/Development/QuoraChallengesTestData')
```

Similarly, to test your function (called examplar) :

```
import quorachallenge as qc
qc.AutoTest('my_challenge', _directory='~/Development/QuoraChallengesTestData
↪') (examplar)
```

And to display your test data :

```
import quorachallenge as qc
qc.testdata('my_challenge', _directory='~/Development/QuoraChallengesTestData')
```

Note: Every care is taken to try to ensure that this code comes to you bug free. If you do find an error - please report the problem on :

- GitHub Issues
- By email to : Tony Flury

Index