Yaksh: Facilitating Learning by Doing

Prabhu Ramachandran[§¶*], Prathamesh Salunke[‡], Ankit Javalkar[‡], Aditya Palaparthy[‡], Mahesh Gudi[‡], Hardik Ghaghada[‡]

https://youtu.be/ngrfZIgrnW4

Abstract—Yaksh is a free and open-source online evaluation platform. At its core, Yaksh focuses on problem-based learning and lets teachers create practice exercises and quizzes which are evaluated in real-time. A large array of question types like multiple choice, fill-in-the-blanks, assignment upload, and assertion or standard I/O based programming questions are available. Yaksh supports Python, C, C++, Java, Bash, and Scilab programming languages. In addition, Yaksh allows teachers to create full-blown courses with video and/or markdown text-based lessons. Yaksh is designed to be secure, easily deployable, and can scale-up to 500+ users simultaneously.

Introduction

Yaksh is created by the FOSSEE Python team. The *FOSSEE project* (http://fossee.in) based at IIT Bombay, is funded by the Ministry of Human Resources and Development, MHRD (http://mhrd.gov.in) of the Government of India. The goal of the FOSSEE project is to increase the adoption of Free and Open Source Software in Education in India. The project started in 2009 to develop and promote a variety of open source projects. FOSSEE's Python group attempts to promote the adoption of Python in India. More details on the activities of the Python group of FOSSEE have been presented earlier at SciPy 2016 [PR2016]. Yaksh was described briefly there. However, Yaksh has evolved considerably in the last few years. It has been used for several courses at IIT Bombay as well as online. In addition, Yaksh provides a simple interface to host a MOOC and we discuss this feature as well.

As part of FOSSEE's efforts we have created learning material for Python and have conducted hundreds of workshops on Python. We find that to effectively train people to learn to program, it is imperative to make them solve programming problems. Yaksh has been created by FOSSEE for this purpose.

Overview of Yaksh

Since the emergence of learning management system (LMS) and massive open online course (MOOC) providers, e-learning has grown significantly. Despite the ever increasing adopters, major platforms still use simple questions like multiple-choice questions and uploading of assignments from students as a means to evaluate students' performance. Yaksh seeks to improve on this.

It is well known that practice assignments and problem solving improve understanding. In the case of programming languages, this is especially so. Programming is a skill and to develop it, one must necessarily write programs. Providing an interface where users can attempt a question and immediately obtain feedback on the correctness of their program would be very useful both to a student and to a teacher. This same interface could also be used to assess the performance of the student. This is useful for the student to understand where they can improve and to the teacher to identify which concepts are not properly understood by the students. In the Indian context, a recent study [AM2017] found that even though there are many graduates with a computer science background, only 5% of the students are able to write programs with the correct logic. Indeed, our own experience is that many students learn computer science theoretically without writing too many computer programs. It is therefore important to provide a tool that facilitates practice programming and programming assessment.

In 2011, the first version of Yaksh was developed to administer programming quizzes for an online teacher training course that FOSSEE conducted. More than 600 teachers were trained and we wanted them to be able to write programs and have those corrected automatically. This work was presented at SciPy India 2011 [PR11].

Yaksh is a free and open-source online evaluation software that allows teachers to create courses and students to watch lessons and attempt tests which are evaluated immediately. Yaksh is designed to be used by a large number of users concurrently thereby making it apt for use in schools, colleges and other educational institutes for training a large number of students.

Yaksh is implemented in Python and uses Django (https://www.djangoproject.com/). It can be installed as a Django app with the *pip* command, thus allowing other Django based web projects to install the app within their project. The sources are available from: https://github.com/FOSSEE/online_test

To use Yaksh, one could sign-up on the official https://yaksh.fossee.in website or host it on one's own servers. The most standard and secure way to deploy Yaksh on a server is to build separate docker images using docker compose. Instructions for this are available in the Yaksh sources and are easy to setup.

For teachers, Yaksh provides a wide array of question types which include the basic question types like multiple choice, fill-in-the-blanks, assignment upload, etc. One can also add standard I/O and assertion-based questions for simple and basic programming

* Corresponding author: prabhu@aero.iitb.ac.in
§ Department of Aerospace Engineering
¶ IIT Bombay, Mumbai, India
‡ FOSSEE IIT Bombay, Mumbai, India

questions. For complex programs, teachers can add a hook-based test case which enable them to take the student answer and evaluate it in whatever way they want. Once the questions are created, they can create a question paper that can be added to a practice exercise or a quiz. The question paper can have a mixed set of fixed questions or a random set of questions selected from a pool of questions. In conjunction with quizzes, teachers can also add video or markdown-based lessons. Teachers can also monitor students in real-time during a test, as well as their overall progress for the course, thereby gaining insight on how students are performing.

Yaksh is designed to be easy-to-use by a student. All they have to do is sign-up, enroll for a course and start. They could go through the lessons, practice a few questions and then attempt the quiz, on which their performance is rated. While doing so, they get easy-to-understand feedback for any mistakes they make from the interface, thereby improving their answers.

Yaksh is being used extensively by the FOSSEE team to teach Python to many students all across India. Over 6000 students have used the interface to learn Python. It has been used in several courses taught at IIT Bombay and also for conducting recruitment interviews internally.

There are a few other open source software packages that do all or part of what Yaksh does.

nbgrader is a Jupyter Notebook plugin that can be used to grade programming assignments. The student submits jupyter notebooks containing code blocks which are then evaluated manually or automatically. nbgrader provides a very convenient Jupyter based interface. Instead, Yaksh offers instant feedback and grading, supports a variety of different languages, and also allows one to host a full course.

relate is similar to Yaksh in scope and goals. It allows a user to create a web based course with a grading interface quite similar to Yaksh. However, entering content into the software is based largely on YAML which is great for developers but not all end-users. Yaksh provides several question types and different ways to evaluate students' code.

Datacamp also provide several tools that are well suited for hosting very attractive courses online. It provides an easy to use and interactive interpreter for programming, which is also pluggable. However, it is not necessarily designed from the ground up for online assessment of students and live quizzes and exercise programs.

In this paper we first discuss how Yaksh may be installed, its features, and a high-level overview of its design and implementation. We then present some information on how Yaksh has been used at FOSSEE for a variety of tasks.

Installation and setup

Deployment of a web application for developement or for production purposes, should be as easy as possible. There are a few different ways of setting up Yaksh:

- Trial instance with Docker
- Trial instance without Docker
- Production instance using Docker and Docker compose.

Yaksh can be deployed with a limited number of commands using the invoke Python package to make the deployment as easy as possible.

Yaksh is written in Python and depends on Django and a few other Python dependencies. The dependencies can be installed using the pip package manager tool. It is recommended to use Yaksh along with Docker.

Yaksh can be cloned from the Github repository. To do this one can run:

```
$ git clone https://github.com/FOSSEE/online_test.git

$ cd online_test
```

One can then install the required dependencies, for Python 2, by running:

```
$ pip install -r requirements/requirements-py2.txt
```

or for Python 3, by running:

```
$ pip install -r requirements/requirements-py3.txt
```

It is recommended that one use Python 3 to run Yaksh.

Quickstart

The method discussed here allows a user to setup a local instance of Yaksh to try the platform for a limited number of users. Yaksh can be run within a demo instance on a local system to try the platform for a limited number of users. To set up a demo instance one can run:

```
$ invoke start
```

This command will start the code server within a docker environment.

In case docker is not available, the code server can also be run without docker by running:

```
$ invoke start --unsafe
```

However, this is not recommended since this leaves the base system potentially vulnerable to malicious code. In case one wishes to use this method, all Python dependencies will have to be installed using `sudo`.

In order to access the interface, one can run the web server using:

```
$ invoke serve
```

This command will run the Django application server on the **8000** port and can be accessed using a browser.

Production Setup With Docker

In order to setup Yaksh on a Production server with docker compose, one first needs to set certain environment variables. To do so, one can create a `.env` file with the following details:

```
DB_ENGINE=mysql

DB_NAME=yaksh

DB_USER=root

DB_PASSWORD=db_password

DB_PORT=3306
```

The local system needs to have Docker Compose installed. Then, one must navigate to the Docker directory:

```
$ cd /path/to/online_test/docker
```

Running the following commands will ensure that the platform is setup:

```
$ invoke build
```

```
$ invoke begin

$ invoke deploy --fixtures
```

The `build` command builds the docker images, the `begin` command spwans the docker containers and the `deploy` command runs the necessary migrations.

The demo course/exams

Since setting up a complete course with associated Modules, Lessons, Quizzes and Questions can be a tedious process for a first time user, Yaksh allows moderators to create a Demo Course by clicking on the 'Create Demo Course' button available on the dashboard.

One can then click on the Courses tab and browse through the Demo Course that has been just created.

One can read more about Courses, Modules, Lessons and Quizzes in the sections below.

Basic features of Yaksh

Once Yaksh is installed and running, one can create a full fledged course with lessons, practice, and evaluation based quizzes. Yaksh supports following languages such as Python, Java, C, C++, and Scilab. It provides several question types such as Single Correct Choice (MCQ), Multiple Correct Choice (MCC), Programming, Fill in the blanks, Arrange the options, Assignment upload. For simple and complex questions several test case types are provided such as standard input/output test case, Standard Assertion test case, Hook based test case, MCQ based test case, etc. The interface provides instant feedback for the student to improve their submissions. While administering quizzes or practice sessions, one can monitor the student's progress in real-time. This is particularly useful in practice sessions so as to help students who are not doing well. Finally, a student gets a certificate after successful completion of a course.

All the features are explained in detail in the workflow section.

Internal design

The two essential pieces of Yaksh are:

- Django Server
- Code server

Fig 1 shows the workflow for the evaluation of code submitted by a student and how this relates to these two pieces.

Django Server

Django is a high-level Python web framework. Django makes it is easy to create web applications, handles basic security issues, and provides a basic authentication system.

Django makes it easy to store information in a database by providing an object-relational mapping (ORM). This allows users to define the database tables at a very high level without having to write raw SQL queries.

Django provides a view controller to handle the requests sent from the client side. A view then interacts with the database using the ORM, retrieves data and pushes it to a template for rendering into an HTML page.

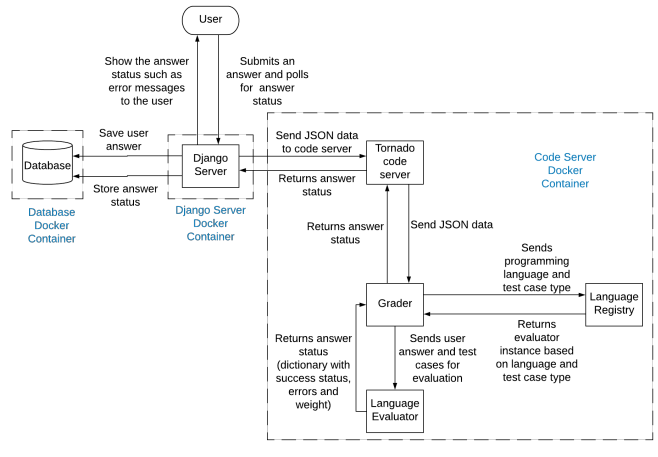

Fig. 1: *Flow diagram for code evaluation procedure*

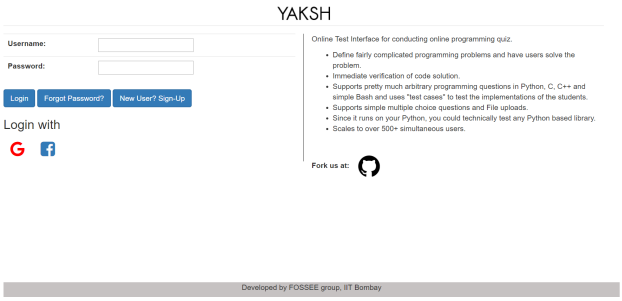

Fig. 2: *The Yaksh application login screen*

Authentication system

Yaksh uses the Django authentication system for handling basic user authentication, cookie-based user sessions and permissions for users and groups. Additionally, Yaksh uses email verification to provide users with a second layer of security while creating user accounts. To create an account on Yaksh, one can either go to the website and sign-up or can sign-up via the OAuth system provided for Google and Facebook accounts. By default the user is logged-in as a **student**, although the user can become a moderator if the user is added to the **moderator** group. Fig. 2 shows the login screen for Yaksh.

Yaksh models

A Django model is a Python class that subclasses `django.db.models.Model` representing the database table. Each attribute of the model represents a database table field.

The models for Yaksh are as follows:

- `User`
  This is the default model provided by Django for storing the user name, first name, last name, password etc.
- `Profile`
  This model is used for storing more information about a user such as institute, department etc.
- `Question`
  This model is used for storing question information such as name, description etc. Once the questions are created they are added in the question paper
- `TestCase`

This model is used for storing question test cases.
Different test case models are available which subclass the `TestCase` model. Some of these are -

  - `StandardTestCase`
    This model is used for test cases that use assertions to test success or failure.
  - `StdIOBasedTestCase`
    This model is used for test cases based on the standard output produced by a test.
  - `McqTestCase`
    This model is used for MCQ (single correct choice) or MCC (multiple correct choice) type of question.
  - `HookTestCase`
    This model is used for questions where there is a need for more complex testing. This model comes with a predefined function *check_answer* where the student answer (path to user submitted files for assignment uploads) is passed as an argument. The question creator can hence scrutinise the user answer in much more specific ways.
  - Fill in the blanks Test case
    This model supports integer, float, string types for fill in the blanks questions.
  - `ArrangeTestCase`
    This model is used for creating a test case with jumbled options which can be re-ordered by students.

- `Course`
  Is used for creating a course.
- `Quiz`
  Is used for creating a quiz.
- `QuestionPaper`
  Is used for creating a questionpaper for a quiz containing all the questions for the quiz.
- `AnswerPaper`
  Is used for storing the answer paper for a particular course and quiz.
- `Answer`
  Is used for storing the answer submitted by the user which are added to the answer paper.
- `Lesson`
  A lesson can be any markdown text which can have an embedded video of a particular topic.
- `LearningUnit`
  A learning unit can either be a lesson or a quiz.
- `LearningModule`
  A learning module can be any markdown text which can have an embedded video of a particular topic. A learning module contains learning units.

Code Server

The Code Server is an important part of Yaksh. The evaluation of any code is done through the code server. We have used the Tornado web framework to manage the asynchronous process generation. A `settings.py` file is provided which is used to specify various parameters necessary for the code server.

This settings file contains information such as:

- number of code server processes required to process the code.
- server pool port, a common port for accessing the Tornado web server.
- server host name, a server host for accessing the Tornado web server.
- a timeout to prevent infinite loops locking up a process.
- dictionary of code evaluators based on the programming language.

```
code_evaluators = {
 "python": {"standardtestcase": "yaksh.python_assertion_evaluator.PythonAssertionEvaluator",
            "stdiobasedtestcase": "yaksh.python_stdio_evaluator.PythonStdIOEvaluator",
            "hooktestcase": "yaksh.hook_evaluator.HookEvaluator"
           },
        }
```

Fig. 3: *Dictionary mapping of Python code evaluator*

A Tornado HTTP server is started with the specified server hostname and pool port from the settings. The server takes the following arguments -

- UID of an answer: This is the unique ID associated with an answer submitted. This is specifically required to poll the server for the status of the submitted answer.
- JSON Data: This contains all the data required for evaluation of a code answer, namely, user answer, language of the question, test cases associated with the question, and files required by the code, if any.
- User directory Every user is allotted a user directory, in which script files are executed. The path of this user directory is passed to the server.

The aforementioned arguments are passed to the Tornado server which takes the JSON data and sends it to a `Grader` for unpacking. The `Grader` unpacks the data, selects a language evaluator using a language registry and sends it to that language evaluator for evaluation. The language evaluator takes the user answer and evaulates it in the specified user directory. The evaluator then sends the output of the evaluation back to the Tornado server through the `Grader`. The Django server, meanwhile, keeps polling the Tornado server for the status of the evaluation. If the evaluation is complete, the Tornado server hands over the data to the Django server for saving and displaying.

Grader

Grader extracts the data such as language, test case type, test cases, user directory path from json metadata sent to it. It then creates the user directory from the path. Then it sends the test case type and language information to the language registry to get the evaluator. Once the evaluator is obtained, grader calls the evaluator and sends the test cases, user answer to the evaluator and code execution starts.

Language Registry

The language registry takes a programming language and test case type and generates a evaluator instance using the dictionary mapping in the settings file and returns the evaluator instance to the Grader.

Dictionary mapping of evaluator is as shown in Fig 3

For example say *Python* language and *standard assert* test case type are set during question creation, then Python assertion evaluator is instantiated from the dictionary mapping and the created instance is returned to grader.

Evaluators

Evaluators are selected based on the programming language and test case type set during the question creation.

For each programming language and test case type separate evaluator classes are available.

Each evaluator class subclasses `BaseEvaluator`. The `BaseEvaluator` class includes common functionality such as running a command using a Python subprocess, creating a file, and writing user code in the file, making a file executable etc.

There are several important aspects handled during code evaluation:

- Sandboxing
  A user answer might be malicious i.e. it might contain instructions which can access the system information and can damage the system. To avoid such a situation, all the code server process run as "nobody" so as to minimize the damage due to malicious code.
- Handling infinite loops
  There are chances that user answers contain infinte loops and lock up a process. To avoid this, code is executed within a specific time limit. If the code execution is not finished in the specified time, a signal is triggered to stop the code execution and sending a message to the user that code might contain an infinite loop. We use the `signal` module to trigger the `SIGALARM` with the server timeout value. Unfortunately, this does make our code server Linux/MacOS specific.
- Docker
  To make the code evaluation more secure all the code evaluation is done inside a docker container. Docker containers can also be used to limit the use of system resources such as CPU utilization, memory utilization etc.

Workflow of Yaksh

Instructor workflow

An instructor (also called the moderator) has to first create a course before creating a quiz, lesson or module. Before creating a quiz, the instructor has to create some questions which can be added to a quiz. The instuctor can create any number of questions through the online interface. These can be either multiple-choice, programming, assignment upload, fill in the blanks or arrange option questions. All these question types must be accompanied with several test cases. A sample Python question along with its test case is shown in the Fig. 4 and Fig. 5. The instructor can set minimum time for a question if it is part of an exercise. A question can have partial grading which depends on a weightage assigned to each test case. A question can have a solution which can be either a video or any code. This allows us to pose a question, ask the student to attempt it for a while and then show a solution.

A programming question can have test case types of standard assert, standard I/O or a hook. Fig. 5 shows a sample test case of standard assert type. In a similar way, the instructor can add test cases for standard I/O. For simple questions, standard assert and standard I/O type test cases can be used. For complex questions, hook based test case is provided where the user answer is provided to the hook code as a string and instructor can write some code to check the user answer. For other languages assertions are not easily possible but standard input/output based questions are easy to create. The moderator can also create a question with jumbled options and student has to arrange the options in correct order.

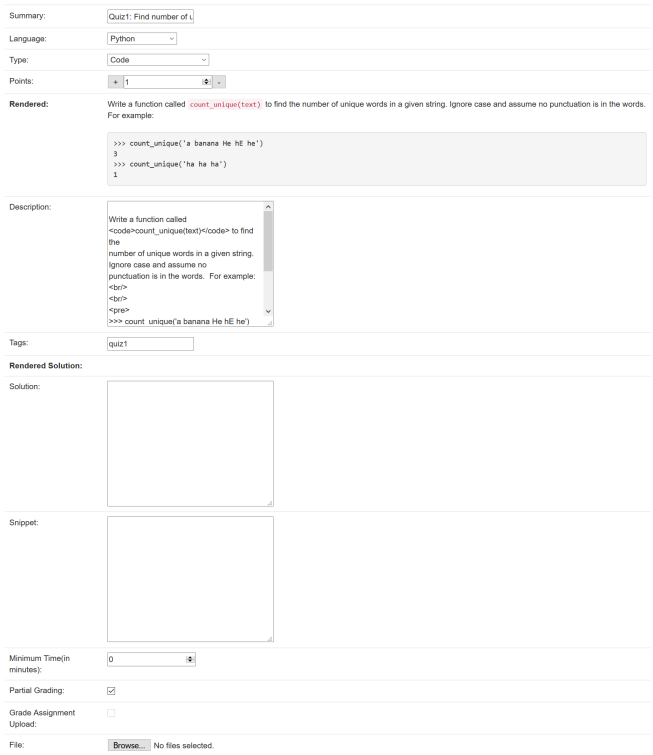

Fig. 4: *Question interface*

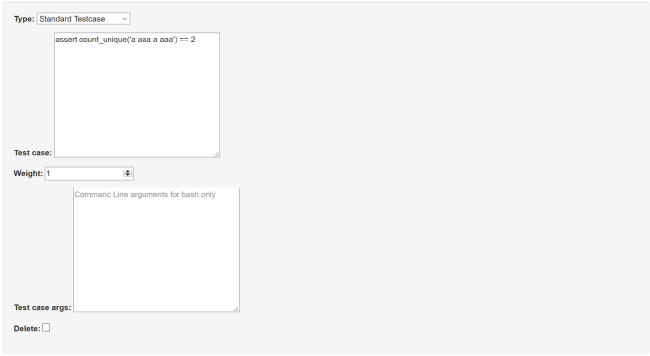

Fig. 5: *Sample Test case*

Detailed instructions on creating a question and test cases are provided at (https://yaksh.readthedocs.io).

The moderator can also import and export questions. The moderator then creates a quiz and an associated question paper. A quiz can have a passing criterion. Quizzes have active durations and each question paper will have a particular time within which it must be completed. For example, one could conduct a 15 minute quiz with a 30 minute activity window. Questions are automatically graded. A user either gets the full marks or zero if the tests fail. If a question is allowed to have partial grading then during evaluation the user gets partial marks based on passing test cases.

The moderator can then create learning modules. A module encapsulates learning units, i.e., lessons and quizzes. A lesson can have description either as a markdown text or a video or both. After lesson creation, the moderator can create modules. A module can have its own description either as a markdown text or a

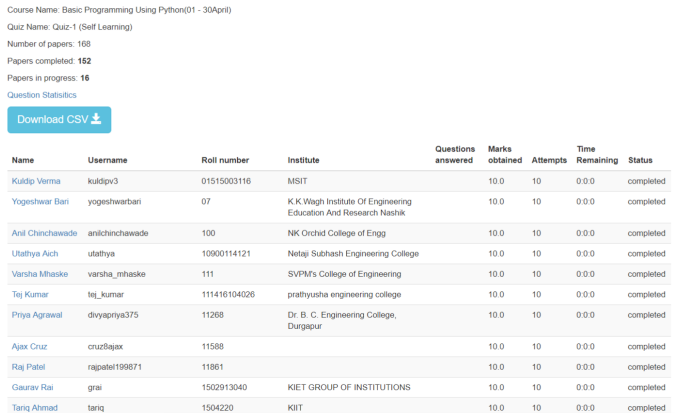

Fig. 6: *The moderator interface for monitoring a quiz on Yaksh.*

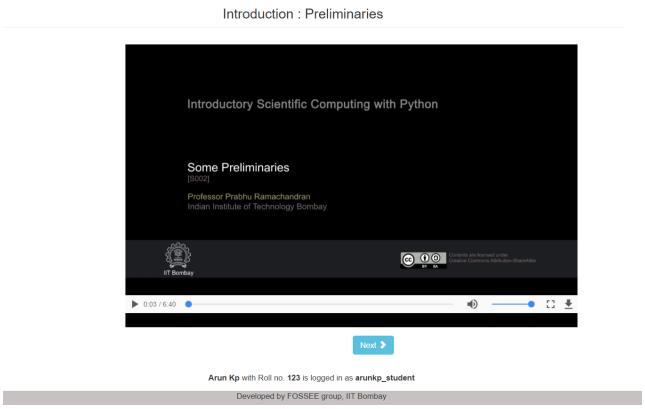

Fig. 7: *The interface showing a video lesson*

video or both. All the lessons and quizzes are added to the created module. The moderator can create any number of modules, lessons and quizzes as desired. These modules are added to a course.

Fig. 6 shows a monitor page for a quiz from one of the courses running on Yaksh. The instructors can also monitor students in real time during a quiz thereby gaining insight on how students are performing. The moderator can also view student progress for overall course, such as the number and percent of completed modules.

The moderator can regrade answerpapers using three ways:

- Answer paper can be regraded per quiz.
- Answer paper can be regraded per student.
- Answer paper can be regraded per question.

Student workflow

Working on the student side is relatively easy. After login, a student can view all the open courses or search for a course. Once the course is available, the student can enroll in a course. A student has to complete the course within a specified time. After enrolling, the student will be able to see all the modules and its units (Lessons/Quizzes) for the course. A student can view all the lessons and once the lessons are finished student can attempt the quiz. Fig. 7 shows a video lesson from the monthly running Python course.

Fig. 8 shows a MCQ question from a quiz. A student can select any one of the option and submit the answer.

Fig. 9 shows a programming question from a quiz in Python course. Once the student clicks on check answer, the answer is

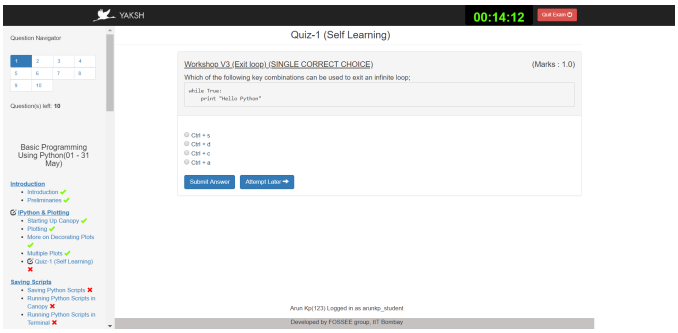

Fig. 8: *The interface for a multiple-choice question on Yaksh.*

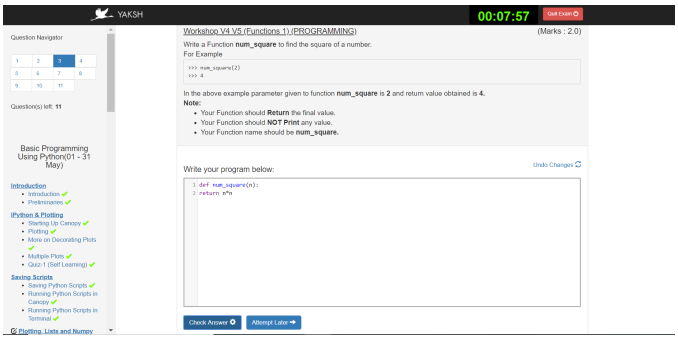

Fig. 9: *The interface for a programming question on Yaksh.*

sent to the code server for evaluation and the result from the code server is shown. From the Fig. 9 we can see that there is an indentation error in the code. Once the answer is submitted we get an indentation error message as shown in the Fig. 10. After submmiting the answer, if the answer does not pass the test case then student gets an assertion error as shown in the Fig 11.

Fig. 12 shows an StdIO based question. Once the answer is submitted we get the error output as shown in Fig 13. Fig 13 shows the user output and expected output separately, indicating

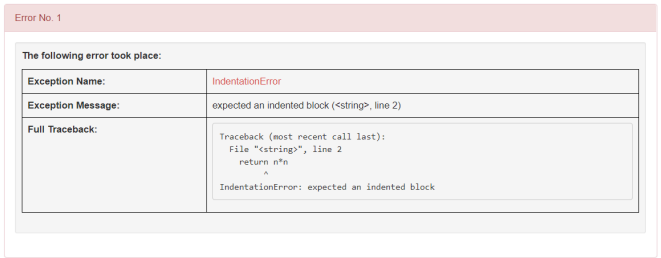

Fig. 10: *Error output after submitting the code answer.*

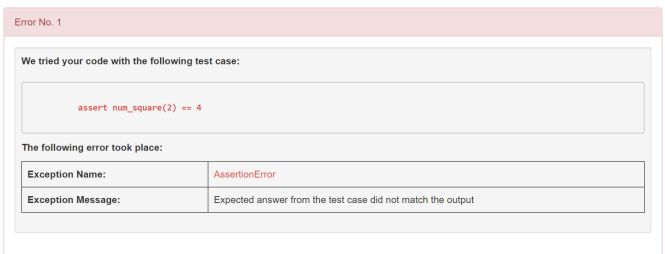

Fig. 11: *Asserrtion Error output after submitting the code answer.*

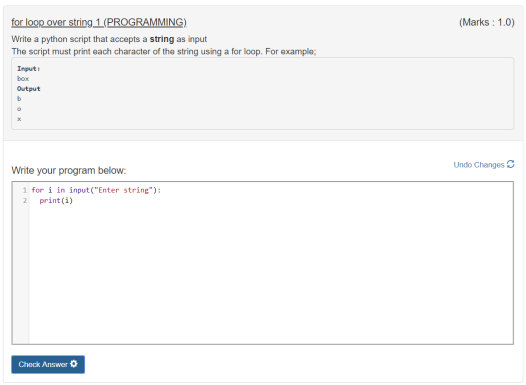

Fig. 12: *The interface for a stdio question type on Yaksh.*

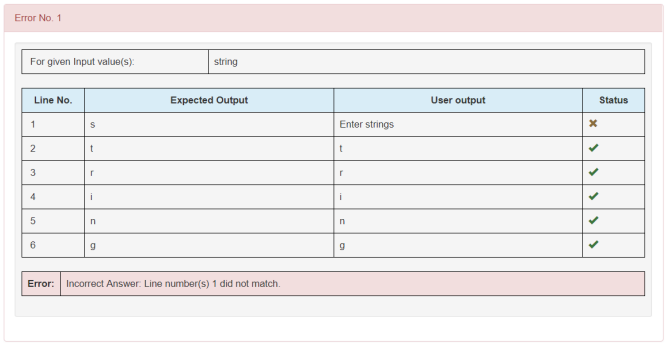

Fig. 13: *Error output for stdio question type.*

the line by line difference between user output and expected output making it easy to trace where the error occured.

Students can submit the answer multiple times, thereby improving their answers. Suppose a student is not able to solve a question, that question can be skipped and attempted later. All the submitted and skipped question's answers are stored so that the instructor can view all the attempts made by the student. Students can view the answerpaper for a quiz after completion.

Students can take the practice exercises where each question in the exercise is timed. Students must solve the question within the specified time, if not done within time then the solution for the question is shown and student can submit the answer once again. This makes it easy for the student to understand the mistake and correct it. These exercises run for infinite time and allows multiple attempts.

Once the course is completed, the student can view the course grades and download the certificate for that course which is generated automatically.

Supporting a new language

Adding a new language is relatively easy. In the settings file one needs to add a mapping for the evaluator corresponding to the language. An example for adding new evaluator is shown in Fig 14.

In the given Fig 14, Python is the programming language, standardtestcase, stdiobasedtestcase, hooktestcase are the test case type which are mapped to corresponding evaluator class. Here the values of the dictionary correspond to the full name of the Evaluator subclass, in this case `PythonAssertionEvaluator` is the class which is responsible to evaluate the code.

```
code_evaluators = {
 "python": {"standardtestcase": "yaksh.python_assertion_evaluator.PythonAssertionEvaluator",
            "stdiobasedtestcase": "yaksh.python_stdio_evaluator.PythonStdIOEvaluator",
            "hooktestcase": "yaksh.hook_evaluator.HookEvaluator"
            },
 "new_language": {
             "standardtestcase": "yaksh.new_language_assertion_evaluator.new_languageAssertionEvaluator",
             "stdiobasedtestcase": "yaksh.new_language_stdio_evaluator.new_languageStdIOEvaluator",
             "hooktestcase": "yaksh.hook_evaluator.HookEvaluator"
            }
           }
```

Fig. 14: *Dictionary mapping for new code evaluator*

Separate evaluator files needs to be created for all the test case types except the hook test case.

An evaluator class should define four methods `__init__`, `teardown`, `compile_code`, and `check_code`.

- `__init__` method is used to extract all the metadata such as user answer, test cases, files (if any for file based questions), weightage (float value), partial_grading (boolean value).
- The `teardown` method is used to delete all the files that are not relevant once the execution is done.
- All the code compilation tasks will be performed by the `compile_code` method. There is no need to add this method if there is no compilation procedure.
- The execution of the code is performed in the `check_code` method.

The `check_code` method must return three values -

- success (`bool`) - indicating if code was executed successfully and the student answer is correct
- weight (`float`) - indicating total weightage of all successful test cases
- error (`str`) - error message if success is false

Some experiences using Yaksh

During its inception in 2011, Yaksh was designed as an evaluation interface with the idea that anyone can use Yaksh to test and grade the programming skills of students. As an evaluation interface, Yaksh was first used to evaluate 600 teachers. Since then, Yaksh has been used for teaching students, especially for courses at IIT Bombay and for conducting employment hiring tests within FOSSEE. With the introduction of Python Workshops (https://python-workshops.fossee.in/), an initiative of FOSSEE to remotely train students and teachers across India, Yaksh has since been refactored around the MOOC ideology, introducing the ability to learn with an emphasis on hands-on programming. We look at the various activities where Yaksh is used below.

Courses at IIT Bombay

Yaksh has been used as a online learning and testing tool for some courses at IIT Bombay. Yaksh is used to teach Python to some undergraduate students. These courses have served as a test-bed for the software. Thus far, about 300 students from IIT Bombay have been taught using Yaksh.

Usage for Python Workshops

In early 2017, FOSSEE started conducting remote Python workshops in technical colleges across India. These workshops consist of several sessions spread through one or three days, depending on the type of the course an institute chooses. A session typically

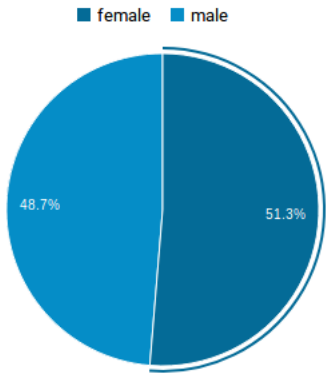

Fig. 15: *Male:Female ratio of visitors on Yaksh.*

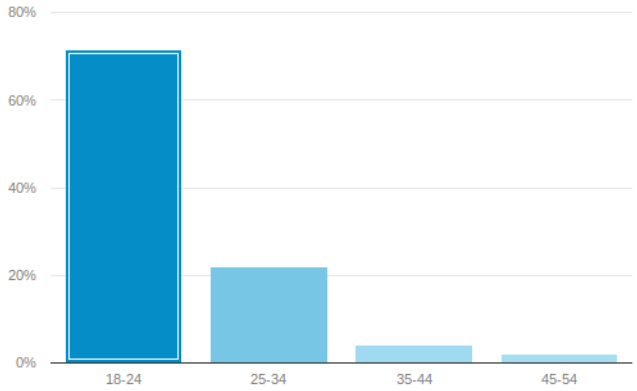

Fig. 16: *Age breakdown of visitors on Yaksh.*

begins with screening a video tutorial inside the venue. The tutorials are followed by a demanding set of exercises and quizzes, both of which are conducted on Yaksh. This is followed by brief Q&A sessions with the remote Python instructors from FOSSEE. Finally a certificate is awarded to those students who successfully finish the course. Apart from this, Yaksh also hosts a monthly, self learning online course, consisting of the same workshop materials and some bonus contents. Here are some statistics based on these activities -

1) As of mid 2018, around 13,000 active users are on Yaksh, with more expected to join by the end of the year.
2) Rapidly growing user base with about 730, 4500 and 7500 user registrations for year 2016, 2017 and mid-2018 respectively.
3) 100+ institutes have conducted the workshop with about 6000 students participating and about 3600 students obtaining a certificate.
4) For the first three months of the Python self learning course, an estimate of 3500 students enrolled with 1200 completing the course within the time frame and 400 students obtaining a passing certificate.
5) An equal ratio of male to female participants with most users from the age of 18-24 as seen in the Figures. 15 and 16.
6) Average time spent on the website by a user is around 30 minutes.
7) Major users are from tier 1 cities of India, regarded as highly developed IT hubs like Hyderabad, Bengaluru, Pune, and Mumbai.

Usage for hiring

One surprising use case for Yaksh has been as a tool for evaluating employment candidates by conducting tests. Yaksh has been used several times for hiring for teams functioning inside the FOSSEE project.

Plans

The team behind Yaksh is devoted to further improving user experience for both moderators and students. This includes addition of features like Instant Messaging (IM) service for moderators and teachers to guide and solve students' doubts in real time. The team also plans to add support for more programming languages to include a larger question base. Moderators will have facility to do detailed analysis on student performance in future.

Many colleges and schools in India do not have good internet access. We are hoping to make it easy for such institutions to locally host Yaksh using a bootable USB drive.

In addition, we are planning to make it easy for students to download the course materials and any videos in order to view the lectures offline.

For moderators, a stable web-API is being designed for other websites to harness the power of Yaksh. With this API, moderators could be able to embed lessons and quizzes available on Yaksh in Jupyter notebooks.

Conclusions

As discussed in this paper, Yaksh is a free and open source tool can be used effectively and extensively for testing programming skills of students. The features provided by Yaksh facilitates teachers to automate evaluation of students in almost real time, thereby reducing the grunt work. With addition of MOOC like features, students can learn, practice and test their programming abilities within the same place. The Python team at FOSSEE continues to promote and spread Python throughout India using Yaksh.

Acknowledgments

FOSSEE would not exist but for the continued support of MHRD and we are grateful to them for this. This work would not be possible without the efforts of the many FOSSEE staff members. The past and present members of the project are listed here: http://python.fossee.in/about/ the authors wish to thank them all.

References

[PR2016] Prabhu Ramachandran, Spreading the Adoption of Python in India: the FOSSEE Python Project", Proceedings of the 15th Python in Science Conference (SciPy 2016), July 6-12, 2016, Austin, Texas, USA. http://conference.scipy.org/proceedings/scipy2016/prabhu_ramachandran_fossee.html

[kmm14] Kannan Moudgalya, Campaign for IT literacy through FOSS and Spoken Tutorials, Proceedings of the 13th Python in Science Conference, SciPy, July 2014.

[FOSSEE-Python] FOSSEE Python group website. http://python.fossee.in, last seen on May 7 2018.

[PR11] Prabhu Ramachandran. FOSSEE: Python and Education, Python for science and education, Scipy India 2011, 4th-11th December 2011, Mumbai India.

[AM2017] 95% engineers in India unfit for software development jobs, claims report. http://www.aspiringminds.com/automata-national-programming-report