**iCS673 Software Engineering** 

**Team 6 - IGroup**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Haiyang Wang | QA leader | *Haiyang Wang* | 2022/11/10 |
| Dawei Yin | Implmentation & design | *Dawei Yin* | 2022/11/10 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **Iteration 2** | **Haiyang,Dawei** | **11/11/2022** | **Tests in Django: iGroup, Account** |
| **Iteration 3** | **Haiyang,Dawei** | **12/10/2022** | **Tests in Django:Survey** |

# Testing Summary

In this section, you will summarize what was tested, who is involved in testing, testing techniques used, and testing result. You may have the following tests

Haiyang

1. test\_questions (app)
2. test\_surveys (app)
3. test\_answers (app)
4. test\_answer\_sheets (app)
5. test\_single\_choice\_score (app)
6. test\_multiple\_choice\_answer (app)
7. test\_teams (app)
8. test\_weight (whole source code file / web not include)
9. test\_mixed\_questions\_weight (whole source code file / web not include)
10. test\_for\_effects\_of \_more\_choices (whole source code file / web not include)
11. test\_for\_effects\_of \_more\_questions (whole source code file / web not include)
12. test\_for\_effects\_of \_num\_of\_teams (whole source code file / web not include)

Siming

1. test\_student (app)

David

1. test\_froms (web/account)
2. test\_urls (web/account)
3. test\_views (web/account)
4. test\_urls (web/iGroup)
5. test\_views (web/iGroup)
6. test\_survey\_service (web/service)

# Manual Testing Report

test\_questions:

**Single\_choice\_quesion tests**

Set up 3 normal single choice question test objects and 2 special test objects

1. test01\_single\_choice\_question\_init

Test initialization of single choice question objects

Steps: initialize test objects and see whether the input has been stored correctly

Expected:

Question\_name/ description/ weight/ choice are correctly stored

Objects’ type is SingleChoiceQuestion object

Test passed

1. test02\_get\_weight

Test get\_weight() function

Dependences: test01\_single\_choice\_quesion\_init must pass

Steps: call get\_weight for different test objects and see whether the right results have been returned

Expected:

1 for question1, 2 for question2, 3 for question3

0 for question4, None for question5

Test passed

1. test03\_add\_choice

Test add\_choice() function

Dependences: test01\_single\_choice\_quesion\_init must pass

Steps: add new choices to test objects, check whether they have been added, check whether the number of choices have been increased.

Expected:

Len = 4 after add “D” to question1

Len = 4 after add “4” to question2

Len do not change after add None or empty string

Question1 has choices after add: [“A”, “B”, “C”, “D”]

Question2 has choices after add: [“1”, “2”, “3”, “4”]

Test passed

1. test04\_get\_question\_type

Test get\_choice\_type() function

Dependences: test01\_single\_choice\_quesion\_init must pass

Steps: check all the test objects to see whether they have the right question type

Expected:

All questions have “single” question type

Test passed

1. test05\_delete\_choice

Test delete\_choice() function

Dependences: test01\_single\_choice\_quesion\_init must pass

Steps: call delete\_choice() ro delete some choices from the test objects, check the result and the number of choices after

Expected:

Question1 deletes choice 1 and remains [“A”, “C”], having choice size as 2

Question2 deletes choice 0 and remains [“2”, “3”], having choice size as 2

Question1 then deletes choice 99999999 and remains the same

Question2 then deletes choice -1 and remains the same

Test passed

1. test06\_get\_all\_choice

Test get\_all\_choice() function

Dependences: test01\_single\_choice\_quesion\_init must pass

Steps: call get\_all\_choice to check the choice results

Expected:

Question1 has choices [“A”, “B”, “C”]

Question2 has choice [“1”, “2”, “3”]

Question3 has choice [“A”, “B”, “C”]

Question4 has choice []

Question5 has choice []

Test passed

1. test07\_get\_choice\_size

Test get\_all\_choice() function

Dependences: test01\_single\_choice\_quesion\_init must pass, test05\_delete\_choice must pass

Steps: call delete\_choice() to delete some choices from test objects, then call get\_choice\_size to check the results

Expected:

Question1 has choice size 1 after delete 2 choices

Question2 has choice size 2 after delete 1 choices

Question3 has choice size 3

Question4 has choice size 0

Question5 has choice size 0

Test passed

**Multiple\_choice\_quesion tests**

Set up 3 normal multiple choice question test objects and 2 special test objects

1. test08\_multiple\_choice\_question\_init

Test initialization of multiple choice question objects

Steps: initialize test objects and see whether the input has been stored correctly

Expected:

Question\_name/ description/ weight/ choice/ max\_num\_choice are correctly stored

Objects’ type is MultipleChoiceQuestion object

Test passed

1. test09\_get\_weight

Test get\_weight() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: call get\_weight for different test objects and see whether the right results have been returned

Expected:

1 for question1, 2 for question2, 3 for question3

0 for question4, None for question5

Test passed

1. test10\_add\_choice

Test add\_choice() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: add new choices to test objects, check whether they have been added, check whether the number of choices have been increased.

Expected:

Len = 6 after add “F” to question1

Len = 6 after add “6” to question2

Len do not change after add None or empty string

Question1 has choices after add: [“A”, “B”, “C”, “D”, “E”. “F”]

Question2 has choices after add: [“1”, “2”, “3”, “4”, “5”, “6”]

Test passed

1. test11\_delete\_choice

Test delete\_choice() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: call delete\_choice() ro delete some choices from the test objects, check the result and the number of choices after

Expected:

Quesion1 delete choice 1 and remains [“A”, “C”, “D”, “E”], having choice size 4

Question2 delete choice 0 and remains [“2”, “3”, “4”, “5”], having choice size 4

Question1 then delete choice 20 and remains the same

Question2 then delete choice -1 and remains the same

Test passed

1. test12\_get\_question\_type

Test get\_choice\_type() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: check all the test objects to see whether they have the right question type

Expected:

All questions have “multiple” question type

Test passed

1. test13\_get\_choice\_size

Test get\_all\_choice() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: call get\_choice\_size to check the results

Expected:

Question1 has choice size 5

Question2 has choice size 5

Question3 has choice size 4

Question4 has choice size 0

Question5 has choice size 0

Test passed

1. test14\_get\_max\_num\_of\_choices

Test get\_max\_num\_of\_choice() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: call get\_max\_num\_of\_choice() to check the results

Expected:

Question1 has max num choice size 2

Question2 has max num choice size 3

Question3 has max num choice size 4

Question4 has max num choice size 2

Question5 has max num choice size 2

Test passed

1. test15\_get\_all\_choice

Test get\_all\_choice() function

Dependences: test08\_multiple\_choice\_quesion\_init must pass

Steps: call get\_all\_choice() to check the choice result

Expected:

Question1 has choices [“A”, “B”, “C”, “D”, “E”]

Question2 has choice [“1”, “2”, “3”, “4”, “5”]

Question3 has choice [“Python”, “Java”, “C”, “C++”]

Question4 has choice []

Question5 has choice []

Test passed

**Test\_surveys:**

Set up 3 multiple choice questions and 3 single choice questions, set up 2 arrays of students. Set up 2 surveys that each has 3 questions. Each survey contains questions from both types. Set up one useless survey that inputs empty strings and None.

1. test01\_survey\_init

Test initialization of survey objects

Dependencies: all tests in test\_question must pass

Steps: input questions, survey\_name, survey\_id and students to survey objects and see whether the input has been correctly stored

Expected:

survey\_name/ survey\_id/ questions/ students are correctly stored

All test survey objects are survey.Survey type.

Test passed

1. Test02\_append\_question

Test append\_question()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass.

Steps: set up a single question and a multiple question, then call append\_question to add them to the survey objects, check the results and the number of questions.

Expected:

append question8 (multiple) to survey1 and it is at index 3 after append, the number of questions of survey1 is now 4

append question7 (single) to survey2 and it is at index 3 after append, the number of questions of survey2 is now 4

Test passed

1. test03\_insert\_question

Test insert\_question()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass.

Steps: set up a single question and a multiple question, then call insert\_question to insert them to the survey objects at a given index, check the results and the number of questions.

Expected:

Insert question8 (multiple) to survey1 at 2 and it is at index 2 after insert, the number of questions of survey1 is now 4. The rest of the questions in survey1 remain the same.

Insert question7 (single) to survey2 at 1 and it is at index 1 after append, the number of questions of survey2 is now 4. The rest of the questions in survey2 remain the same.

Insert question7 (single) to useless survey at 0 and it is at index 0 after append, the number of questions of useless survey is now 1. There are no other questions now.

Test passed

1. test04\_get\_all\_question\_indexes\_by\_type

Test get\_all\_question\_indexes\_by\_type()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass.

Steps: call get\_all\_question\_indexes\_by\_type() to find indexes of specific types of questions in the surveys, then check the result. The result should be a set().

Expected:

survey1 has single questions in indexes {0, 1}

survey1 has multiple questions in index {2}

survey2 has single questions in index {0}

survey2 has multiple questions in indexes {1, 2}

Survey useless has no single questions, which is {}

Survey useless has no multiple questions, which is {}

Test passed

1. test05\_get\_all\_question\_weight\_by\_type

Test get\_all\_question\_weight\_by\_type()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass, test get\_all\_question\_indexes\_by\_type() must pass

Steps: call get\_all\_question\_weight\_by\_type() to find weights of specific types of questions in the surveys, then check the results. The result should be a dict(), keys are the indexes of questions and values are the weights

Expected:

survey1 has single questions in weights {0: 1, 1: 2}

survey1 has multiple questions in weight {2: 1}

survey2 has single questions in weight {0: 3}

survey2 has multiple questions in weights {1: 2, 2: 3}

Survey useless has no single questions, then not weights, which is {}

Survey useless has no multiple questions, then not weights, which is {}

Test passed

1. test06\_get\_question\_by\_index

Test get\_question\_by\_index()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass

Steps: call get\_question\_by\_index() to find specific indexes of questions in the surveys, then check the results. The result should be a question object.

Expected:

survey1 has question2 at index 1

survey2 has question6 at index 2

survey2 has None question at index 50

Survey useless has None question.

Test passed

1. test07\_get\_question\_type\_by\_index

Test get\_question\_type\_by\_index()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass, Test get\_question\_by\_index() must pass.

Steps: call get\_question\_type\_by\_index() to find question types of specific indexes of questions in the surveys, then check the results. The result should be a string, either “single” or “multiple”

Expected:

survey1 has “single” question at index 1

survey1 has “multiple” question at index 2

survey2 has “single” question at index 0

survey2 has “multiple” question at index 2

Survey useless has no questions, then not types, which is None

survey2 has None question at index 50

Test passed

1. test08\_get\_all\_questions\_by\_type

Test get\_all\_question\_by\_type()

Dependencies: all tests in test\_question must pass, test\_init of survey must pass, Test get\_question\_by\_index() must pass.

Steps: call get\_question\_type\_by\_index() to find question types of specific indexes of questions in the surveys, then check the results. The result should be a string, either “single” or “multiple”

Expected:

survey1 has “single” question at index 1

survey1 has “multiple” question at index 2

survey2 has “single” question at index 0

survey2 has “multiple” question at index 2

Survey useless has no questions, then not types, which is None

survey2 has None question at index 50

Test passed

**test\_answers:**

**Single answers:**

Set up 3 single choice questions and make them an array, set up a student array, set up a survey then input students and questions to it. Finally set up 3 normal single answers and 3 special condition answers.

1. test01\_init

Test initialization of single choice answer

Dependences: all tests in test\_questions must pass, all tests in test\_surveys must pass

Steps: input question, survey, choice\_result to answers and see whether the input has been correctly stored

Expected:

question/ survey/ choice\_result have been store correctly

All answers are answer.SingleChoiceAnswer type

Test passed

1. test02\_get\_choice\_result

Test get\_choice\_result()

Dependences: all tests in test\_questions must pass, all tests in test\_surveys must pass, single choice answer init must pass

Steps: call get\_choice\_result(), then see whether the result is correct. The result should be the index of choice.

Expected:

answer1 have choice\_result 0

answer2 have choice\_result 1

answer3 have choice\_result 2

answer4 have choice\_result 99999

answer5 have choice\_result -1

answer6 have choice\_result None

Test passed

**multiple answers:**

Set up 3 multiple choice questions and make them an array, set up a student array, set up a survey then input students and questions to it. Finally set up 3 normal multiple answers and 3 special condition answers.

1. test03\_init

Test initialization of multiple choice answers

Dependences: all tests in test\_questions must pass, all tests in test\_surveys must pass

Steps: input question, survey, choices\_result to answers and see whether the input has been correctly stored

Expected:

question/ survey/ choices\_result have been store correctly

All answers are answer.MultipleChoiceAnswer type

Test passed

1. test04\_get\_choice\_result

Test get\_choice\_result()

Dependences: all tests in test\_questions must pass, all tests in test\_surveys must pass, multiple choice answer init must pass

Steps: call get\_choice\_result(), then see whether the results are correct. The results should be dict(), the keys are the order of choice and the values are the indexes of choice.

Expected:

answer1 choices\_result: {0: 1, 1: 2}

answer2 choices\_result:{0: 3, 1: 1, 2: 4}

answer3 choices\_result: {0: 1, 1: 0, 2: 3, 3: 4}

answer4 choices\_result: {0: 1111, 1: 5555}

answer5 choices\_result: {0: -1, 1: -10}

answer6 choices\_result: None

Test passed

**test\_answer\_sheet:**

Set up 6 questions, containing both types of questions. Set up 2 arrays store questions, each have 3. Set up 2 student arrays. Set up 2 surveys, input students and questions. Set up 6 answers according to 6 questions. Using the answers set up 2 raw\_answers. Finally set up 2 answer sheets and 1 special condition answer sheet.

1. test01\_init

Test initialization of answer sheets

Dependences: all tests in test\_questions must pass, all tests in test\_surveys must pass, all tests in test\_answers must pass

Steps: input surveys, to answers sheets and see whether the input has been correctly stored

Expected:

All surveys have been stored correctly

All answers are dict() at start

All answer\_sheets are answer.AnswerSheet type

Test passed

1. test02\_set\_answers

Test set\_answers()

Dependences: answer sheet init must pass

Steps: call set\_answers(), input raw\_answers to answer\_sheets, then see whether the raw\_ answers have been converted and stored.

Expected:

answer\_sheet1 input raw\_answer1, then the choices results have been converted to answers and then stored by answer\_sheet1

answer\_sheet2 input raw\_answer2, then the choices results have been converted to answers and then stored by answer\_sheet1

answer\_sheet3 input raw\_answer2, answer\_sheet3 has None survey, nothing is stored.

answer\_sheet2 input None, nothing is stored.

Test passed.

1. test03\_get\_answer\_by\_index

Test get\_answer\_by\_index()

Dependences: answer sheet init must pass

Steps: call get\_answer\_by\_index(), input indexes, then see whether the results are right answers

Expected:

Index 0 answer of answer\_sheet1 has the same choice\_result, survey and question as single1 (single choice answer object).

Index 2 answer of answer\_sheet1 has the same choices\_result, survey and question as multiple1 (multiple choice answer object).

Index 0 answer of answer\_sheet1 has the same choices\_result, survey and question as multiple2 (multiple choice answer object).

Index 2 answer of answer\_sheet1 has the same choice\_result, survey and question as single3 (single choice answer answer object).

Index 1111111 answer of answer\_sheet1 does not exist, return None

Index None answer of answer\_sheet2 does not exist, return None

Index 2 answer of answer\_sheet3 does not exist because answer\_sheet3 has None survey, return None

Test passed

1. test04\_get\_all\_answer\_indexes\_by\_question\_type

Test get\_all\_answer\_indexes\_by\_question\_type()

Dependences: answer sheet init must pass, survey init must pass

Steps: call get\_all\_answer\_indexes\_by\_question\_type(), input question types, then see whether the results are right indexes. The results should be set{}, containing indexes of answers.

Expected:

“single” answers of answer\_sheet1 are in {0, 1}

“single” answer of answer\_sheet2 is in {2}

“multiple” answer of answer\_sheet1 is in {2}

“multiple” answers of answer\_sheet2 are in {0, 1}

“simple question” answers does not exist in answer\_sheet1, then return set()

None answers does not exist in answer\_sheet2, then return set()

“single” answers of answer\_sheet3 does not exist, because it has None survey, return set()

Test passed

1. test05\_get\_all\_answers\_by\_question\_type

Test get\_all\_answers\_by\_question\_type()

Dependences: answer sheet init must pass, survey init must pass, test04\_get\_all\_answer\_indexes\_by\_question\_type must pass

Steps: call get\_all\_answers\_by\_question\_type(), input question types, then see whether the results are right answers. The results should be dict{}, keys are the indexes and values are answers.

Expected:

Index 0 answer of “single” answers of answer\_sheet1 has the same choice\_result, survey and question as single1 (single choice answer object).

Index 1 answer of “single” answers of answer\_sheet1 has the same choice\_result, survey and question as single2 (single choice answer object).

Index 2 answer of “multiple” answers of answer\_sheet1 has the same choice\_result, survey and question as multiple1 (multiple choice answer object).

Index 0 answer of “multiple” answers of answer\_sheet2 has the same choice\_result, survey and question as multiple2 (multiple choice answer object).

Index 1 answer of “multiple” answers of answer\_sheet2 has the same choice\_result, survey and question as multiple3 (multiple choice answer object).

Index 2 answer of “single” answers of answer\_sheet2 has the same choice\_result, survey and question as single3 (single choice answer object).

There is no “simple answer” answer in answer\_sheet1, return dict()

There is no None answer in answer\_sheet2, return dict()

There is no answer in answer\_sheet3, return dict()

Test passed

**test\_single\_choice\_score:**

Set up 3 num\_choices, 3 team\_sizes, 1 zero value team\_size, 1 None value team\_size

1. test01\_cal\_single\_score

Test cal\_single\_score

Steps: use a double loop to iterate through all the normal team\_sizes and num\_choices, then call cal\_single\_score() and input team\_size and num\_choices to see the scores of all pairs whether they are correct or not.

Expected: All pairs of team\_size and num\_choices give right result

Test passed.

1. test02\_special

Test special conditions

Steps: input 0 or None values to cal\_single\_scores to see whether it can handle them

Expected:

Input 0 team\_size and return None

Input None num\_score and return None

Input None team\_size and return None

Test passed

**Test\_teams (unfinished):**

Set up test objects and form them into a team. It has 3 students, 2 single choice questions, 2 multiple choice questions, 1 survey. Each student has one answer\_sheet that storied their answers.

1. test01\_init

Test initialization of teams

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass

Steps: check whether or not all the input has been stored correctly

Expected:

team\_name/ survey/ students correctly stored

Test passed

1. test02\_get\_single\_choice\_answer

Test get\_single\_choice\_answer()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass

Steps: call function, get single\_choice\_answer from students, check whether the result are correct

Expected:

All the single choice answers have been found, all the answer objects should be the same as set up ones.

Test passed

1. test03\_get\_multiple\_choice\_answer

Test get\_multiple\_choice\_answer()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass

Steps: call function, get multiple\_choice\_answer from students, check whether the results are correct

Expected:

All the multiple choice answers have been found, all the answer objects should be the same as set up ones.

Test passed

1. test04\_get\_single\_choice\_scores

Test get\_multiple\_choice\_scores()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass, test of get\_single\_choice\_answers must pass

Steps: call function, get single\_choice\_scores of the team, check whether the results are correct

Expected:

Question single1 has score 0.66667

Question single2 has score 2

Test passed

1. test05\_get\_multiple\_choice\_scores

Test get\_multiple\_choice\_scores()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass, test of get\_multiple\_choice\_answers must pass

Steps: call function, get multiple\_choice\_scores of the team, check whether the results are correct

Expected:

Question multiple1 has score 0.4

Question multiple2 has score 0.54

Test passed

1. test06\_get\_total\_score\_by\_type

Test get\_total\_score\_by\_type()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass, test of get\_multiple\_choice\_scores()

get\_single\_choice\_scores() must pass

Steps: call function, input either “single” or “multiple”, get total scores of that type of answers. Check whether the results are correct

Expected:

“single” has total score 2.66667

“multiple” has total score 0.94

Test passed

1. test07\_get\_total\_score

Test get\_total\_score()

Dependencies: tests of question, student, answer, answer\_sheet, survey, score calculations should pass, test of get\_multiple\_choice\_scores(), test of get\_total\_score\_by\_type(), test of get\_single\_choice\_scores() must pass

Steps: call function, get total scores of the team. Check whether the results are correct

Expected:

Total score is 3.60667

Test passed

**Test\_student (incomplete):**

Made up some student instances to test basic functions of getting student’s information including: ID, name, email. In case when needed.

1. test01\_ID

Test get\_ID

Dependencies: test of basic function of student class.

Steps: call function, verify if ID obtain from function expected as input

Test Passed

1. test02\_name\_PT\_1

Test get\_name

Dependencies: test of basic function of student class.

Steps: call function, then use .lower() on both obtained string and input to compare if names are the same by-pass the cap of the alphabet.

Test Passed

1. test03\_name\_PT\_2

Test get\_name

Dependencies: test of basic function of student class.

Steps: call function, then compare if obtained string same input or not by assertTrue/False

Test Passed

1. test04\_email

Test get\_email

Dependencies: test of basic function of student class.

Steps: call function, then compare if obtained email address same as input address. Case of the alphabet was not considered as name\_PT\_2 which may need another one on this.

Test Passed

**Tests of Whole Source Code**

Test\_weight

Set up 10 students with 10 answer sheets for a multiple choice question. Each student will only choose one choice from the question.

1. test01\_negative\_weight\_accuracy

Change the weight to negative values, which means similar choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, the rate of getting perfect teams should be over 80%.

Test passed.

1. test02\_positive\_weight\_accuracy

Change the weight to positive values, which means different choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, the rate of getting perfect teams should be over 80%.

Test passed.

Test\_mixed\_questions\_weight

Set up 10 students with 10 answer sheets for a multiple choice question and a single choice question. Each student will only choose one choice from the questions, they will always have a choice from both questions.

1. test01\_most\_negative\_weight\_for\_mult\_accuracy

Change the weight of the multiple choice question to negative value, which means similar choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, students with similar multiple choice answers should be in a team. The rate of getting perfect teams should be over 80%.

Test passed.

1. Test02\_most\_negative\_weight\_for\_sing\_accuracy

Change the weight of the single choice question to negative value, which means similar choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, students with similar single choice answers should be in a team. The rate of getting perfect teams should be over 80%.

Test passed.

1. test03\_most\_positive\_weight\_for\_mult\_accuracy

Change the weight of the multiple choice question to positive value, which means different choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, students with different multiple choice answers should be in a team. The rate of getting perfect teams should be over 80%.

Test passed.

1. test04\_most\_positive\_weight\_for\_sing\_accuracy

Change the weight of the single choice question to positive value, which means different choices, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight, set num of running, set up instance and run. Get results and see the accuracy.

Expected: in this situation, the algorithm might get perfect teams, students with different single choice answers should be in a team. The rate of getting perfect teams should be over 80%.

Test passed.

Test\_for\_effects\_of\_more\_choices

Set up 10 students with 10 answer sheets for 2 multiple choice questions and 2 single choice questions. Each student will choose at least one choice from the questions, they will always have a choice from all the questions

1. test01\_change\_weight\_for\_question1

Change the weight of the first question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test02\_change\_weight\_for\_question2

Change the weight of the second question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test03\_change\_weight\_for\_question3

Change the weight of the third question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test04\_change\_weight\_for\_question4

Change the weight of the fourth question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

Test\_for\_effects\_of\_more\_questions

Set up 10 students with 10 answer sheets for 3 multiple choice questions and 3 single choice questions. Each student will choose at least one choice from the questions, they will always have a choice from all the questions

1. test01\_change\_weight\_for\_question1

Change the weight of the first question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test02\_change\_weight\_for\_question2

Change the weight of the second question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test03\_change\_weight\_for\_question3

Change the weight of the third question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test04\_change\_weight\_for\_question4

Change the weight of the fourth question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test05\_change\_weight\_for\_question5

Change the weight of the fifth question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test01\_change\_weight\_for\_question1

Change the weight of the sixth question, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change weight to negative, set num of running, set up instance and run, get results and see the accuracy. Change weight to positive, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

Test\_for\_effects\_of\_more\_teams

Set up 12 students with 12 answer sheets for 3 multiple choice questions and 3 single choice questions. Each student will choose at least one choice from the questions, they will always have a choice from all the questions. Check the results of splitting into 2 teams, 3 teams and 4 teams

1. test01\_three\_teams

Change the number of teams, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change number of teams to 3, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test02\_two\_teams

Change the number of teams, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change number of teams to 2, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

1. test03\_four\_teams

Change the number of teams, and see the accuracy of our algorithm.

Dependencies: all source code tests passed.

Steps: change number of teams to 4, set num of running, set up instance and run, get results and see the accuracy.

Expected: in this complex situation,there is no perfect answer. The purpose of our algorithm is to always get better teams. The rate of getting better teams finally should be over 80%.

Test passed.

**Django Web Tests:**

iGroup:

1. View tests
   1. test\_login\_client\_instance\_home\_GET
   2. test\_not\_login\_client\_instance\_home\_GET
   3. test\_login\_client\_instance\_detail\_GET\_not\_found
   4. test\_not\_login\_client\_instance\_detail\_GET\_not\_found
   5. test\_unauthorized\_login\_client\_instance\_detail\_GET
   6. test\_login\_client\_instance\_detail\_GET\_valid
   7. test\_not\_login\_client\_instance\_detail\_GET\_valid
   8. test\_login\_client\_instance\_create\_GET
   9. test\_not\_login\_client\_instance\_create\_GET
   10. test\_login\_client\_instance\_create\_POST\_add\_new\_instance
   11. test\_login\_client\_instance\_create\_POST\_add\_new\_instance
2. Url tests
   1. test\_instance\_home\_url\_is\_resolved
   2. test\_instance\_detail\_url\_is\_resolved
   3. test\_instance\_create\_url\_is\_resolved
   4. test\_instance\_config\_url\_is\_resolved

Account:

1. View tests
   1. test\_register\_instructor\_GET
   2. test\_login\_instructor\_GET
   3. test\_logout\_instructor\_GET
   4. test\_register\_instructor\_POST\_add\_new\_user
   5. test\_register\_instructor\_POST\_no\_data
2. Url tests
   1. test\_register\_url\_is\_resolved
   2. test\_login\_url\_is\_resolved
   3. test\_logout\_url\_is\_resolved
3. Form tests
   1. test\_instructor\_creation\_form\_valid\_data
   2. test\_instructor\_creation\_form\_conflict\_password
   3. test\_instructor\_creation\_form\_no\_data
   4. test\_instructor\_creation\_form\_easy\_password
   5. test\_instructor\_creation\_form\_invalid\_email

Survey:

1. View tests
   1. test\_create\_a\_survey
   2. test\_create\_a\_survey\_intruder
   3. test\_create\_a\_survey\_not\_login
   4. test\_view\_survey\_index
   5. test\_intruder\_view\_survey\_index
   6. test\_not\_login\_view\_survey\_index
   7. test\_add\_question
2. Url tests
   1. test\_register\_url\_is\_resolved
   2. test\_login\_url\_is\_resolved
   3. test\_logout\_url\_is\_resolved

# Testing Metrics

In this section, you shall report any metrics used for the evaluation, e.g. # of test cases, test coverage, defects rate, etc.

# of test cases

# References

Source codes

# 