# Designing an Intelligent Tutoring System Across Multiple Classes

[Extended Abstract]

Laura O. Moraes
Systems and Computing Department
(COPPE/PESC)
Universidade Federal do Rio de Janeiro (UFRJ)
lmoraes@cos.ufrj.br

Carlos Eduardo Pedreira
Systems and Computing Department
(COPPE/PESC)
Universidade Federal do Rio de Janeiro (UFRJ)
pedreira56@gmail.com

## ABSTRACT

The ability to understand a person's knowledge is important in educational settings. This can be used to recognize gaps in knowledge and to diagnose misunderstandings and misconceptions. This paper presents an intelligent tutoring system created to gather student knowledge data and the proposed methodology to generate the datasets. We asked 14 professors to determine the concepts found in a set of problems and we compare the student behavior found in each methodology.

## Keywords

open datasets, intelligent tutoring system, educational data mining, computer science education

## 1. INTRODUCTION

In this work, we have two main motivations: create an online learning environment to gather data about students' knowledge and stimulate students to learn the Python language.

It is important to understand why students succeed or fail when taking a course so we can improve teaching methods by identifying students' needs and to provide personalized education. Smart learning content is defined as visualizations, simulations and web-based environments that provide outputs for students based on the students' input [2]. The adoption of smart learning content in classrooms and in self-learning environments motivates students [1, 7, 11, 12], improves student learning, decreases student dropout or failure [1, 9, 8, 6] while increasing their self-confidence, especially in female students [9].

Also, Python is a general-purpose language, which means it can be used in a large variety of projects. This can be great to stimulate students, since they can work in projects they actually relate to. Python is also user-friendly and for the past seven years, it has been the fastest-growing major programming language [17], being correlated with trending careers, such as DevOps and Data Scientist [20]. However, according to the 2015 review [6], only 11% of the Educational Data Mining and Learning Analytics papers about programming courses reported using Python as the course language.

These factors motivate our objective: the creation, deployment and use of online intelligent systems in Introduction to Programming classes using the Python language as a way to uncover students' difficulties, to understand their knowledge and to provide timely feedback to keep students engaged.

The main contributions of this paper are: 1) generating an anonymous open-source students interaction database using the Python language with corresponding solutions and, 2) developing the Machine Teaching system[1], an ITS with a built-in recommendation engine.

## 2. DATA ACQUISITION METHODOLOGY

This section presents the web system developed to acquire student data and interact with students and educators. The system architecture is proposed as an improvement from a common architecture from the Intelligent Tutoring System literature. We also present the methodology used to collect student data.

### 2.1 System Architecture

To capture students' data, a web-system was designed and implemented. Intelligent Tutoring Systems (ITS) are systems designed to assist the tutoring of students on a personalized level. They have been around since the 1960s [22], having its first formal definition in the 1990s [18]. Nowadays, there are several different ITS covering a broad range of subjects with success used by hundreds of thousands of students a year. Ihantola et al. [6] present a table showing a list of them and their supported programming languages.

Ihantola et al. [6] also define a common architecture for these systems in their 2015 review. Common front-end features



[1] http://www.machineteaching.tech

among them are an IDE for students to write, edit and execute code, a submission interface (can be embodied in the IDE or in a separate part of the system), feedback for students' actions and visualization schemes for teachers and researchers. Back-end usually supports saving data in some kind of storage, usually a relational database. However, part of this proposal is to build an integrated personalized exercise recommendation engine within the system. Therefore, none of the existing systems could be used without modifications. To avoid legacy code and to have a better control of the desired features and captured student data, it was decided to build one from scratch using open-source Python and Javascript libraries. In addition, the classes in which the system was employed, use a modularization based approach to teach imperative programming [4]. Each question requires self contained modules of code as answer which translates in Python as functions. So, the proposed system also had to take this format into consideration to correct students' assignments. Besides these adaptations, it shares most of the functionalities with the other systems. Fig. 1 illustrates an abstract view of the system, adapted from Ihantola et al. [6]. The main differences are the built-in recommendation engine, that controls what exercises are shown in the IDE for the student to solve, the way of correcting students assignments and the extra collected data, like time spent typing and the total time spent solving the question.

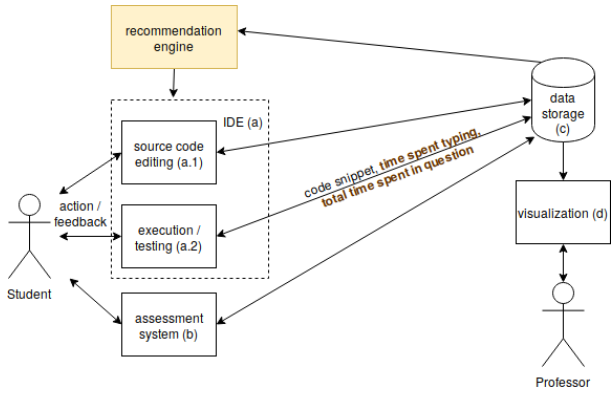


**Figure 1: Abstract view of the system architecture.**

The exercises currently available in the system belong to either Applying or Creating categories in Bloom's Taxonomy [21]. The students are presented with a problem and they should write the expected answer in a free-text coding format. For each exercise, a test case function generator was defined to correct the results. The students get feedback every time they submit an answer and they can see whether they passed or failed a unit test case. If they get all of them correct, the task is considered done and the student may move on to another problem. The system saves a state every time a student submits an answer.

## 2.2 Data Acquisition

For the data acquisition process, we used the system to collect the data in two approaches. The system was either used a single time for revision purposes (revision dataset), and throughout a whole semester (semester dataset).

The first approach uses 48 CS1 problems crawled from four Python web tutorials: Practice Python [15], Python School [19], Python Programming Exercises [5], and W3Resource [23]. The chosen sources provided the exercise statements along with the exercise code solution. Students' responses to these exercises were collected in 10 different *Introduction to Programming* courses during 2 semesters. Students were assigned two different strategies: either the system showed random problems or they would follow a predefined path. The students were introduced to the system at the end of the semester before their finals exams, or at the beginning of the next semester. In both cases, the exercises were supposed to act as revision exercises of all CS1 content.

In our second approach, we accompanied the students throughout a whole semester in four different *Introduction to Programming* courses (they had the same syllabus, but different professors). Every week, an exercise list concerning the subject given in class was available in the system. They had one week as a deadline to finish them and their performance on these lists composed part of their final grade. In total, they had to solve 65 problems.

# 3. RESULTS

This section presents statistics for both datasets, comparing the different behaviors found in them.

## 3.1 Revision Dataset

In total, there are 3,632 records from 192 students with an average of 18.4 attempts in 4.4 problems. Also, the dataset is imbalanced: there are 764 (21%) successful attempts and 2,868 (79%) failed attempts. It means that, on average, each student attempts a problem 4 times before getting all test cases correct in the fifth. Some simple statistics for the dataset are shown in Table 1. Fig. 2 is a histogram showing the distribution of success and failures per problem. Similar behavior is found in the distribution of success and failures attempts per student. Both success attempt distributions have smaller variance and smaller mean than the corresponding fail attempt distributions. This is expected since the students are given several tries to submit a correct response before moving on to the next problem.

**Table 1: Revision dataset statistics**

| | Avg | Median | Min | Max |
|---|---|---|---|---|
| attempts per question | 75.67 | 55 | 10 | 304 |
| attempts per student | 18.44 | 11 | 2 | 266 |
| different students per question | 18.17 | 14 | 4 | 71 |
| different questions per student | 4.43 | 3 | 1 | 44 |

For this dataset, we asked 14 professors to indicate the concepts needed in each question for a student to be able to solve it based on the exercise solution. The teaching experience of the professors range between 2 to 20 years and they were not necessarily involved in the classes' participating in this work. Each professor should associate up to three concepts (from 15 available) to 15 randomly assigned code snippets. On average, each code received four evaluations. From the 54 code snippets, 37 of them had one or more concepts in which all the professors agreed. If we lower the threshold to contabilize where at least 75% of the professors agree (3

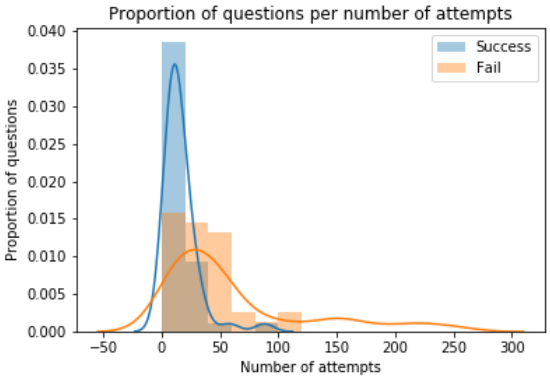


**Figure 2: Distribution of questions' success and fail attempts**

out of 4), this number increases to 53. So, we decided to use this 75% threshold of agreement to relate the concepts and the exercises. The concepts are not mutually exclusive and a problem can be assigned to more than one concept. Around 45% of the problems involved the loop concept and 40% the conditional concept. About 22% involved working with strings and 12% involved math questions.

## 3.2 Semester Dataset

This dataset is 7.5 times bigger in number of attempts than the previous one, containing 27,491 attempts records. However, since it accompanied the same students for an entire semester, the number of students is actually smaller: 181 different students. Simple statistics for the dataset are shown in Table 2. This dataset has a slightly higher success rate than the previous one, although it is still very imbalanced. It contains 6,849 (24.91%) success attempts against 20,642 (75.09%) unsuccessful ones. This can be explained by the fact that each weekly set of exercises covers mostly what was seen in class in the same week, so students did not have a lot of time to forget the subject [3, 10, 13, 14, 16], in contrast to the review exercises that were done before finals or at the beginning of the next semester.

**Table 2: Semester dataset statistics**

| | Avg | Median | Min | Max |
|---|---|---|---|---|
| attempts per question | 422.94 | 349 | 85 | 1291 |
| attempts per student | 151.88 | 114 | 2 | 1002 |
| different students per question | 87.22 | 87 | 39 | 145 |
| different questions per student | 31.32 | 31 | 1 | 65 |

Another interesting difference between the datasets is the average number of attempts per question. Whereas in the revision dataset it is lower than the total number of questions, indicating that not all the questions were answered, in the semester dataset it is two times the total number of available questions, indicating that students redid the exercises even though they had already succeeded at it. This can be used to measure if they are using the system and the exercises to study and review the content, for example.

Fig. 3 shows the quantity of students per number of questions histogram. We can notice that only 15 out of the 181 (8%) students attempted more than 59 out of the 65 (92%) different exercises. The exercise distribution is actually quite flat. If we divide it in three equal parts, approximately one third of the class (64 students, 35%) attempted only one third of the exercises, one third (60 students, 33%) attempted two thirds of the exercises and the last third (57 students, 31%) attempted between 45 and 65 exercises. When provided in real time, this information can be used by the professors to find students that are having difficulties finishing exercises and provide personalized assistance.

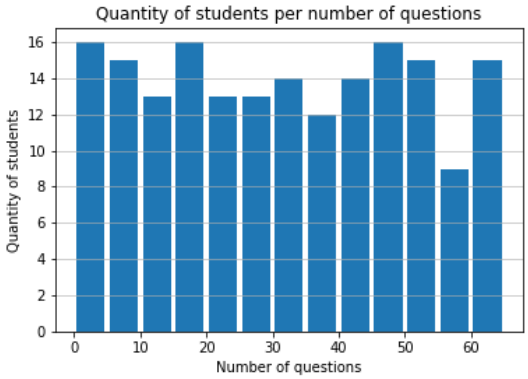


**Figure 3: Distribution of students per number of questions**

# 4. CONCLUSIONS

In this paper, we presented the Machine Teaching ITS system, developed to assist students and professors in modularized function-based Python classes. We also presented and analyzed the two approaches in which data were collected. For the first approach, we invited professors to associate the needed concepts for each question and calculated their agreement rate. In general, they agree in at least two concepts for each exercise. For the second approach, we visualized the distribution of the dataset, which can be used by professors during the semester to identify students with difficulties.

## 4.1 Future work and call for collaboration

We are currently working on integrating a recommendation engine in the system. After this step, we will perform some A/B tests with professors and students and collect their opinion on the available tools. In addition, there is a lot to be explored on these datasets. For example, we would like to study the differences in behavior between revision and semester students, investigate student procrastination, infer student knowledge based on their answers, research temporal learning effect, among other ideas. We invite researchers interested in exploring the dataset to contact the authors.

# 5. ACKNOWLEDGMENTS

We would like to thank the professors that contributed with the research, either by adopting the tool in class or by evalu-

ating the results. This work was supported by CNPq (141089/2016-4) and was supported in part by FAPERJ (E26/202.838/2017-CNE), CAPES (PROEX - 1201036), and CNPq (306258/2019-6).

# 6. REFERENCES


[1] L. Benotti, F. Aloi, F. Bulgarelli, and M. J. Gomez. The effect of a web-based coding tool with automatic feedback on students' performance and perceptions. In *Proc. 49th ACM Tech. Symp. Comp. Sci. Educ.*, SIGCSE '18, pages 2–7, Baltimore, Maryland, USA, Feb. 2018. http://doi.acm.org/10.1145/3159450.3159579.

[2] P. Brusilovsky, S. Edwards, A. Kumar, L. Malmi, L. Benotti, D. Buck, P. Ihantola, R. Prince, T. Sirkiä, S. Sosnovsky, J. Urquiza, A. Vihavainen, and M. Wollowski. Increasing adoption of smart learning content for computer science education. In *Proc. Work. Group Rep. 2014 Innov. Technol. Comput. Sci. Educ. Conf.*, ITiCSE-WGR '14, page 31–57, Uppsala, Sweden, June 2014. https://doi.org/10.1145/2713609.2713611.

[3] B. Choffin, F. Popineau, Y. Bourda, and J.-J. Vie. DAS3H: Modeling Student Learning and Forgetting for Optimally Scheduling Distributed Practice of Skills. In *Proc. 12th Int. Conf. Educational Data Mining*, pages 29–38, Montreal, Canada, July 2019.

[4] C. Delgado, J. Da Silva, F. Mascarenhas, and A. Duboc. The teaching of functions as the first step to learn imperative programming. In *Anais do Workshop sobre Educação em Computação (WEI)*, pages 388–397. Sociedade Brasileira de Computação - SBC, Jan. 2016. https://doi.org/10.5753/wei.2016.9683.

[5] J. Hu. Python programming exercises, 2018. https://github.com/zhiwehu/Python-programming-exercises.

[6] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. A. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proc. 2015 ITiCSE Work. Group Rep.*, ITICSE-WGR '15, pages 41–63, Vilnius, Lithuania, July 2015. http://doi.acm.org/10.1145/2858796.2858798.

[7] I. Jivet, M. Scheffel, M. Specht, and H. Drachsler. License to evaluate: Preparing learning analytics dashboards for educational practice. In *Proc. 8th Int. Conf. Learn. Analytics Knowl.*, LAK '18, page 31–40, Sydney, New South Wales, Australia, Mar. 2018. https://doi.org/10.1145/3170358.3170421.

[8] E. Johns, O. Mac Aodha, and G. J. Brostow. Becoming the expert-interactive multi-class machine teaching. In *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, pages 2616–2624, 2015.

[9] A. N. Kumar. The effect of using problem-solving software tutors on the self-confidence of female students. In *Proc. 39th SIGCSE Tech. Symp. Comput. Sci. Educ.*, SIGCSE '08, page 523–527, Portland, OR, USA, Mar. 2008. https://doi.org/10.1145/1352135.1352309.

[10] A. Lalwani and S. Agrawal. What Does Time Tell? Tracing the Forgetting Curve Using Deep Knowledge Tracing. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, and R. Luckin, editors, *Artif. Intell. Educ.*, LNCS, pages 158–162, June 2019. https://doi.org/10.1007/978-3-030-23207-8_30.

[11] A. Latham, K. Crockett, D. McLean, and B. Edmonds. A conversational intelligent tutoring system to automatically predict learning styles. *Comput. & Educ.*, 59(1):95 – 109, Aug. 2012. https://doi.org/10.1016/j.compedu.2011.11.001.

[12] R. Lobb and J. Harlow. Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51, Feb. 2016. https://doi.org/10.1145/2810041.

[13] K. Nagatani, Y. Y. Chen, Q. Zhang, F. Chen, M. Sato, and T. Ohkuma. Augmenting knowledge tracing by considering forgetting behavior. In *Proc. World Wide Web Conf.*, WWW '19, San Francisco, CA, USA, May 2019. http://doi.org/10.1145/3308558.3313565.

[14] P. Nedungadi and M. S. Remya. Incorporating forgetting in the Personalized, Clustered, Bayesian Knowledge Tracing (PC-BKT) model. In *Proc. 2015 Int. Conf. Cogn. Comput. Inf. Process.*, Noida, India, May 2015. https://doi.org/10.1109/CCIP.2015.7100688.

[15] M. Pratusevich. Practice python, 2017. www.practicepython.org.

[16] Y. Qiu, Y. Qi, H. Lu, Z. Pardos, and N. Heffernan. Does Time Matter? Modeling the Effect of Time with Bayesian Knowledge Tracing. In M. Pechenizkiy, T. Calders, C. Conati, S. Ventura, C. Romero, and J. Stamper, editors, *Proc. 4th Int. Conf. Educational Data Mining*, pages 139–148, Eindhoven, the Netherlands, July 2011.

[17] D. Robinson. The incredible growth of python. https://stackoverflow.blog/2017/09/06/incredible-growth-python/, 2017. Online; accessed 12-June-2020.

[18] J. Self. Theoretical foundations for intelligent tutoring systems. *J. Artif. Intell. Educ.*, 1(4):3–14, Sept. 1990.

[19] S. Sentance and A. McNicol. Python school, 2016. https://pythonschool.net/.

[20] Stack Overflow. Developer survey results 2018. https://insights.stackoverflow.com/survey/2018/, 2019. Online; accessed 12-June-2020.

[21] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for cs assessment. Jan. 2008.

[22] K. Vanlehn. The behavior of tutoring systems. *Int. J. Artif. Intell. Ed.*, 16(3):227–265, Aug. 2006.

[23] W3Resource. W3resource, 2018. https://www.w3resource.com/python/python-tutorial.php.