How to run pytest html report -> pytest –html=report.html

NewClassRooms Assignment

Project Setup: Python 3.11.3 & Pytest 7.3.1 installed. For IDE using Pycharm community version built on May 16,2023.
Run Tests: 1)Unzip sent folder. 2)Launch Pycharm. 3)Open Project "newClassRooms" using Pycharm. 4)At the opened project using the built in Pycharm "Terminal" type the following command to run the tests -> "pytest - -html=test_report.html -vv & hit return. 5) As the tests are running the terminal console logs execution output then you execute the following command -> "open test_report.html" to see the captured pytest test cases execution results.

Test Data Setup: See pre condition step defined for each test case and Notes section.

Test Cases Definition:

Test Case	Steps	Expected Results	Notes:
(ActionType: CountByGender) Verify a valid Gender Count is returned as a sorted value from "Highest to Lowest" for each specific gender available in the passed user data.	1) Pre Condition:Use the requirement referenced End point (i.e: https://randomuser.me/api) to setup dynamic random set of user data for each run of a test case. Implement a "get_users(random_number)" function against the above end point to return the required set of user data based on a fixed random number generator within the fixed range of (1,10).	List of user data is setup.	Assumptions : random endpoint returns data results in the expected JSON format. Instead of the dynamic approach a static approach could have been pursued where a sample of the representative user data could have been persisted to a result file set and used instead as the users payload.
	Execute : 1) Request -> Post 2) Against Endpoint -> https://census-toy.nceng.net/prod/toy-census 3) Using payload : {"actionType": "CountByGender", "users" : get_users(random_users)}	Properly formatted JSON response is returned. Example of what to expect in case of single male available in the user data->	Note: don't pass the "top" option and validate that the number of returned results is not limited.

		`{'name': 'male', 'value': 1}`	
	Validate the returned specific action type response body. For this step : 1) Implement a function that will take the same payload data as the request being tested and returns the expected response based on the count of specific gender found in the user data. The specific total gender count is sorted from highest to lowest using key “value. 2)Assert the expected response is equal to the actual request response.	Gotten actual is as expected -	
	Validate (actual response).status_code is as expected.	200	Requirement followup: Shouldn’t the status code be 201 instead of 200?
	Validate (actual response).headers.	Got expected data for the headers data that the test script validates - See note.	Requirement followup: No expected response headers data is specified - (potentially open issue)
	Below sample of currently returned header data : {'Content-Type': 'application/json', 'Content-Length': '55', 'Connection': 'keep-alive', 'Date': 'Mon, 19 Jun 2023 16:53:48 GMT', 'x-amzn-RequestId': 'a41f3fa3-fd3e-41f9-8b10-dc34b3d9a604', 'x-amz-apigw-id': 'Gxo0FG1WoAMFbcw=', 'X-Amzn-Trace-Id': 'Root=1-64908819-0a3ed5730bdedccc699f086f;Sampled=0;lineage=6e69f56a:0', 'X-Cache': 'Miss from cloudfront', 'Via': '1.1 7fa5b1fb7f2b2970294e5477604d07d4.cloudfront.net (CloudFront)', 'X-Amz-Cf-Pop': 'MIA3-C5', 'X-Amz-Cf-Id': 'i4Cr-dVRn-A7GAPETopp6Fq1PbjX4M0U3KUpKw3YUCMuQHArwuUu7A==’}		See above note about expected response header data.

(ActionType: CountByCountry) Verify a valid Country Count is returned as a sorted value from “Highest to Lowest” for each specific country available in the passed user data.	1) Pre Condition: (user data setup) See previous test case - Follow same step. 2) Import countries from pycountry library and use it in function get_country_code() to implement a list of Alpha country codes.	List of user data is setup.	Same as previous test case.
	Execute : 1) Request -> Post 2) Against Endpoint -> https://census-toy.nceng.net/prod/toy-census 3) Using payload : {“actionType”: “CountByCountry”, “users” : get_users(random_users)}	Properly formatted JSON response is returned. Example of what to expect in the case of a single country used twice in the user data -> `{'name': 'IN', 'value': 2}`	Note: don’t pass the “top” option and validate that the number of returned results is not limited.
	Validate (actual response).status_code is as expected.	200	Requirement followup: Shouldn’t the status code be 201 instead of 200?
	Validate (actual response).headers.	Got expected data for the headers data that the test script validates - See note.	Requirement followup: No expected response headers data is specified - (potentially open issue)
	Validate the returned specific action type response body. For this step : 1) Implement a function that will take the same payload data as the request being	Gotten actual is as expected -	

	tested and the list of computed country codes. Then returns the expected response based on the count of specific countries found in the user data. The specific total country count is sorted from highest to lowest using key “value. 2)Assert the expected response is equal to the actual request response.		
	Validate (actual response).status_code is as expected.	200	Requirement followup: Shouldn’t the status code be 201 instead of 200?
	Validate (actual response).headers.	Same results as previous test case.	
(ActionType: CountPasswordComplexity) Verify that a specific user password is returned sorted from highest to lowest based on computed password complexity.	1) Pre Condition: (user data setup) See previous test case - Follow same step. 2) Implement a “get_password_complexity(password)” function that will return the password complexity based on the number of non alpha characters that the specific password contains.	List of user data is setup.	Same as previous test case.
	Execute : 1) Request -> Post 2) Against Endpoint -> https://census-toy.nceng.net/prod/toy-census 3) Using payload : {“actionType”: “CountByPasswordComplexity”, “users” : get_users(random_users)}	Properly formatted JSON response is returned. Example of what to expect in the case of a single user password used twice in the user data -> `{'name': 'camel', 'value': 0}`	Note: don’t pass the “top” option and validate that the number of returned results is not limited.
	Validate (actual response).status_code is as expected.	200	Requirement followup: Shouldn’t the status code be 201 instead of 200?
	Validate (actual response).headers.	Same results as	

		previous test case.	
	Validate the returned specific action type response body. For this step : 1) Implement a function that will take the same payload data as the request being tested and calls the setup function to compute the password complexity for each password found in the user data. Then returns the expected response based on specific user password and its computed complexity sorted from highest to lowest using key “value”. 2)Assert the expected response is equal to the actual request response.	Gotten actual is as expected -	
(“top” option: Verify that when the option is passed in the specific request payload the returned number of results is equal or less than the value of “top”)	1) Pre Condition: (user data setup) See previous test case - Follow same step.	List of user data is setup.	Same as previous test case.
	Case of “top” option set to Non Zero value (ex.: 5)		
	Repeat the “CountByCountry” test case with the following payload {“actionType”: “CountByCountry”, “users” : get_users(random_users), “top”:5}	Same result as the original test case with following exception : Number of returned results is >= top option value (i.e.: 5)	
	Repeat the “CountByGender “ test case with the following payload : Using payload : {“actionType”: “CountByGender”, “users” : get_users(random_users), “top”:5}	Same result as the original test case with following exception : Number of returned results is >= top option value (i.e.: 5)	

	Repeat the “CountByPasswordComplexity“ test case with the following payload : {“actionType”: “CountByPasswordComplexity”, “users” : get_users(random_users), “top”:5}	Same result as the original test case with following exception : Number of returned results is >= top option value (i.e.: 5)	
	Case of “top” option set to Zero value.		
	Repeat each of the above Non Zero top value with top set to 0.	Same results as the cases where top option was not specified (i.e: number of results is not limited)	
Testing for exceptions & Errors.	Tried various error cases looking for properly handling invalid data and no 500 server exception - Below sample of negative use cases I tried.		
	Invalid key or value	JSON decoder error was reported..	Nice if there is a way to intercept the JSON exception and have the end point report an error code and a message.
	Negative value for top option.	Was ignored and results returned as if top was 0 or not passed.	

Execution & Results: Tests results are captured in an html report that can be examined for “Pass” & “Failed” data results - See above **“Run Tests”** section for how to access it. Note: the initial Zipped test folder contains a sample from previous execution that can be examined.

Encountered Issues & Bugs:
Following is a list of potentials bugs discovered by the test script:

1) 1st screen shot : Returned results not sorted by computed value from highest to lowest.
2) 2nd screen shot: Passed top option value not being respected.
3) 3rd screen shot: Name improperly sorting - Note: This case can be argued as the requirement reads to sort on value only from highest to lowest. (unless ambiguity on testing/QA part)
4) 4th screen shot : Another improper sorting case.

Note: Each reported failure needs to be reported as a bug. The reported bug will contain the following section properly defined:

1)Bug Title : (Example - countByCountry Action Not Sorted Properly).

2)Description: When executing the test case for the action type “countByCountry” the sort order for the returned result at times came incorrect.

3)Environment: (See Endpoint URL)

4)Precondition: Execute the specific test case setup to establish the user data set.

5)Steps: Execute ‘countByCountry’ main test case. Important: May to run the test case more than once to observe the issue

6) Observed results : Attach screen shot of the failure showing the error and the data used.

7) Expected results: Returned result to be ordered as per the stated requirement for the specific action type deterministically. Attach a screenshot of the success case.

IMPORTANT: 1) Access the API end point logs for further analyzing /isolating the potential bug. 2) If set up locally run under the debugger and examine the code logic code.

Note: By inference the above noted potential listed issues/bugs are across all the action types. Need to fix first then regress test and continue validating .

Observed Bug Screen Shot:

File | /Users/oozoozi/PycharmProjects/newClassRooms/report.html

5 passed, 0 skipped, 1 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

how all details / Hide all details

Result	Test	Duration	Links
Failed (hide details)	test_api.py::test_CountByCountry_withTopOptions	1.29	

```
def test_CountByCountry_withTopOptions():
        top_option_values = [0, 5]
>       execute_request_withtopoption('countByCountry', top_option_values)

test_api.py:204:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
test_api.py:152: in execute_request_withtopoption
    validate_response(expected_request_response, actual_request_response)
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

expected_response = [{'name': 'ES', 'value': 2}, {'name': 'US', 'value': 1}, {'name': 'UA', 'value': 1}]
actual_response = [{'name': 'US', 'value': 1}, {'name': 'ES', 'value': 2}, {'name': 'UA', 'value': 1}]

    def validate_response(expected_response, actual_response):
        values_actual = []
        values_expected = []
        assert len(actual_response) == len(expected_response)
        for i in range(0,len(actual_response)):
            values_actual.append(actual_response[i]['value'])
            values_expected.append(expected_response[i]['value'])
>       assert values_expected == values_actual
E       assert [2, 1, 1] == [1, 2, 1]
E         At index 0 diff: 2 != 1
E         Full diff:
E         - [1, 2, 1]
E         + [2, 1, 1]

test_api.py:165: AssertionError
 ------------------------------Captured stdout call------------------------------
EXPECTED RESPONSE - SORTED
[{'name': 'ES', 'value': 2}, {'name': 'US', 'value': 1}, {'name': 'UA', 'value': 1}]
PASSED TOP OPTION VALUE -> : 0
ACTUAL REQUEST RESPONSE -> : [{'name': 'US', 'value': 1}, {'name': 'ES', 'value': 2}, {'name': 'UA', 'value': 1}]
```

Passed (show details)	test_api.py::test_countByGender	1.27	

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

actionType = 'countPasswordComplexity', top_option_values = [0, 5]

    def execute_request_withtopoption(actionType, top_option_values):
        is_subset = False
        expected_request_response = []
        country_codes = get_country_code()
        request_payload = create_payload(actionType)
        for i in range(0, len(top_option_values)):
            request_payload['top'] = top_option_values[i]
            if actionType == 'countByGender' :
                expected_request_response = get_expected_gender_response(request_payload)
            elif actionType == 'countPasswordComplexity' :
                expected_request_response = get_expected_countPwdComplexity_response(request_payload)
            elif actionType == 'countByCountry' :
                expected_request_response = get_expected_countByCountry_response(request_payload, country_codes)
            actual_request_response = post_request(request_payload)
            ##print("ACTUAL REQUEST RESPONSE FOR: ", actionType)
            print("PASSED TOP OPTION VALUE -> :", request_payload['top'])
            print("ACTUAL REQUEST RESPONSE -> :", actual_request_response)

            # Validate response
            if top_option_values[i] == 0 :
                assert len(actual_request_response) == len(expected_request_response)
                validate_response(expected_request_response, actual_request_response)
            else:
>               assert len(actual_request_response) <= top_option_values[i]
E               AssertionError: assert 6 <= 5
E                +  where 6 = len([{'name': '187187', 'value': 6}, {'name': 'poohbear', 'value': 0}, {'name': 'vauxhall', 'value': 0}, {'name':
'genius', 'value': 0}, {'name': 'sebastian', 'value': 0}, {'name': 'ooooo', 'value': 0}])

test_api.py:154: AssertionError
 ------------------------------Captured stdout call------------------------------
EXPECTED RESPONSE - SORTED
[{'name': '187187', 'value': 6}, {'name': 'poohbear', 'value': 0}, {'name': 'vauxhall', 'value': 0}, {'name': 'genius', 'value': 0}, {'name':
'sebastian', 'value': 0}, {'name': 'ooooo', 'value': 0}, {'name': 'petunia', 'value': 0}, {'name': 'punkrock', 'value': 0}]
PASSED TOP OPTION VALUE -> : 0
ACTUAL REQUEST RESPONSE -> : [{'name': '187187', 'value': 6}, {'name': 'poohbear', 'value': 0}, {'name': 'vauxhall', 'value': 0}, {'name': 'genius',
'value': 0}, {'name': 'sebastian', 'value': 0}, {'name': 'ooooo', 'value': 0}, {'name': 'petunia', 'value': 0}, {'name': 'punkrock', 'value': 0}]
EXPECTED RESPONSE - SORTED
[{'name': '187187', 'value': 6}, {'name': 'poohbear', 'value': 0}, {'name': 'vauxhall', 'value': 0}, {'name': 'genius', 'value': 0}, {'name':
'sebastian', 'value': 0}, {'name': 'ooooo', 'value': 0}, {'name': 'petunia', 'value': 0}, {'name': 'punkrock', 'value': 0}]
PASSED TOP OPTION VALUE -> : 5
ACTUAL REQUEST RESPONSE -> : [{'name': '187187', 'value': 6}, {'name': 'poohbear', 'value': 0}, {'name': 'vauxhall', 'value': 0}, {'name': 'genius',
'value': 0}, {'name': 'sebastian', 'value': 0}, {'name': 'ooooo', 'value': 0}]
```

```
def test_countByCountry():
        country_codes = get_country_code()
        countByCountry_payload = create_payload('CountByCountry')
        # Create expected response based on passed payload
        expected_countByCountry_response = get_expected_countByCountry_response(countByCountry_payload,country_codes)
        countByCountry_response = post_request(countByCountry_payload)
        print("ACTUAL REQUEST RESPONSE: ")
        print(countByCountry_response)
>       assert expected_countByCountry_response == countByCountry_response
E       AssertionError: assert [{'name': 'FI', 'value': 2}, {'name': 'NO', 'value': 2}, {'name': 'FR', 'value': 1}, {'name': 'DE', 'value': 1},
{'name': 'RS', 'value': 1}, {'name': 'IR', 'value': 1}, {'name': 'NL', 'value': 1}, {'name': 'MX', 'value': 1}] == [{'name': 'NO', 'value': 2},
{'name': 'FI', 'value': 2}, {'name': 'FR', 'value': 1}, {'name': 'DE', 'value': 1}, {'name': 'RS', 'value': 1}, {'name': 'IR', 'value': 1}, {'name':
'NL', 'value': 1}, {'name': 'MX', 'value': 1}]
E         At index 0 diff: {'name': 'FI', 'value': 2} != {'name': 'NO', 'value': 2}
E         Full diff:
E           [
E         +  {'name': 'FI', 'value': 2},
E            {'name': 'NO', 'value': 2},
E         -  {'name': 'FI', 'value': 2},
E            {'name': 'FR', 'value': 1},
E            {'name': 'DE', 'value': 1},
E            {'name': 'RS', 'value': 1},
E            {'name': 'IR', 'value': 1},
E            {'name': 'NL', 'value': 1},
E            {'name': 'MX', 'value': 1},
E           ]

test_api.py:185: AssertionError
 ------------------------------Captured stdout call------------------------------
EXPECTED RESPONSE - SORTED
[{'name': 'FI', 'value': 2}, {'name': 'NO', 'value': 2}, {'name': 'FR', 'value': 1}, {'name': 'DE', 'value': 1}, {'name': 'RS', 'value': 1}, {'name':
'IR', 'value': 1}, {'name': 'NL', 'value': 1}, {'name': 'MX', 'value': 1}]
ACTUAL REQUEST RESPONSE:
[{'name': 'NO', 'value': 2}, {'name': 'FI', 'value': 2}, {'name': 'FR', 'value': 1}, {'name': 'DE', 'value': 1}, {'name': 'RS', 'value': 1}, {'name':
'IR', 'value': 1}, {'name': 'NL', 'value': 1}, {'name': 'MX', 'value': 1}]
```

Failed *(hide details)*	test_api.py::test_CountByCountry_withTopOptions	1.22	

```
def test_CountByCountry_withTopOptions():
        top_option_values = [0, 5]
>       execute_request_withtopoption('countByCountry', top_option_values)

test_api.py:204:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
test_api.py:152: in execute_request_withtopoption
    validate_response(expected_request_response, actual_request_response)
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

expected_response = [{'name': 'AU', 'value': 2}, {'name': 'DE', 'value': 1}, {'name': 'NL', 'value': 1}, {'name': 'US', 'value': 1}]
actual_response = [{'name': 'US', 'value': 1}, {'name': 'AU', 'value': 2}, {'name': 'DE', 'value': 1}, {'name': 'NL', 'value': 1}]

    def validate_response(expected_response, actual_response):
        values_actual = []
        values_expected = []
        assert len(actual_response) == len(expected_response)
        for i in range(0,len(actual_response)):
            values_actual.append(actual_response[i]['value'])
            values_expected.append(expected_response[i]['value'])
>       assert values_expected == values_actual
E       assert [2, 1, 1, 1] == [1, 2, 1, 1]
E         At index 0 diff: 2 != 1
E         Full diff:
E         - [1, 2, 1, 1]
E         ?     ---
E         + [2, 1, 1, 1]
E         ?  +++

test_api.py:165: AssertionError
 ------------------------------Captured stdout call------------------------------
EXPECTED RESPONSE - SORTED
[{'name': 'AU', 'value': 2}, {'name': 'DE', 'value': 1}, {'name': 'NL', 'value': 1}, {'name': 'US', 'value': 1}]
PASSED TOP OPTION VALUE -> : 0
ACTUAL REQUEST RESPONSE -> : [{'name': 'US', 'value': 1}, {'name': 'AU', 'value': 2}, {'name': 'DE', 'value': 1}, {'name': 'NL', 'value': 1}]
```