Chapter 1

User Manual

1.1 Basic Concepts

1.1.1 Quiz Type

This software supports two types of objective question quizzes:

1. **Simple quiz:** In this quiz, the students are supposed to answer all questions. All students answer the same question paper.
2. **Jumbled quiz:** In this form of quiz, each student gets a different question paper to answer, which contains questions sampled out of a larger item bank and jumbled.

1.1.2 Question Types

There are two types of questions that can be included in the question papers: multiple choice questions (MCQ) and match the following (MTF). An MCQ is a question with two or more possible options out of which one or more may be correct choices. For example:

> Name two dynamically typed programming languages:
>
> 1. Java
> 2. **Python**
> 3. Haskell
> 4. **Ruby**

In the above, options 2 and 4 are correct answers.
The second question type is match the following.

> Match the following programming languages on the left column with properties on the right:

1. Python	A. Static typing
2. Java	B. Dynamic typing
3. C++	C. Untyped
4. OCaml	D. Implicit typing
	E. Explicit typing
	F. Functional
	G. Object oriented

In the above example, the matches are as follows

1. Python matches with B., D., F. and G.
2. Java match with A., E. and G.
3. C++ match with A., E. and G.
4. OCaml match with A., D., F., and G.

A few points to note here:

- The mapping can be many to many, even allowing some options on the either side to have no matches at all (e.g. C.).
- There is no restriction on the number of options on either columns. For example, in the above example, we have 4 options on the LHS and 7 on the RHS.

1.2 Simple Quiz

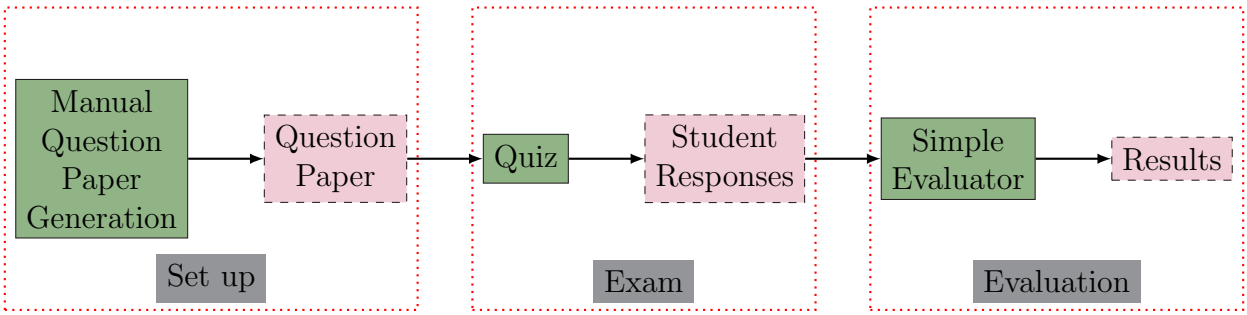


Figure 1.1: Simple Quiz Workflow

The question paper creation for a simple quiz is manual. All students solve an identical question paper. The evaluation step directly runs on the student responses using the `SimpleEvaluator` module.

1.2.1 Setup

Suppose you intend to conduct a simple quiz. As mentioned above, in a simple quiz, all students solve the same question paper. Such a quiz is ideal when there is small class and enough assurance that cheating is not a possibility. Of course, all questions are assumed to be either multiple choice questions (MCQ) or match the following questions (MTF).

The assessment project can be setup conveniently using the setup utility. The setup utility automatically generates the skeletal infrastructure for conducting such a quiz.

1.3 Jumbled Quiz

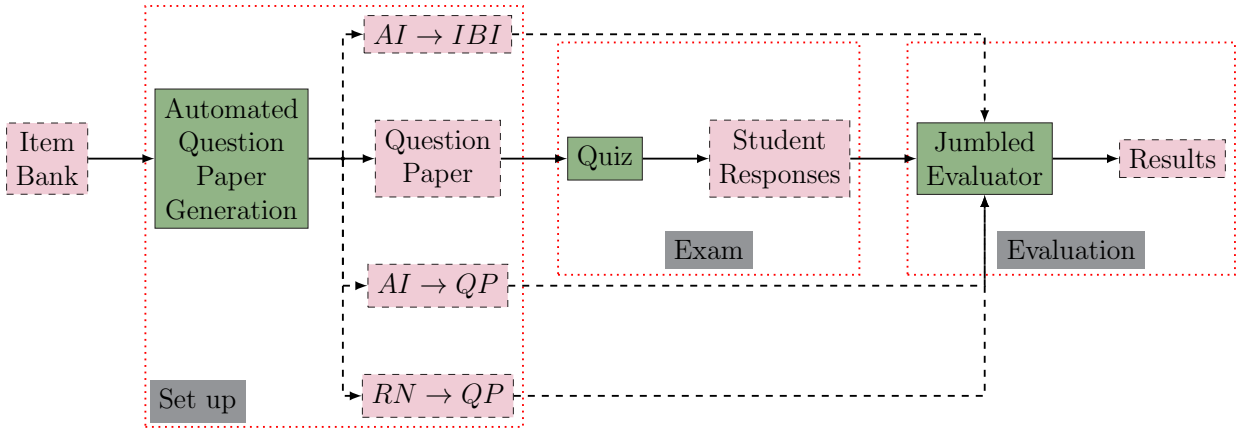


Figure 1.2: Jumbled Quiz Workflow

The question paper creation step uses the `genAIs` (generate assessment items) and genQPs (generate question papers) modules. Once the question papers are generated, quiz is conducted. Finally, the automated evaluation process takes place using the `JumbledEvaluator` module.

1.4 Getting Started with a Demo

Let's get a bit hands on by running a demo and feeling our way around with the system. We assume that you would be using a bash like command shell for interacting with the system. We call the evalobj directory as `APPHOME`.

1.4.1 Start the demo

Go to the demo directory:

```
cd APPHOME/test/j2/quizzes/demo
```

Clean up the directory:

```
./reset.sh
```

Take a look at the contents of the directory.

```
backup  config.json  packages  __pycache__  reset.sh
```

The only file here which is important to notice is the configuration file config.json. Every quiz will have its own configuration file with all the relevant meta-data pertaining to the quiz.

Its contents (as they stand on date)are reproduced here:

```
{
  "course name"       : "Software Design",
  "course code"       : "SE101",
  "assessment name"   : "demo",
  "assessment type"   : "jumbled",
  "roll number file"  : "/home/sujit/IIITB/projects/evalobj/test/j2/class.csv",
  "assessment home"   : "/home/sujit/IIITB/projects/evalobj/test/j2/quizzes/demo/",
  "number of items per assessment instrument" : "3",
  "items"             : [
    {
      "name" : "item1",
      "properties" :
        {
          "qtype" : "MCQ",
          "options" : "4",
          "marks"   : "1"
        }
    },
    {
      "name" : "item2",
      "properties" :
        {
          "qtype" : "MCQ",
          "options" : "4",
          "marks"   : "1"
        }
    },
    {
      "name" : "item3",
      "properties" :
        {
          "qtype" : "MCQ",
          "options" : "4",
          "marks"   : "1"
        }
    },
    {
      "name" : "item4",
      "properties" :
        {
          "qtype" : "MCQ",
          "options" : "4",
          "marks"   : "1"
        }
    },
    {
      "name" : "item5",
      "properties" :
        {
          "qtype" : "MCQ",
          "options" : "4",
          "marks"   : "1"
        }
    }
  ]
}
```

Here are the important fields of the configuration file and their meanings:

- `course name`: Name of the course
- `course code`: Course code as prescribed by the university
- `assessment name`: The exam name. Here it's "demo", but in practice it could something like "Quiz 1", "Mid-term Examination" etc.
- `assessment type`: This is the quiz type. Currently, there are two types available: `simple` and `jumbled`.
- `roll number file`: The full path of the file which contains the roll numbers of all the students taking the examination. Note that the roll number file needs to have the roll numbers in a particular way. As it stands today, the required format is that the roll numbers will be listed in column 1 of a CSV file starting with the second row, the first row having the header "Roll Number". For example:

```
Roll Number
rn1
rn2
rn3
rn4
```

- `assessment home`: This directory – for the application to know during runtime.
- `number of items per assessment instrument`: Number of questions you want on your question paper. Its relevance is based on the quiz type. In particular, a jumbled quiz will generate question papers each having so many questions on it. Simple quiz will simply ignore this field.
- `items`: This is the list of questions in the item bank. Here, we list 4 items named `item1`, `item2`, `item3` and `item4`. Each is an MCQ with 4 options and carrying 1 mark.

Note that everytime we are creating a new quiz, config.json has to be created. Smooth flow of the rest of the process would depend on our writing this file properly.

1.4.2 Generate the quiz

```
APPHOME/src/setup.py config.json
```

This generates all the necessary files for the quiz. Let's take a look at the contents of the directory:

```
assessment-instruments  backup  config.json  config.py  evaluati
gen.py  item-bank  packages  __pycache__  reset.sh
```

- `assessment-instruments`: This is the directory created – currently empty – to contain the question papers once they are generated.
- `config.py`: This is the file which contains the Python translation of the contents of the `config.json` file.

- `evaluation`: This is the evaluation directory. The `evaluate.py` file is the evaluator file which will be run once the quiz is conducted and submissions have come in.
- `gen.py`: This script needs to be run next to generate the quiz.
- `item-bank`: This contains the item bank, in the form of stubs of LaTeX text. These need to be filled up create the question papers. Note that there are five item stubs created, since that's the number of items mentioned in `config.json`. For example, the contents of the item1.tex is:

```
\question
\label{q:SE101:demo:item1}
\begin{enumerate}
        \item option 1
        \item option 2
        \item option 3
        \item option 4
\end{enumerate}
```

- `packages`: When the `gen.py` file is executed, it is this directory – currently empty – where the question papers will be generated.

1.4.3 Generate the Question Papers

Run the `gen.py` script.

```
python3 gen.py
```

The main effect of running this script would that the `packages` directory is no more empty. It contains one directory for each roll number. In there, among other files, you will find the question paper for that roll number: `packages/rn1/rn1.pdf`, `packages/rn2/rn2.pdf`, `packages/rn3/rn3.pdf` and `packages/rn4/rn4.pdf`.

There's one more change, though it needn't concern the user: a new file `AItoIBI.csv` has been generated. It's the mapping from roll assessment instrument to item bank items. As I type this, it looks like the following:

rn2	item5	item3	item4
rn3	item4	item1	item3
rn4	item4	item3	item2
rn1	item2	item4	item1

Again, it's not important for the user to know how to read it. However, for the curious minded, it means the following: Each row provides the source of each question in an assessment instrument (question paper). For example, for the assessment instrument `rn2` (first row), the first item is `item5` of the item bank, the second item is `item3` of the item bank and the third item is `item4` of the item bank.

This completes the process of generating the question papers.

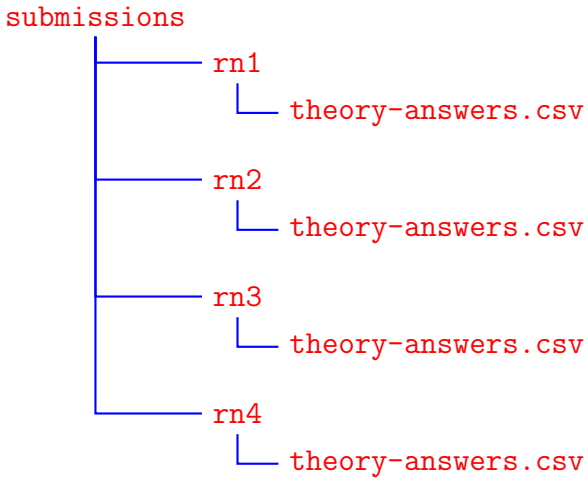
1.4.4 Administer the Quiz

At the time of administering the quiz, these question papers can be shared with the students using an appropriate external method, e.g. email, LMS or a shared drive. This is outside the purview of this system.

At the end of administering the quiz, the submitted answers are provided in the submissions directory in a particular format. In this demo, we simulate the process of answer submission by running the backup script provided.

```
cd backup
./restore.sh
cd ..
```

Once this is done, a new directory named submissions is created. Let's examine the contents of the submissions directory. You find the following contents:



Each CSV file `submissions/rn`*i*`/theory-answers.csv` is the answer sheet of `rn`*i*.

Another effect of this step is a bit of a cheating! It over-writes the `AItoIBI.csv file` to ensure that the answers in the submitted answer sheets match the item bank items properly[1]. However, if this seems confusing at this point, please simply ignore it.

The final effect of this step is addition of a new file `theory-answers.csv` in the `evaluation` directory. This file corresponds to the instructor's reference solution. Its contents are as follows:

1			
1	2		
3			
3	4		
1	2	3	4

1.4.5 Evaluate the Quiz

To evaluate the quiz, simply enter the evaluation directory and run the evaluate script:

[1]Note that the question paper to item bank items mapping was generated randomly. Hence, it is impossible to reproduce using an already stored set of answers.

```
cd evaluation
python3 evaluate.py
```

You will see the following output:

```
evaluating  rn1  ...
evaluating  rn2  ...
evaluating  rn3  ...
evaluating  rn4  ...
rn3      3.0
rn1      3.0
rn4      3.0
rn2      3.0
total marks =  roll number: reference
item scores: [1.0, 1.0, 1.0, 1.0, 1.0]
total score: 5.0
```

The result of evaluation is stored in the `result.csv` file in the `evaluation` directory. As of now, it looks like the following:

rn3	1.0	0	1.0	1.0	0	3.0
rn1	0	0	1.0	1.0	1.0	3.0
rn4	1.0	0	1.0	1.0	0	3.0
rn2	1.0	1.0	0	0	1.0	3.0

This tabulates the marks each roll number (row) has got for each question (column). For example, the first row says that `rn3` has scored `1.0` for `item1`, `0` for `item2`, `1.0` for `item3`, `1.0` for `item4` and `0` for `item5`. The total score for `rn3` is `3` as summarised in column 7. Note that the items for which `rn3` has score 0 (i.e. `item2` and `item4`) are those which were not there in its question paper, which means the student scored full-marks.

Chapter 2

Design

2.1 Question Paper Codes

2.1.1 Question Paper Generation

To discourage cheating in the class, we generate a set of question papers by randomly selecting n questions out of an item-bank of N questions. A set K of distinct *assessment instruments* are generated, numbered 0, 1, ..., $|K-1|$. We call them *assessment instruments*.

The *question paper generator* module G generates a set C of codes each of which can be mapped to any one of the assessment instruments of K. Each of the code c in C is finally mapped to one distinct question paper with c printed on it. This way, the students will not be able to identify which assessment instrument $k \in K$ their copy of the question paper belongs to. Each question paper will have an empty table called the *response table* on page one which will be used by the student to fill in his responses.

G also generates a map from assessment instrument to question order. This tells us the original question number of each item in the give assessment instrument. For example:

Figure 2.1 shows a possible mapping from assessment instruments to item bank items. The table can be interpreted as follows: There are 10 assessment instruments numbered 0 through 9. For each assessment instrument $AI \in K$ (here $|K| = 10$), there is a row in the table. Each cell in that row has the item bank item number for that item. For instance, for $AI = K[0]$, $AI[0] = 3$, $AI[1] = 5$ and so on.

This module will generate a map – called $QP \mapsto AI$ between *question paper code* to *assessment instrument*. For example:

$QP \mapsto AI = [0, 1, 2, ..., 9, 0, 1, 2, ...]$ could be one such mapping. It says that $QP \mapsto AI[0] = 0$ (i.e. the question paper with code 0 maps to assessment instrument number 0). Similarly, $QP \mapsto AI[11] = 1$ (i.e. the question paper with code 11 maps to assessment

0	3	5	1	4	9	12	15	2	10	2
1	4	1	2	11	6	7	5	14	8	12
	...									
9	5	12	2	1	6	7	3	11	8	10

Figure 2.1: Assessment Instrument Item to Item Bank Item map $AII \mapsto IBI$

instrument number 1) and so on.

2.1.2 TA's Job

The TA will note down following:

1. Question paper code for each roll number creating a *roll number* to *question paper code* map $RN \mapsto QP$.

2. transfer the responses into a CSV file corresponding to each student exactly as in the response table.

2.1.3 Automated Evaluation

The *response rearranger* refers to the $RN \mapsto QP$ and $QP \mapsto AI$ map to extract the assessment instrument for each roll number. Using this, the evaluator rearranges the responses in the order as per the item bank to create a rearranged response for the roll number n, $R'(n)$. This is given to the evaluator for final automated evaluation.