

Assignment

Introduction

Overview

The assignment is an individual assessment. It contributes **30%** of your final marks.

The due date for the programming exercises is on **26 October 2023, 11.59 pm (Sydney time)**.



This is an assignment, and staff are not permitted to give guidance on your code or how to solve the specific problem. That is the purpose of the assessment that you are required to perform to achieve the grade.

You may ask clarification questions about the assignment description. This is often necessary to implement functionality that is otherwise ambiguous.

If you have a question to ask on Ed please search before asking. However, remember that you should not be posting any assignment code publicly, as this would constitute academic dishonesty. Also, do not wait too long before starting. This assignment needs time and sustained effort.



Late submissions are not accepted unless an approved special consideration is granted.

Task Introduction and Summary

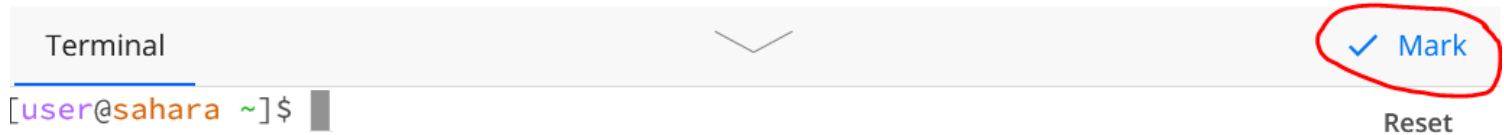
You are writing a program that allows students such as yourself to complete an online exam. The exam can contain multiple-choice questions (single or multiple answers) and short answer questions (numerical or text). The user interface of the program is text-based. In short, students have to type in commands into the terminal instead of 'point and click'. You can see the rest of the description here:

<https://edstem.org/au/courses/12106/lessons/38968/slides/293370>

Programming Exercises

All submissions must be made via Ed, including any supporting documentation that is produced during the planning of the program design such as flowcharts, pseudocodes, and UML class diagrams.

To make a submission, you will need to press the "Mark" button. You may submit as many times as you wish **before the due date**.



Your submission will be marked using an auto-grader that checks the correctness of the program with the given specification. The marking system will automatically check your code against the public test cases after you make a submission by pressing the `Mark` button. If you attempt to deceive the auto-grader or do not answer the questions, 0 marks will be awarded for the question. After each submission, you will be able to view all the previous submissions and your files in the code submission section

The following rules apply to your submission:


- Your submission must be able to compile and run within the Ed environment provided.
- Only the files given in the scaffold codes will be started by the auto-marker.

Ensure that you have submitted your codes with the correct file name as given in the questions' scaffold. The auto-marker will only start the files that are specified in the given scaffold.

The Python version that is presently being used on the Ed system:

```
$ python3 --version
Python 3.11.3
```

Restrictions

 The following must be read, as it says what you can and can't use.

Keywords

The following is not allowed:

- `for`
- `in (__contains__())`
- `global`
- `lambda`
- `nonlocal`

Built-in Functions

The following is not allowed:

- `all()`
- `any()`
- `dir()`
- `eval()`
- `enumerate()`
- `filter()`
- `globals()`
- `locals()`
- `map()`

Imports

Only the following modules are allowed:

- `import math`
- `import datetime`
- `import sys`
- `import time`
- `import os`
- `import shutil`
- `import random`
- `import unittest`
- `import io`
- any module you have written, e.g. `import shop`

Everything else is **restricted**.

Size restrictions

A submission containing a file longer than 3000 lines or larger than 500kB will not be accepted. To check the file sizes of your files, use the bash command `ls -s --block-size=KB`

Help and feedback

You are encouraged to ask questions about the assignment during the Helpdesk and on the Ed discussion board; however, remember that **you should not be posting any assignment code publicly, as this would constitute academic dishonesty**. Also, you should not disclose your code

or talk about your solutions in any of the PASS sessions.

Marking Criteria

Your marks for the assignment will be determined based on the number of test cases passed for each question as well as manual grading by your tutor. We will use the **latest** submission received before the due date for marking.

- **Automatic tests - 20/30**

Your marks for this component will be the ratio of the number of test cases passed to the sum of all the test cases administered. There will be public (and hidden) testcases that are available for feedback purposes - to allow you to test the conformance of your program with respect to the description but these do not test all the functionalities described in the assignment. It is important that you thoroughly test your own code. The private test cases will only be applied *after* the deadline and will be different from the examples given.

- **Manual grading - 10/30**

The manual grading from your tutor will consider the style, layout, and comments you provide in your `test_program.py` and your answers in `test_plan.md`. During marking, your tutor will not be debugging your code and will only run `test_program.py` according to the directions you have given in the docstring of the file. The marks for this component will be reported to this Canvas Assignment link:

<https://canvas.sydney.edu.au/courses/51815/assignments/475131>



The latest submission before the due date will be used for marking. Marks will only be reported based on this single submission. Request to grade files and derive marks from a combination of different submissions will **not** be accepted. It is your responsibility to check that your latest submission is complete, and that you pressed the "Mark" button!

Friendly note

Sometimes we find typos or other errors in specifications. Sometimes the specification could be clearer. Students and tutors often make great suggestions for improving the specification. Therefore, this assignment specification may be clarified up to **Week 10, 12th of October**. No major changes will be made. Revised versions will be clearly marked in the **Log of Changes** slide and the most recent version will be in here.

Academic Declaration

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment is my

own work and has not been copied from other sources or been previously submitted for award or assessment. I also did not use any generative AI tools (including ChatGPT) to assist in writing this assignment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realize that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

Exam Program


At the end of this assignment, you should have a program allowing students like yourself to complete an online exam. The exam can contain multiple-choice questions (single or multiple answers) and short answer questions (numerical or text). The user interface of the program is text-based. In short, students have to type in commands into the terminal instead of 'point and click'.

Command Line Arguments

The program expects the administrator to provide the details of the exam to the program, in the order of appearance, as command line arguments:

- An absolute path to the **directory** containing two files - an exam contents and a student list file. The files in this directory must be correctly labeled. See below for **File Formats**.
- A number indicating the duration of the exam in minutes (as an integer).
- [optional] A flag `-r` to indicate that the answers are to be shuffled. If this is not provided, the answers are displayed as-is.

If more command line arguments are supplied than needed, the program will ignore the remaining arguments. If users do not provide sufficient arguments during the running of the program, the program will display the message `Check command line arguments` to the terminal and it terminates.

 The content of Week 6 is required for this part of the assignment.

File Formats

The program expects the exam contents and student list to be provided by the administrator in files labeled as follows:

- `questions.txt` to denote this as the file containing the exam contents.
- `students.csv` to denote this as the file containing the list of students registered for the exam.

Additional files found in the directory are ignored. If any of these two files are not found, the program displays the message `Missing files` to the terminal and it terminates.

`questions.txt`

The program must parse the file to extract these fields for each question:

- `Type` : data representing the question type. This is case-insensitive and can take on these three values: `short`, `single` or `multiple`.
- `Question` : data representing the question itself.

- **Possible Answers** : data representing the answer options that candidates can choose from when it is a multiple-choice question. It is restricted to the choice of **A**, **B**, **C**, and **D**. The field should be set to **None** for short answer questions and must not be **None** for multiple-choice questions. Multiple answers must be separated by a comma.
- **Expected answer** : data representing the correct answer. This will be used by the auto-grader for marking.
- **Marks** : data representing the total marks awarded for a question.

To aid the program in parsing, administrators are advised to prepare each question in the file in this format:

```
Question - <type>
<Question details>
Possible Answers:
A. <Response>
B. <Response>
C. <Response>
D. <Response>
Expected Answer: <A|B|C|D|text|numerical>
Marks: <integer>
```

The contents presentation must match the specified question type. These conditions must be fulfilled:

- If a question is a multiple choice question, either single or multiple answers, the field **Possible Answers** must exist. The expected answer must be **A**, **B**, **C**, or **D**.
- If the question is of type multiple choice questions with a single answer, there must only be one expected answer. For those with multiple answers, the answers must be separated by a comma (,)
- Short-answer questions must not have **Possible Answers** following the **Question** field.

These are examples of content presentation for each question type:

- **Single** type question - single answer

```
Question - Single
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
<Start>I love INFO1110<End>
*****
Possible Answers:
A. print(I love INFO1110)
B. print("I" "love" "INFO1110")
C. print("I love" "INFO1110")
D. print("I love INFO1110")
Expected Answer: D
Marks: 1
```

- Multiple choice questions - multiple answers

```
Question - Multiple
Select all the animals that are mammals:
Possible Answers:
A. whale
B. dog
C. fish
D. frog
Expected Answer: A, B
Marks: 2
```

- Short answer

```
Question - Short
This question is based on the program below:
*****
# Start
name = input("What's your name?: ")
# End
*****
After running the program, you have keyed in "Bob" followed by the Enter key.
What is the value stored in variable name?
Expected Answer: "Bob"
Marks: 1
```

The administrator may opt to prepare the file to contain a mixture of question types or all of the same type. The program must be able to differentiate this. The question numbering will be dependent on the presentation sequence of the question itself in the file. For example, the questions parsed from the file `sample_types.txt` will be viewed in the terminal in the following sequence by a candidate during the exam:

```
Question 1 - Single Answer[1]
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
<Start>I love INFO1110<End>
*****
A. print(I love INFO1110)
B. print("I" "love" "INFO1110")
C. print("I love" "INFO1110")
D. print("I love INFO1110")
Expected Answer: D
```

```
Question 2 - Multiple Answers[2]
Select all the animals that are mammals:
A. whale
B. dog
C. fish
D. frog
Expected Answer: A, B
```


Question 3 - Short Answer[1]

This question is based on the program below:

Start

```
name = input("What's your name?: ")
```

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Expected Answer: "Bob"

-End-

These are some examples of **invalid** content presentation for each question type:

- Invalid number of answer options for **Single** question

Question - Single

This question is based on the program below:

Start

```
name = input("What's your name?: ")
```

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Possible Answers:

A. print(I love INF01110)

B. print("I" "love" "INF01110")

C. print("I love" "INF01110")

Expected Answer: D

Marks: 1

- Invalid Marks for **Multiple** question

Question - Multiple

Some question.

Possible Answers:

A. First Option

B. Second Option

C. Third Option

D. Fourth Option

Expected Answer: A, B, D

Marks: -2

- **Short** question with answer options

Question - Short

How can you generate random numbers in python?

Possible Answers:

- A. *Bob*
- B. "Bob"
- C. "BobEnter"
- D. Bob

Expected Answer: random() function

Marks: 1

students.csv


The list of students who will be sitting for the exam must be saved in a `csv` file with the following details:

- `SID`: represent the student's identification number
- `Name`: represents the student's full name
- `Extra Time`: represents additional time, in minutes, granted to the candidate. If additional time is granted, it must be a whole number. Else, it can be left as blank or `0`.

The first row is the header. Each row after that represents a candidate that will be doing the exam.

For example, this is the data in `sample_shortlist.csv` which contains the details for `5` candidates with `1` candidate is given an extra `10` minutes of time while the rest are not given additional time for the exam.

```
SID,Name,Extra Time
500010347,Damian Carroll,
500594412,Morgan Valencia,10
500622656,Tom Goldfinch,
500685031,Lisbeth Todd,
500498361,Todd Werner,
```

 The contents from Week 7 is required for this part of the assignment.

Exam Process

The program must support the entire examination process which covers these stages:

- Set up the exam.
- Assign the exam to candidates.
- Administer the exam and auto-marking.

Your job is to write Python codes to support both administrators and candidates for each stage.

If files are not in this format, the program should terminate as it is not able to successfully complete process 1. Set up the exam. In other words, the other processes must not load.



This part of the assignment requires all contents from Week 1 to Week 10.

1. Setup the exam

How do we setup the exam?

The exam contents must be loaded into the program using specially formatted files. See **Exam Program - File Formats** - `questions.txt`, for more information on this. At this stage, the program parses the question file to extract all the exam contents. An exam can only be set up if all questions in the file are complete i.e. all requested information as specified in **Exam Program - File Formats** for each question are present and successfully extracted by the program.

Once the exam is setup, administrators should can opt to preview the exam or continue on with the next stage of the process. The program will continue prompting user with this message `Do you want to preview the exam [Y|N]?` until they enter `N` or `n`. If a response other than `y / Y` or `n / N` is given, the program will display `Invalid command.` before prompting for another input. If the exam is not setup correctly, the program must warn administrators of the problem and terminate with the error message `Error setting up exam.` See **Program Execution** below for an example of an unsuccessful setup.

The module `setup.py` contains functions that will be used by the program to setup the exam, but we will get to that later.

Let's begin to code...

Question

A class called `Question` should be implemented to represent a single question extracted from the file. This class must be written in `question.py`. A `Question` object must have the following attributes:

- `qtype`: A string representing the type of question. It will always be one of these values: `single`, `multiple`, `short` and `end` which represent questions of type multiple choice questions with single answer, multiple choice questions with multiple answers, short answer questions and the end of the exam respectively.
- `description`: A string representing the question description. This is what someone would read to be able to answer the question.
- `answer_options`: A list representing the possible answers for the questions of type `single` or `multiple`. A `short` question would have this attribute as an empty list.
- `correct_answer`: A string representing the expected answer response for the question description resulting in the response being marked correct.
 - For questions of type `single`, it will be a single option value (either `A`, `B`, `C` or `D`).

- For questions of type `multiple`, it will be one of the following:
 - A single option value (`A`, `B`, `C` or `D`).
 - A comma separated `string` of option values (e.g. `A, B, C`).
- `marks`: An integer representing the amount of marks awarded to the question if a candidate gets the answer correct.

A `Question` object will have the following setter and getter methods. The setter methods are defined to verify that the provided argument is a valid value or fulfils a set of given conditions before the values are assigned to the instance variables.

- `__init__`: A constructor which accepts one argument of type `str` representing the type of question. The type of question must be valid, else it will default to `None`. The default value for attribute `description`, `correct_answer` and `marks` is `None`. Attribute `answer_options` has a default value of an empty list.
- `set_type`: The argument `qt` will only be assigned to the attribute if the argument is one of the given values: `single`, `multiple`, `short` or `end`. If the method successfully assigns the value instance variable, the method returns `True`, otherwise `False`.



For `set_description`, if the question type is `end`, immediately return `False`.

- `set_description`: The argument `desc` will only be assigned to the attribute if the argument is of type `str` and is not an empty string. If the method successfully assigns the value instance variable, the method returns `True`, otherwise `False`.



For `set_correct_answer`, if the question type is `end`, immediately return `False`.

- `set_correct_answer`: The argument `ans` will only be assigned to the attribute if the argument is of type `str`.
 - For questions of type `single`, it must be a single option value (either `A`, `B`, `C` or `D`).
 - For questions of type `multiple`, the answer must either be a single option value (`A`, `B`, `C` or `D`), or a comma separated string of option values (e.g. `A, B, C`).
 - For questions of type `short`, the answer can be anything.

For `set_correct_answer`, if the method successfully assigns the value instance variable, the method returns `True`, otherwise `False`.



For `set_answer_options`, if the question type is `short` or `end`, `opts` should be assigned to the attribute without any checks.

- `set_answer_options`: The argument `opts` will only be assigned to the attribute if the argument is of type `list` and each element in this list is a tuple with two elements. The first element in the tuple represents the answer description as a `str` and the second element is a `bool` value that flags whether the answer description is correct (`True`) or incorrect (`False`).

However, all flags should initially be `False`.

Conditions for answer options to be valid (it should be checked in this order):

- The answer description must start with `A.`, `B.`, `C.` or `D.`.
- There should be 4 answer options in the order `A`, `B`, `C`, `D`.
- `correct_answer` is of `str` type.
- `correct_answer` has only one correct answer for a `single` type question.
- `correct_answer` has at least one correct answer for a `multiple` type question.

If all above is valid, you'll then need to set the flags based on `correct_answer` so the answer options are correctly labelled as correct and incorrect (`True` and `False`).

For `set_answer_options`, if the method successfully assigns the value instance variable, the method returns `True`, otherwise `False`.

i For `set_marks`, if the question type is `end`, immediately return `False`.

- `set_marks`: The argument `mark` will only be assigned to the attribute if the argument is of type `int` and is more than or equals to `0`. The method returns `True` if the value is successfully assigned to the attribute.

i For `get_answer_option_descriptions`, if the question type is `short` or `end`, you can return an empty string immediately.

- `get_answer_option_descriptions`: The method returns a string object representing all the answer descriptions available for students to select during the exam.

A `Question` object has the following methods to help administrators setup the exam correctly:

- `preview_question`: The method returns a string object showing the full question as it would appear in the exam. If the question number is `0`, replace with `X`.
 - The method returns a string object formatted as follows:


```
Question X - <Question type> Answer[<Marks>]
<Question description>
[Optional] <Answer Options>
Expected Answer: <Correct Answer>
```

i Note that `end` questions will have a different way to preview (a much simpler one, just displays `-End-`).

- `shuffle_answers`: The method randomizes the order of the answer options for multiple choice questions. **If the question doesn't have multiple choice questions, do not produce the order and return out early.**

- You need to first implement the function `generate_order` which returns a list of 4 integers in a random order.

- The integers should be between 0 and 3 inclusive to determine order.
- Numbers in the list should be unique e.g. `[3,1,0,2]` and **NOT** `[3,2,1,3]`.

-  You are required to use `random.randint()` to generate these numbers. Keep in mind that the number returned from this function is **random**, meaning the amount of times this function must be called to create the list of integers cannot be determined before runtime.

As example, the first time you run this program, you may only need to call this function 4 times for one question, but in the next run of your program, you may need to call this function 10 times for one question (you get some duplicates). If you get a number that's already in the list, it should be ignored.

The following is an example of randomisation when creating the list.

```
[]
random.randint() -> 1
[1]
random.randint() -> 2
[1, 2]
random.randint() -> 1
[1, 2]
random.randint() -> 0
[1, 2, 0]
random.randint() -> 1
[1, 2, 0]
random.randint() -> 3
[1, 2, 0, 3]
```

- You need to call the `generate_order` function and use the returned list to update the following attributes:
 - order of the answer options along with the boolean values.
 - correct answer.

The UML class diagram for your class `Question` should look like this after implementation:

Question
qtype: str description: str answer_options: list correct_answer: str marks:int
__init__(qt:str) set_type(qt:str) : bool set_description(desc:str) : bool set_answer_options(opts:list) : bool set_correct_answer(ans:str) : bool set_marks(mark:int) : bool get_answer_option_descriptions() : str shuffle_answers() : None preview_question() : str

Here is an example of some possible `Question` objects could look like just so you can get an idea.



Please keep in mind that **not every question** will be **exactly** like this. The questions generated must be determined by the `questions.txt` file inside the directory given as a command line argument when starting the program.

Single

```
qtype:      'single'
description: '''Which of the following Python statements will display this message in the termi
*****
```



```
Terminal output:
<Start>I love INFO1110<End>
*****
answer_options: [('A. print(I love INFO1110)', False),
                 ('B. print("I" "love" "INFO1110")', False),
                 ('C. print("I love" "INFO1110")', False),
                 ('D. print("I love INFO1110")'), True)]
correct_answer: 'D'
marks:         1
```

Multiple

```
qtype:         'multiple'
description:    'Select all the animals that are mammals:'
answer_options: [('A. whale', True),
                 ('B. dog', True),
                 ('C. fish', False),
                 ('D. frog', False)]
correct_answer: 'A, B'
marks:         2
```

Short

```
qtype:         'short'
description:    '''This question is based on the program below:
*****
# Start
name = input("What's your name?: ")
# End
*****
After running the program, you have keyed in "Bob" followed by the Enter key.
What is the value stored in variable name?'''
answer_options: []
correct_answer: '"Bob"'
marks:         1
```

Referring back to the multiple choice question above, let's have a look if we shuffled it using the `shuffle_answers` function. If `generate_order` returned `[2, 3, 1, 0]` as an example, the question would now look like this:

```
qtype:         'multiple'
description:    'Select all the animals that are mammals:'
answer_options: [('A. fish', False),
                 ('B. frog', False),
                 ('C. dog', True),
                 ('D. whale', True)]
correct_answer: 'C, D'
marks:         2
```

Notice only `answer_options` has had elements rearranged and `correct_answers` now has different options, the rest remains the same.

Exam

You will need to implement class `Exam` to represent the details of an exam. This class must be written in `exam.py`. An `Exam` object must have the following attributes:

- `duration`: An integer representing the total amount of time for candidates to do the exam.
- `path_to_dir`: A string representing the path to directory containing the files to setup the exam.
- `shuffle`: A Boolean value indicating if the shuffle mode should be activated (`True`) or not (`False`).
- `exam_status`: A Boolean value indicating if the exam is setup correctly. It will be `True` if the attribute `questions` contains a list of `Question` objects.
- `questions`: A list of `Question` object representing the questions in the exam.
- `name`: A string representing the name of the exam.

An `Exam` object will have the following setter and getter methods. The setter methods are defined to verify that the provided argument is a valid value or fulfils a set of given conditions before the values are assigned to the instance variables.

- `__init__`: A constructor which accepts three arguments representing the values related to duration, directory path and shuffle flag. The `exam_status` is `False` and `question` is an empty list since there are no questions upon initialisation.
- `set_name`: **It is assumed that the name of the exam is the same as the directory name** (not the entire path, just the directory name). The name should not contain any whitespaces. If a whitespace is present, it should instead be converted to `_`. This method will use the `path` parameter to get the directory path, in which we can then grab the directory name, perform the whitespace to `_` conversion, then assign the value to the attribute. As example, if `path` is `/home/exams/info1110 test`, then `name` should be `info1110_test`.
- `get_name`: The method returns a string representing the `name` of the exam in upper case. However, the string returned must have spaces in replacement of `_` e.g. if the `name` of the exam is `info1110_test`, then this method should return `INFO1110 TEST`.
- `set_exam_status`: The setter method toggles the attribute to `True` if `questions` is not an empty list.
- `set_duration`: The method receives a single argument of type `int` and this value will be assigned to `duration` if the value is more than `0`.
- `set_questions`: The method assigns the argument to `questions` only if the list of `question` objects received successfully completes the exam. The method returns `True` to indicate that the value is assigned to `questions` successfully, else `False`. If the list is not a `list` instance, the function returns `False` with no output. An exam is complete if the conditions for each field described in **Exam Program - File Formats** is fulfilled and the last element in the list is of type `end`. The following error messages should be displayed to enable the exam administrator to troubleshoot problems in the question file input:

- **Description or correct answer missing** : This applies to all questions except for **end** and occurs when the question description and correct answer has the default value.
- **Answer options incorrect quantity** : This applies only to questions of type **single** or **multiple** and occurs when the answer options are expected but none or invalid quantity is given.
- **Answer options should not exist** : This applies only to questions of type **short** which should not have any answer options for candidates to choose from during the exam.
- **End marker missing or invalid** : This applies only to question of type **end** which should be the last element in the list and should have default values for all of its attributes.

The instance method `preview_exam` is present to help administrators preview the entire exam to ensure that it is set up correctly. The method returns a string object formatted as follows:

```
<Exam Name>
Question <Question Number> - <Question type> Answer[<Marks>]
<Question description>
[Optional] <Answer Options>
Expected Answer: <Correct Answer>

Question <Question Number> - <Question type> Answer[<Marks>]
<Question description>
[Optional] <Answer Options>
Expected Answer: <Correct Answer>

-End-
```

where `<field>`, represents the data for the given question and exam. There must be a single line between each question. See Program Execution for an example of a full text output of a exam preview. These are the format specifications for each field:

- **<Exam Name>** : All characters must be in upper case and `_` must be converted to white spaces.
- **<Question Number>** : This must be an integer and start from 1.
- **<Question type>** : First character must be capitalized e.g. **Single**.
- **<Marks>** : Must follow the question type and be enclosed in square brackets e.g. **Single Answer[1]**
- **<Answer Options>** : Must be present for **single** and **multiple** type of questions and must be displayed immediately after the question description.
- **<Correct Answer>** : Correct answers must be listed one whitespace immediately after the **Expected answer:**.

The UML class diagram for your class `Exam` should look like this after implementation:

Exam
name: str duration: int setup_dir: str shuffle: bool exam_status: bool questions: list
__init__(t: int, path: str, s: bool) set_name(path) : None get_name() : str set_exam_status() : None set_duration(t: int) : None set_questions(ls: list) : bool preview_exam() : str

Here is an example of a possible `Exam` object could look like just so you can get an idea. In the above section for **Question**, we showed examples of 3 questions. We will denote these questions as `question_1`, `question_2` and `question_3` respectively for the example below.



Please keep in mind that **not every exam** will be **exactly** like this. The exam generated must be determined by several factors, including the command line arguments and the questions.

```
name:      'INF01110 TEST 1'
duration:  60
setup_dir: '/home/test_bad_questions'
shuffle:   True
exam_status: True
```

questions: [q1, q2, q3]

Your Task

1. Complete class `Questions` and `Exam` to implement all the methods given in the specification.
2. Complete function `extract_questions` in `setup.py` which accepts a single parameter, an open file object in read mode, and returns a `list` of `Question` objects with each `Question` object representing a question that is extracted from the file `questions.txt`. The last element in the `list` must be of type `end` which indicates the end of the exam. You have to include this yourself as the file does not come with an end question. You may create additional helper functions to aid the parsing of the file contents.
 - You can assume that the contents of the file would always have these fields: `Question -`, `Expected answer:` and `Marks:`. **Everything else may or may not appear in the file**, and you are required to use the relevant methods in the `Questions` class to initialise objects using the content of these fields.

```
Question - <Type>
<Question Details>
Possible Answers:
A. <Response>
B. <Response>
C. <Response>
D. <Response>
Expected Answer: <Correct Answer>
Marks: <Marks>
```

- If a question has an invalid type, a `Question` object with `None` type with default attributes should be added to the list of Questions.



You are not required to do any error handling here, as that should be done by the class methods in your `Question` class e.g checking the validity of answer options will be done by the `set_answer_options` method.



Following from above, make sure you use the methods from the `Question` class to set the attributes. As an example, for the marks, we do not assume it is an integer or a negative number. When we call `set_marks`, it will handle checking if its valid to set the attribute.

3. Complete function `parse_cmd_args` in `program_one.py`, to parse the command line arguments to extract the information required by the program e.g. directory path to folder containing contents required by the program, exam duration and shuffle flag and return these as a tuple. The function should return `None` if the function encounter exceptions while attempting to parse the arguments. The error message `Check command line arguments` should be displayed to terminal if the number of command line arguments passed to the program is insufficient. The error message `Duration must be an integer` must be displayed if the function expects an integer but a value receives is not an integer.

4. Complete function `setup_exam` in `program_one.py` which parses the required file provided in the `Exam` object and returns a tuple `(exam_obj, status)` where `exam_obj` is the updated exam object and `status` would be `True` if the exam is successfully setup. Otherwise, `False`. The function must call method `set_questions` and `set_exam_status` to update the `Exam` object during setup.
5. Implement function `main` in `program_one.py` to check if the files `questions.txt` and `students.csv` are present in the directory path received from command line argument. If the files are not found, the function displays the error message `Missing files` and terminates. If the files are found, the program should proceed to setup the exam by calling the function `setup_exam` with the correct argument. The program must notify users that it is attempting to setup the exam by displaying the message `Setting up exam...`. If the exam is not setup successfully, the program must notify users of the problem and display the message `Error setting up exam` before terminating. If it is successfully setup, the program notifies user that the `Exam is ready...` and gives users the option to preview the exam.



Please do not modify the `import setup` statement in `program_one.py`, as it may cause you to fail some test cases.

Program Execution

The autograder will only start `program_one.py`. These are sample input/outputs of the program.

- Unsuccessful setup due to first question in `/home/test_bad_questions/questions.txt` is missing the `Expected Answer` field.

```
$ python3 program_one.py /home/test_bad_questions 60
Setting up exam...
Description or correct answer missing
Error setting up exam
```

- Successful setup

```
$ python3 program_one.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #a
Invalid command.
Do you want to preview the exam [Y|N]? #y
INFO1110 TEST 1
Question 1 - Single Answer[1]
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
```

<Start>I love INF01110<End>

- A. print("I love INF01110")
- B. print("I" "love" "INF01110")
- C. print("I love" "INF01110")
- D. print(I love INF01110)

Expected Answer: A

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

- A. whale
- B. dog
- C. fish
- D. frog

Expected Answer: A, B

Question 3 - Short Answer[1]

This question is based on the program below:

Start

name = input("What's your name?: ")

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Expected Answer: "Bob"

-End-

Do you want to preview the exam [Y|N]? #n

2. Assign exam to candidates

How do we get candidates?

The candidates taking the exam must be loaded into the program before anyone can do the exam. This list of candidates must be placed in a specially formatted `csv` file, see **Exam Program - File Formats** - `students.csv`, to be read by the program. If the program does not find any candidates in the file, the program warns administrators of this problem with the error message `No candidates found in the file` and terminates. If the program is successful in parsing the list of candidates from the `csv` file, the program proceeds to assign an exam to each candidate. If the shuffle mode is activated, the possible answers for multiple choice questions will be shuffled and the exam for each candidate will vary depending on the results of shuffling. Once this stage is complete, administrators can opt to preview the exam for a selected candidate, all candidates or proceed to the next stage.

The exam preview for each candidate should be formatted as follows. These `<field>` should be updated with the candidates details.

```
Candidate: <Candidate name>(<Candidate SID>)
Exam duration: <total duration> minutes
You have <total duration> minutes to complete the exam.
<Exam Name>
Question 1 - <Question Type> Answer[<Marks>]
<Question description>
[Optional] <Answer Options>
Response for Question 1:

Question n - <Question Type> Answer[<Marks>]
<Question description>
[Optional] <Answer Options>
Response for Question n:

-End-
```

Let's begin to code...

You will need to implement class `Candidate` to represent the details of an exam candidate. This class must be written in `candidate.py`. A `Candidate` object must have the following attributes:

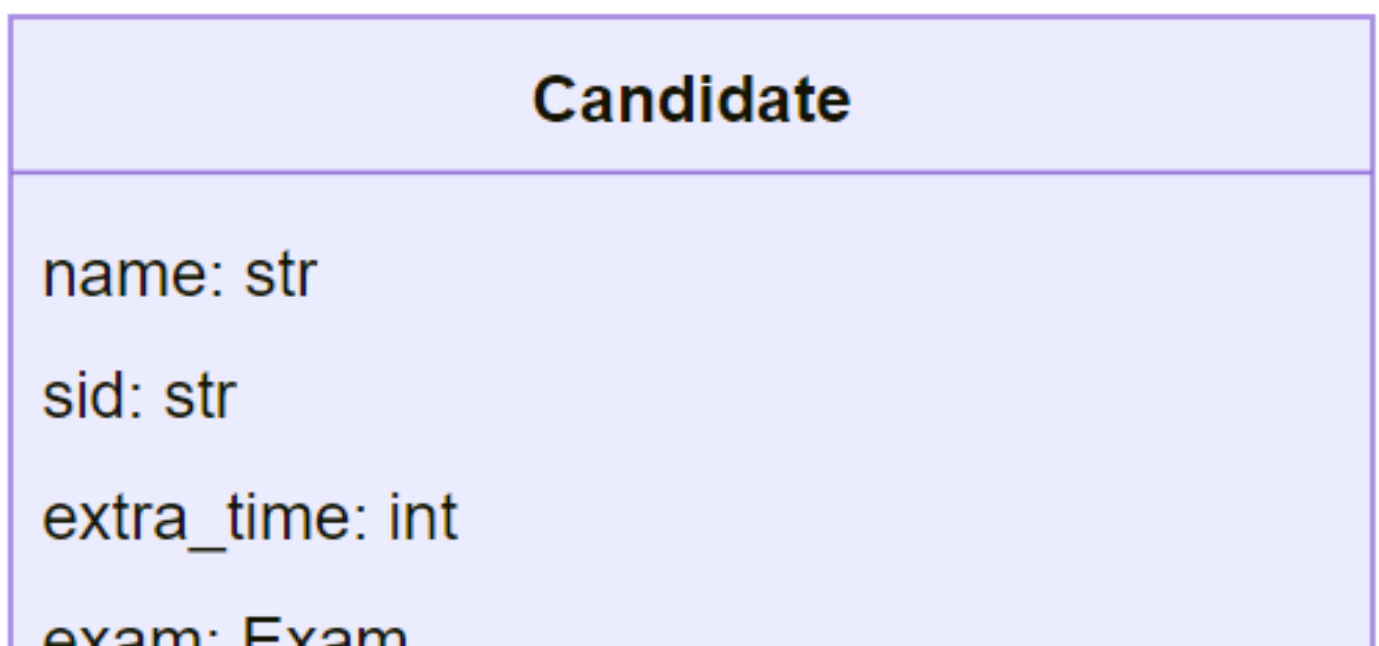
- `name`: A string representing the full name of the candidate when they register for the exam.
- `sid`: A string representing student identification number.
- `extra_time`: An integer representing additional time (in minutes) granted to the student.

- `exam`: An `Exam` object representing the details of the exam that the candidate will be doing on Exam Day.
- `confirm_details`: A Boolean value denoting whether the candidate passes (True) or fails (False) the verification check on exam day.
- `results`: A list representing the marks for each question that the candidate have achieved.

A `Candidate` object will have the following methods to allow candidates to complete the exam and administrator to save the result of their exam attempt.

- `__init__`: A constructor which accepts three arguments representing the values related to the candidate's student identification number, registered full name and extra time adjustments if any. Candidates can have no exam object assigned to them upon initialization.
- `get_duration`: The method calculates candidate's exam duration after factoring in the original exam duration and the extra time awarded.
- `edit_sid`: The method updates the candidate student identification number if the argument is a valid identification number. A valid identification number is a string representation of a positive integer with 9 numbers.
- `edit_extra_time`: Update the instance variable `extra_time` if the argument is an integer that is more than or equals to `0`.
- `set_confirm_details`: Update the attribute `confirm_details` to `True` if the arguments received matches the candidate details.
- `set_results`: Update the attribute `results` if the candidate has passed the verification check and the number of marks in the list correspond to the number of questions in the exam (excluding end question).
- `do_exam`: The method displays the candidate's details followed by each question of the exam as it would appear on the exam day itself.

The UML class diagram for your class `Candidate` should look like this after implementation:



Exam: Exam

confirm_details: bool

results: list

__init__(sid: str, name: str, time: int)

get_duration() : int

edit_sid() : None

edit_extra_time() : None

set_confirm_details(sid: str, name: str) : bool

set_results(ls:list) : None

log_attempt(data) : None

do_exam() : None

See **Section 3. Administer exam** and auto-marking for details on the `log_attempt` method.

Here is an example `Candidate` object so you can get an idea of what one would look like. This candidate has completed the exam (that's why `confirm_details` are `True` as they completed the verification check. For `results`, they did not fully ace the exam, but got some marks). We will denote the example exam object in **Section 1. Setup the exam** as `exam_1` for below.

```
name:      'Robert Pattinson'
sid:       '123456789'
extra_time: 37
exam:      exam_1
confirm_details: True
results:   [0, 1, 1]
```

Your Task

1. Complete class `Candidate` to implement all the methods given in the specification.
2. Write function `sort` in `setup.py` which has two parameters, a list and an integer. The list contains a list of items to be sorted. The parameter `order` represents the order of sorting:
 - `0` : no sort, returns a new list of the original data
 - `1` : sort in ascending order, returns a new list of the original data sorted in ascending order i.e. small to big
 - `2` : sort in descending order, returns a new list of the original data sorted in descending order i.e. big to small.

You must write your own sorting function and are not allowed to use the built-in functions or methods to accomplish this. If the first argument is invalid, the function returns an empty list. If order is not between `0` and `2`, the function returns a new list of the original data.

3. Complete function `extract_students` in `setup.py` which accepts a single parameter, an open file object in read mode, and returns a `list` of unique `Candidate` objects that is sorted in ascending order based on the SID. If the argument is invalid, the function should return an empty list. The function should parse the file contents from `students.csv` to extract each student's details, sort the students in ascending order based on their student identification number (SID) and then create `Candidate` objects to represent the student during the exam.
4. Complete function `assign_exam` in `program_two.py` which has one parameter of type `Exam` and returns a list of `Candidate` objects or `None`. Each candidate object must have an `Exam` object assigned to the instance variable `exam`. If the shuffle mode is `True` for the exam, this `Exam` object may or may not be different to other candidates in the exam depending on the result of randomization. The function must call `extract_students` to extract the list of candidates from `students.csv` that is found in setup directory. If the function is unsuccessful in accomplishing this, the function should display the message `No candidates found in the file` and return `None`. Otherwise, it should proceed to assign exam to each unique candidate extracted from the file. The function informs users that it is doing this step by displaying the message `Assigning exam to candidates...` and indicates the end of the process by displaying the message `Complete. Exam allocated to {n} candidates.` where `n` is the total number of candidates in the exam.

Hint: If randomisation occurs with the exams, then each student's exam will be different. Having the **same** exam object assigned to every student will not be a good idea, as shuffling will end up shuffling for everyone. Hence, there's a few steps you'll need to do. For each student, you'll need to:

1. For each question in the exam, make a new question with the same details.
2. Use these list of newly made question objects to make a new exam object.
3. Assign this new exam to the student.

We've provided you a scaffold to do this which requires adding a `copy_question` instance method in `Question` and a `copy_exam` instance method in `Exam`. If you do this correctly, given

an `exam` object, you can make a new exam for a student by simply calling `exam.copy_exam()` without issues of exam details conflicting across multiple students.

```
class Exam:
    # have all methods here such as init, setters, etc...
    # ...
    # ...

    def copy_exam(self):
        """
        Create a new exam object using the values of this instances' values.
        """
        # TODO: make a new exam object (call the constructor)
        new_exam = # (replace this comment with code)

        # make a new list of questions to reassign to the attribute
        new_questions = []
        i = 0
        while i < len(self.questions):
            original_question = self.questions[i]
            # call the copy method for this question
            # TODO: (you'll need to write this instance method in Question)
            new_question = original_question.copy_question()

            # insert this into new list of questions
            new_questions.append(new_question)
            i += 1

        # TODO: assign this new question list to the new exam
        # (replace this comment with code)

        # return the new exam
        return new_exam
```

The last step is that you'll need to modify function `main` in `program_two.py` to call function `assign_exam` if the first stage of the process (that is, `program_one`) is successfully completed. If the function `assign_exam` is successfully executed, the program should proceed to allow administrators to do the following inputs:

1. Preview a selected candidate's exam given the candidate's SID.
2. Preview all the exam for all candidates in the exam using `-a`.
3. Quit the preview feature using `-q`.

The program prompts the user with the following message `Enter SID to preview student's exam (-q to quit):`. If the user enters a valid SID number but the number is not found in the exam's list of candidates, the program displays the message `SID not found in list of candidates.` and re-prompts for another input. If an invalid SID number is entered, the program displays the message `SID is invalid.` and re-prompts for another input again.



Hint: Given you implemented the `main` function of `program_one` correctly, your first line in `main` for `program_two` could be `program_one.main(args)` (ensure you check the log of changes as the main now takes in the args). You can change the `main` in `program_one` to return anything you want to help you with the main in `program_two`.

Program Execution

The autograder will only start `program_two.py`. These are sample input/outputs of the program after implementing the first and second stage of the process and the administrator decides to preview all the exam for all candidates:

```
$ python3 program_two.py /home/info1110_test_1 69 -r
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-a

Candidate: Damian Carroll(500010347)
Exam duration: 69 minutes
You have 69 minutes to complete the exam.
INFO1110 TEST 1
Question 1 - Single Answer[1]
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
<Start>I love INFO1110<End>
*****
A. print("I" "love" "INFO1110")
B. print("I love" "INFO1110")
C. print(I love INFO1110)
D. print("I love INFO1110")
Response for Question 1:

Question 2 - Multiple Answers[2]
Select all the animals that are mammals:
A. fish
B. whale
C. dog
D. frog
Response for Question 2:

Question 3 - Short Answer[1]
This question is based on the program below:
*****
# Start
name = input("What's your name?: ")
# End
*****
```

After running the program, you have keyed in "Bob" followed by the Enter key.
What is the value stored in variable name?

Response for Question 3:

-End-

Candidate: Todd Werner(500498361)

Exam duration: 69 minutes

You have 69 minutes to complete the exam.

INF01110 TEST 1

Question 1 - Single Answer[1]

Which of the following Python statements will display this message in the terminal?

Terminal output:

<Start>I love INF01110<End>

A. print("I love" "INF01110")

B. print(I love INF01110)

C. print("I love INF01110")

D. print("I" "love" "INF01110")

Response for Question 1:

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

A. dog

B. frog

C. fish

D. whale

Response for Question 2:

Question 3 - Short Answer[1]

This question is based on the program below:

Start

name = input("What's your name?: ")

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3:

-End-

Candidate: Morgan Valencia(500594412)

Exam duration: 79 minutes

You have 79 minutes to complete the exam.

INF01110 TEST 1

Question 1 - Single Answer[1]

Which of the following Python statements will display this message in the terminal?

Terminal output:

<Start>I love INF01110<End>

- A. print("I" "love" "INF01110")
- B. print(I love INF01110)
- C. print("I love INF01110")
- D. print("I love" "INF01110")

Response for Question 1:

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

- A. whale
- B. dog
- C. fish
- D. frog

Response for Question 2:

Question 3 - Short Answer[1]

This question is based on the program below:

Start

name = input("What's your name?: ")

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3:

-End-

Candidate: Tom Goldfinch(500622656)

Exam duration: 69 minutes

You have 69 minutes to complete the exam.

INF01110 TEST 1

Question 1 - Single Answer[1]

Which of the following Python statements will display this message in the terminal?

Terminal output:

<Start>I love INF01110<End>

- A. print("I love" "INF01110")
- B. print(I love INF01110)
- C. print("I" "love" "INF01110")
- D. print("I love INF01110")

Response for Question 1:

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

- A. whale
- B. dog
- C. fish
- D. frog

Response for Question 2:

Question 3 - Short Answer[1]

This question is based on the program below:

```
*****
```

```
# Start
```

```
name = input("What's your name?: ")
```

```
# End
```

```
*****
```

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3:

-End-

Candidate: Lisbeth Todd(500685031)

Exam duration: 69 minutes

You have 69 minutes to complete the exam.

INF01110 TEST 1

Question 1 - Single Answer[1]

Which of the following Python statements will display this message in the terminal?

```
*****
```

Terminal output:

```
<Start>I love INF01110<End>
```

```
*****
```

A. `print("I" "love" "INF01110")`

B. `print("I love" "INF01110")`

C. `print(I love INF01110)`

D. `print("I love INF01110")`

Response for Question 1:

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

A. frog

B. dog

C. whale

D. fish

Response for Question 2:

Question 3 - Short Answer[1]

This question is based on the program below:

```
*****
```

```
# Start
```

```
name = input("What's your name?: ")
```

```
# End
```

```
*****
```

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3:

-End-

Enter SID to preview student's exam (-q to quit): #-q

Preview for a single candidate:


```

$ python3 program_two.py /home/info1110_test_1 69 -r
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #500010347

Candidate: Damian Carroll(500010347)
Exam duration: 69 minutes
You have 69 minutes to complete the exam.
INFO1110 TEST 1
Question 1 - Single Answer[1]
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
<Start>I love INFO1110<End>
*****
A. print("I" "love" "INFO1110")
B. print("I love" "INFO1110")
C. print(I love INFO1110)
D. print("I love INFO1110")
Response for Question 1:

Question 2 - Multiple Answers[2]
Select all the animals that are mammals:
A. fish
B. whale
C. dog
D. frog
Response for Question 2:

Question 3 - Short Answer[1]
This question is based on the program below:
*****
# Start
name = input("What's your name?: ")
# End
*****
After running the program, you have keyed in "Bob" followed by the Enter key.
What is the value stored in variable name?
Response for Question 3:

-End-

Enter SID to preview student's exam (-q to quit): #-q

```

Incorrect input:

```

$ python3 program_two.py /home/info1110_test_1 69 -r
Setting up exam...
Exam is ready...

```

```
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-s
SID not found in list of candidates.

Enter SID to preview student's exam (-q to quit): #-q
```

3. Administer exam and auto-marking

Finishing our Program...

In the final stage of the process, candidates follow the prompts given by the program to complete their exam. The program starts by asking candidates to enter their student identification number to start the exam. If they respond with an invalid SID for three attempts, the program directs candidates to `Contact exam administrator.` and terminates. If a valid SID is given but the candidate is not listed as a candidate for the exam, the program ask users if they want to attempt entering their identification number again. If they fail to enter the exam after three attempts, the program directs candidates to `Contact exam adminsitator.` and terminates. If the SID is valid and the candidate is officially assigned to the exam, the program proceeds to verify their identity by asking the candidate to provide their full name. If the name matches the name given in the `students.csv` file for the exam, the program proceeds to display each question of the exam and prompt users for their answer.

All candidates are given one attempt to do the exam and the exam must be completed in one sitting. Candidates are not allowed to modify their answers after submitting their response for each question. The auto-marker will proceed to grade the response. The program saves a copy of the student's responses as a text file for audit purposes.

Your Task

- Update class `Question` to add method `mark_response` which accepts one argument representing the response given by candidate and returns the marks awarded for this response. For questions of type `multiple`, partial marks should be awarded if students get any of the answers correct. The marks should be evenly distributed between each correct answer. For example, if a question is worth 2 marks and the expected answer is `B` and `D` but the student's response is `B` and `C`, a total of `1` mark should be awarded to the student. *(We are aware this is not how the real world works, but it keeps things simple for implementation, we should not be worrying about marking formulas for this course. So as example, if a student was to answer every option, they would get 2 marks, despite them not knowing what's actually correct, do not do this in your final exam students!😄)*. If the question is an `end` question, prematurely return `None` (the default value). The mark should be rounded to 2dp.
- Design testcases to verify that the method `mark_response` is implemented correctly for each type of question. Write your answers in `test_plan.md`. You must have at least `2` testcases (a positive and a negative) for each type of question. Hence, a minimum total of `6` testcases. Then, implement the test cases in `test_program.py`. Write instructions as docstring at the top of file to explain how to run the test program and interpret its results. This question requires

contents from Week 10.

- Implement the method `log_attempt` in `candidate.py` to enable the responses submitted by the candidate during the exam to be written to a file that is labelled with the candidates SID.
- Modify method `do_exam` in `candidate.py` to have a single **default** parameter `preview` of type `bool`. If `preview` is set to `True`, the method displays the contents of the exam but does not accept any inputs from the user. In short, the behaviour can be used by administrators to preview the exam from the candidate's point of view in Section 2. Assign `exam` to `candidates`. If it is set to `False`, the method will wait for user's to provide a response to the question. If the question is of type `single`, the method must check that responses is a valid answer option e.g. `A`, `B`, `C`, or `D` (it cannot be `A`, `B` as an example). If the question is of type `multiple`, the method must check responses is a valid answer option or a collection of it e.g. `A`, `B`, `C`, `D`, `A`, `B`, `A`, `B`, `D`, etc. Once a response is received, the method must call `mark_response` to determine the marks that should be awarded based on the candidate's response. If the answer is invalid, the student automatically gets 0 for that question. At the very end, it will then save the details of the question, response and marks to the candidate's file by calling `log_attempt`.
- Modify function `main` in `program_final.py`, to prompt candidates with instructions to enter their student identification number to start the exam. If the response provided by user is not a valid SID number, the program displays the message `Invalid SID.` and re-prompts for another input.

If the number is a valid SID number but the number is not found in the candidate list, the program displays the message `Candidate number not found for exam.` and prompt users `Do you want to try again [Y|N]?`. If the response is not `y` or `n`, the program displays the message `Response must be [Y|N].` and re-prompts candidates for their decision to try again (note that this does not contribute to the three attempts, there will be an example below). If the user responds with `n` or `N`, the program informs user to `Contact exam administrator to verify documents.`. If the user responds with `y` or `Y`, the program prompts to `Enter the student identification number (SID) to start exam:` and waits for a new SID to be entered by the user. If they made THREE mistakes during this stage, the program displays the message `Contact exam administrator.`

If the number is a valid SID and the number is found in the candidate list, the program proceeds to inform users `Verifying candidate details...` and proceeds to prompt `Enter your full name as given during registration of exam:`. The program checks if the combination of `SID` and name (case insensitive) given by the user matches the candidate records. If it does not match the records, the program informs that `Name does not match records.` and re-prompts user to enter their full name again. The program tracks invalid attempts by the users. If they made THREE mistakes during the verification stage, the program locks the user out by displaying `Contact exam administrator to verify documents.` and terminates.

If it matches the records, the program proceeds to with administer the exam with the message `Start exam...` and proceeds to show each question in the exam to the user by calling the method `do_exam`.



Similarly to `program_two` using `main` of `program_one` in the first line, if done correctly, the first line of `program_final` should be calling `program_two`'s `main` function with whatever return value you wish for it to have. This is only a suggestion, you are free to do what you want.

Program Execution

The autograder will only start `program_final.py`. These are sample input/outputs of the program after implementing the all stages of the process and the administrator decides to preview all the exam for all candidates:

- Incorrect SID on all tries.

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #abcd1234
Invalid SID.
Enter your student identification number (SID) to start exam: #1234567
Invalid SID.
Enter your student identification number (SID) to start exam: #12345678
Invalid SID.
Contact exam administrator.
```

- Valid SID but not in records for all tries.

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #500594411
Candidate number not found for exam.
Do you want to try again [Y|N]? #y
Enter your student identification number (SID) to start exam: #500594413
Candidate number not found for exam.
Do you want to try again [Y|N]? #y
Enter your student identification number (SID) to start exam: #500594414
Candidate number not found for exam.
Do you want to try again [Y|N]? #y
Contact exam administrator.
```

- Incorrect SID, invalid response, then valid SID but not in records. However, we fail the try again prompt multiple times.

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #500594411
Candidate number not found for exam.
Do you want to try again [Y|N]? y
Enter your student identification number (SID) to start exam: #1234567
Invalid SID.
Enter your student identification number (SID) to start exam: #500594414
Candidate number not found for exam.
Do you want to try again [Y|N]? #nope
Response must be [Y|N].
Do you want to try again [Y|N]? #nope
Response must be [Y|N].
Do you want to try again [Y|N]? #nope
Response must be [Y|N].
Do you want to try again [Y|N]? #y
Contact exam administrator.
```

- Valid SID but not in records, followed by not wanting to try again.

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #500594411
Candidate number not found for exam.
Do you want to try again [Y|N]? #n
```

- Correct SID and candidate in list but different name from records,

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #500594412
```

```
Verifying candidate details...
Enter your full name as given during registration of exam: #john smith
Name does not match records.
Enter your full name as given during registration of exam: #jane smith
Name does not match records.
Enter your full name as given during registration of exam: #tom smith
Contact exam administrator to verify documents.
```

- Correct SID and name. Successful completion of exam.

```
$ python3 program.py /home/info1110_test_1 60
Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? #n
Loading candidates...
Assigning exam to candidates...
Complete. Exam allocated to 5 candidates.
Enter SID to preview student's exam (-q to quit): #-q
Enter your student identification number (SID) to start exam: #500594412
Verifying candidate details...
Enter your full name as given during registration of exam: #Morgan valencia
Start exam....

Candidate: Morgan Valencia(500594412)
Exam duration: 70 minutes
You have 70 minutes to complete the exam.
INFO1110 TEST 1
Question 1 - Single Answer[1]
Which of the following Python statements will display this message in the terminal?
*****
Terminal output:
<Start>I love INFO1110<End>
*****
A. print(I love INFO1110)
B. print("I" "love" "INFO1110")
C. print("I love" "INFO1110")
D. print("I love INFO1110")
Response for Question 1: #A

Question 2 - Multiple Answers[2]
Select all the animals that are mammals:
A. whale
B. dog
C. fish
D. frog
Response for Question 2: #A, B

Question 3 - Short Answer[1]
This question is based on the program below:
*****
# Start
name = input("What's your name?: ")
# End
```

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3: #Bob

-End-

This is the text stored in the student's file 500594412.txt which is found in /home/info1110_test_1/submissions.

Question 1 - Single Answer[1]

Which of the following Python statements will display this message in the terminal?

Terminal output:

<Start>I love INFO1110<End>

A. print(I love INFO1110)

B. print("I" "love" "INFO1110")

C. print("I love" "INFO1110")

D. print("I love INFO1110")

Response for Question 1: A

Expected Answer: A

You have scored 1.00 mark.

Question 2 - Multiple Answers[2]

Select all the animals that are mammals:

A. whale

B. dog

C. fish

D. frog

Response for Question 2: A, B

Expected Answer: A, B

You have scored 2.00 marks.

Question 3 - Short Answer[1]

This question is based on the program below:

Start

name = input("What's your name?: ")

End

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Response for Question 3: BOb

Expected Answer: "Bob"

You have scored 0.00 marks.

Submission

Specification

Please refer to the slides in this lesson for the specifications.

Style Guide:

The style guide for this assessment can be found on the official Python website:

<https://peps.python.org/pep-0008/>

In addition to the official style guide, you can also refer to these resources:

- [Code style guide - Part 1](#)
- [Code style guide - Part 2](#)

Understanding Test Cases Output

Example 1:

On the header you can find the information on what is being tested:

- **Q1:** The question the test case is part of
- **(Question):** file being tested
- **set_answer_options:** function being tested
- **incorrect order:** description of the test case

✓ Q1: (Question) set_answer_options; single; incorrect order



Your program produced the incorrect output.

DIFF SPLIT DIFF YOUR OUTPUT EXPECTED

```
- Traceback (most recent call last):
-   File "/home/classes/question_test.py", line 590, in <module>
-       basic(name_arg)
-   File "/home/classes/question_test.py", line 284, in basic
-       assert actual_ans == [], output_with_comments([],actual_ans,
'Attribute incorrectly set')
-           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
-   AssertionError:
-       Expected: []
-       ##### <END> #####
-       Got: [('A. int', True), ('C. str', False), ('B. float',
False), ('D. None', False)]
-       ##### <END> #####
-       Additional comments: Attribute incorrectly set
- 
```

[Show explanation](#)

You will find the relevant output under **AssertionError** where:

- **Expected** is what the the test case was expecting your function to do
- **Got** is what your function did
- ##### <END> ##### is a marker showing the end of the expected/actual block (not to be printed or returned by your function)
- **Additional comments:** will give you additional information on the error. In this case, the function did not set the attribute correctly.

Example 2:

The header is similar to the previous example

Q1: (program_one.py) main; sample directory; no preview



Your program produced the incorrect output.

DIFF SPLIT DIFF YOUR OUTPUT EXPECTED

```
- Random...
+ Setting up exam...
Exam is ready...
Do you want to preview the exam [Y|N]? n
```

[Show explanation](#)

- **Highlighted in Green (+):** Your program did not produce/output that line
- **Highlighted in Red (-):** Your program produced an incorrect output and should be removed.

Submission Checklist:



Don't forget to press the **Mark** button to submit your work!!!



The latest submission before the due date will be used for marking. Marks will only be reported based on this single submission. Request to grade files and derive marks from a combination of different submissions will **not** be accepted. It is your responsibility to check that your latest submission is complete.



If you applied for special consideration, you must continue making a submission each day. We will use the last submission before the approved deadline for marking.

You must have the following files in your final submission:

A. Python and Markdown Files

- `setup.py` - contains the helper functions to extract questions and candidates list from file. See "Your Task" for **1. Setup the exam** and **2. Assign exam to candidates**
- `question.py` - contains your implementation for class `Question`.
- `exam.py` - contains your implementation for class `Exam`.
- `program_one.py` - contains the interface of the exam program for **1. Setup the exam**
- `candidate.py` - contains your implementation for class `Candidate`.
- `program_two.py` - contains the interface of the exam program for **2. Assign exam to candidates**
- `test_program.py` and `test_plan.md` - contains the test case design and implemented

testcases for method `mark_response`.

- `program.py` - contains the interface of the exam program for **3. Administer exam and auto-marking**

B. Optional Files

- Additional Python and text files that was used to test your program.
- readme file to explain structure of submission

Unable to load poll



Press **Mark** after you upload your files or edit your workspace to submit your work.

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

Log of changes



No major revision will be made after **Week 10 - 12th October 2023**. All public testcases will be released before this date.

All notable changes to this assignment will be documented here. These consist of:

- Buggy test cases.
- Typos in the question description.
- The question description lacks sufficient details.



If the example output in the question description and the output checked by test cases don't match, **only the test cases will be changed to make it match.**



Dates will be labelled with a warning callout.



Hopefully this does not happen, but any extreme issues addressed in this log will be highlighted in an error callout.

Log

Pinned (Changes to Scaffold)



30 September

4:03PM: You'll need to `import sys` and pass `sys.argv` into the main function call, similarly to `program_one`.

You can either **reset to scaffold** to pull these changes or copy and paste the the code below into `program_two.py` (**make sure to either mark your code or save it somewhere before to restore your code**).

```
'''
Interface of the exam
'''

import setup
import program_one
import sys

def assign_exam(exam):
```

```

pass

def main(args):
    '''
    Implement all stages of exam process.
    You can import functions from program_one to avoid duplication of code.
    '''
    pass

if __name__ == "__main__":
    '''
    DO NOT REMOVE
    '''
    main(sys.argv)

```



15 September

2:45PM: Scaffold has been updated to remove an extra space after `# Start` in the sample question for `short` type included in the directory `Samples` and `info1110_test_1`. You can either **reset to scaffold** to pull these changes or copy it from the snippet below (**make sure to either mark your code or save it somewhere before to restore your code**).

Question 3 - Short Answer[1]

This question is based on the program below:

`# Start`

`name = input("What's your name?: ")`

`# End`

After running the program, you have keyed in "Bob" followed by the Enter key.

What is the value stored in variable name?

Expected Answer: "Bob"

Additions to Clarifications / Fixes on Test Cases



1st October

6:11PM: Section `3. Administer exam and auto-marking`, result marks now are formatted to be in 2dp. Outputs have been updated.

4:08PM: Added a hint for implementing `main` in `program_final` in Section `3. Administer exam and auto-marking`.



Similarly to `program_two` using `main` of `program_one` in the first line, if done correctly, the first line of

`program_final` should be calling `program_two`'s `main` function with whatever return value you wish for it to have. This is only a suggestion, you are free to do what you want.

4:03PM: Added clarification that name entered for `3. Administer exam and auto-marking`. is case-insensitive. An example was given in the output examples, but was not clearly stated in the function description.

3:47PM: Edited `set_confirm_details` test cases, as they only tested the return value. Now they also test the `confirm_details` attribute.

3:21PM: Added more output examples for `3. Administer exam and auto-marking`.

2:33PM: Added `import sys` into `program_final.py`, it was figured you'll identify this yourself but we added it here just to be nice.

3:54AM: Added further clarification for `do_exam` in Section 3.

If the question is of type `single`, the method must check that responses is a valid answer option e.g. `A`, `B`, `C`, or `D` (it cannot be `A`, `B` as an example). If the question is of type `multiple`, the method must check responses is a valid answer option or a collection of it e.g. `A`, `B`, `C`, `D`, `A`, `B`, `A`, `B`, `D`, etc. If the answer is invalid, the student automatically gets 0 for that question.



30 September

8:33PM: Line has been added into `shuffle_answers` to clarify that the randomisation should not be called if it doesn't have to:

"If the question doesn't have multiple choice questions, do not produce the order and return out early."

6:59PM: Added more sample outputs in `2. Assign exam to candidates`.

5:38PM: Added test cases on `generate_order()`.

5:30PM: Added the following hint in `2. Assign exam to candidates`.



Hint: Given you implemented the `main` function of `program_one` correctly, your first line in `main` for `program_two` could be `program_one.main(args)` (ensure you check the log of changes as the main now takes in the args). You can change the `main` in `program_one` to return anything you want to help you with the `main` in `program_two`.



28 September

5:30PM: Added a scaffold in `2. Assign exam to candidates` to help with creating new exam objects for each student when dealing with shuffling. `question` was updated to `original_question` inside the loop.



25 September

6:45PM: One of the test cases had a bug which may have caused some student to erroneously fail or pass and has been fixed now.

- **Q1: (Exam) set_questions; invalid - quantity of options - Fixed**

Special Consideration and Simple Extensions



Late submissions are not accepted. No submission = Zero grade for entire assignment



Approved special consideration will be granted a Extension of Time.

Special Consideration (University policy)

If your performance on assessments is affected by **illness or misadventure** you can apply for special consideration. See this page for information on Special Consideration:

http://sydney.edu.au/current_students/special_consideration/



When you submit a special consideration application, you should continue making submissions after the deadline (marked as late). Only after the outcome of your application is known the consideration can be applied. It may take many days for the outcome to be known and if the consideration is an extension we will accept the last submission before the new deadline.

You need to follow proper bureaucratic procedures for your application to be processed swiftly. If you don't, there will be delays in processing your application and in the worst-case scenario, your application is rejected.

Special consideration applications for illness, injury or misadventure must be made within 3 working days of the assessment deadline. See this link for more information on **when** you need to make the application: <https://www.sydney.edu.au/students/special-consideration/apply.html>



The timeframe to put in the application is very short. You need to make sure your application is sent in *before* the deadline.

Special Consideration Checklist

- Did you select the correct reason for applying special consideration?

The type of exceptional circumstances affect the type of supporting documents required. See the table in this page: <https://www.sydney.edu.au/students/special-consideration/eligibility-documents.html>

- Have you selected the correct unit? It should be **INFO1110** or **COMP9001**.
- Have you selected the correct Assessment item? It should be **Assignment**.
- Have you selected the correct due date for the Assessment Item? It should be **26 October 2023**.

- Have you attached the relevant supporting document?

These are the common documents for illness and misadventure depending on your exceptional circumstances:

- [Statutory declaration](#)
- Professional practitioner certificate - [professional-practitioner-certificate.pdf](#)
- Student declaration - [student-declaration\(1\).docx](#)

Simple Extensions

The university had recently changed information about **simple extensions**. Please see here for more information: <https://www.sydney.edu.au/students/simple-extensions.html>

Applications for simple extensions must be made ***before*** the due date and you must attach a student declaration with your application!

We do not process simple extensions at the unit level. Please follow the instructions in the given [link](#) to apply for it.