IEEE 45th Annual Frontiers in Education Conference

Scaling Youth Development Training in IT Using an xMOOC Platform

Martin Löwis[*], Thomas Staubitz[†], Ralf Teusner[†], Jan Renz[†], Christoph Meinel[†] and Susanne Tannert[†]
[*]Beuth Hochschule
Berlin, Germany
Email: loewis@beuth-hochschule.de
[†]Hasso Plattner Institute
Potsdam, Germany
Email: firstname.lastname@hpi.de

***Abstract*—The paper at hand evaluates the Massive Open Online Course (MOOC) *Spielend Programmieren Lernen (Playfully learning to program)*, an effort to scale the youth development program at the Hasso Plattner Institute (HPI) for a larger audience. The HPI has a strong tradition in attracting children and adolescents to take their first steps towards a career in IT at an early age. The *Schülerakademie*, the *Schülerkolleg*, the *Schülerklub*, and the support for the *CoderDojo* in Potsdam are some of the regular activities in this context to take youngsters by the hand and supply them with material and guidance in their mother tongue. With the emergence of MOOCs and the success of HPI's own MOOC Platform—openHPI—it was a natural step to develop a course to address an audience that is only marginally represented in openHPI's regular courses: school children and adolescents. A further novelty for openHPI in this course was the focus on teaching programming with a high percentage of obligatory hands-on tasks. Particularly for this course, a standalone tool allowing participants to write and evaluate code directly in the browser—without the need to install additional software—has been developed. We will compare this tool to a small selection of similar approaches on other platforms. As it will be shown, the course attracted a far more diverse audience than expected, and therefore, also needs to be seen in the context of spreading digital literacy amongst wider parts of society. In this context we also will discuss the significant differences in the usage of the forum between the course *Spielend Programmieren Lernen* and the course *In-Memory Databases*, a more traditional openHPI course.**

Keywords*—K-12; openHPI; Automated Assessment; E-Learning; Online Learning; Programming; Python; MOOC K-12; openHPI; Automated Assessment; E-Learning; Online Learning; Programming; Python; MOOC ;

I. Introduction

The Hasso Plattner Institute (HPI) in Potsdam, Germany is a university-level institute offering study programs in IT-Systems Engineering. Since its early days, youth development training has been a strong focus of the HPI. In order to raise IT-skills and interest of (junior-)high school students in programming subjects, the Hasso Plattner Institute has established the *Schülerakademie* (lit. "academy for pupils") offering various activities in this area. Part of these activities are organized by the *Schülerklub* ("pupils' club"). Bachelor and master students at the HPI are encouraged to participate in so called *Klubs*, which are self-organized by the students, alongside their regular studies, as a part of their socio-emphatical education at the Institute. The *Schülerklub* is one of those. The members of this Klub are working with school children to raise the enthusiasm for computer science amongst this clientele. They organize events and activities for children and adolescents who are interested in getting first hand information about studying IT Systems Engineering at the HPI. These events include e.g. a summer camp of several days' duration for girls and boys from all over Germany and the HPI CodeNight. The HPI also sponsors third party events, such as the CoderDojo, with locations and support. The *Schülerkolleg* is a special program for school children from 7th to 12th grade, who are particularly interested in computer science and mathematics. For one year, every Tuesday afternoon, the participants of the program meet to experiment with new information technologies and their foundations. They are supported by faculty and students of the HPI as well as four teachers who are sent by the school authority of Brandenburg. Even though the HPI facilitates those numerous activities for high-school students, they are limited by given capacities, such as staff, room size, and number of available computers.

openHPI started in 2012 and, thereby, is one of the first platforms that transferred the idea of MOOCs to Europe and Germany. Run by the HPI, it has offered about 20 self produced courses on various ICT topics since September 2012—hosting between 5,000 and 17,000 enrolled users per course. Typical openHPI courses follow a classical xMOOC schema of a six-week course with several ungraded self-tests and one graded assignment per week. The courses are concluded with a final exam, which also is graded. For each of these graded assignments, the participants can achieve a certain amount of points. To be eligible for a graded record of achievement, a participant has to achieve at least 50% of the overall maximum course score [1]. In this specific course (Python2014), the setting was a little different as the focus was moved to hands-on exercises. The only way to gain grade relevant points was by solving practical programming exercises. Weekly assignments and a final exam have not been offered, ungraded self-tests were available, however.

A number of improvements on the platform enabled us to implement a new course model with a focus on hands-on tasks and practical exercises. A standalone tool has been developed that allows the participants to write, run, and evaluate Python code in the browser. The tool stores the submitted code for each user and automatically grades the submissions.

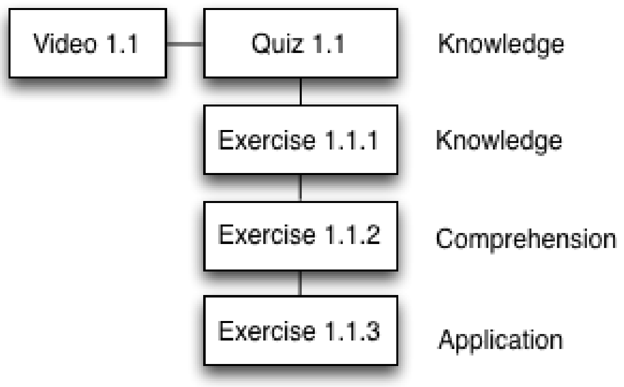


Fig. 1. Exemplary structure a *building block*. Each week contained four of these blocks following the same structure.

The remainder of the paper is structured as follows: The first section will give some information about the course itself and its participants. The second section will give more information about the programming tool, the experiences with it throughout the course, and how it has been connected to the openHPI platform. Furthermore, similar solutions offered on other MOOC platforms will be compared to our approach. The third section will deal with a non-technology-related scaling issue that has been encountered while the course was running: The participation in the forums and the usage of the help-desk, which was noticeably higher than in regular openHPI courses. The final sections will discuss some of the steps we are going to take next and conclude our findings.

II. Course Structure

To scale the HPI's activities in the area of youth development training, a prototypical MOOC to teach children and adolescents the basics of Python programming has been designed throughout summer 2014 and was realized on openHPI in October 2014. The 4-week course consisted of teaching videos, self-tests in quiz-format, and graded hands-on programming exercises. For each week, four videos were offered. Linked to each video were one ungraded self-test and three graded exercises in different levels of difficulty. According to Bloom's

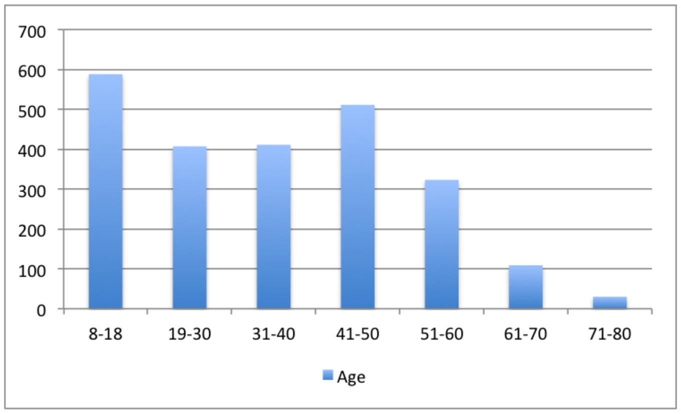


Fig. 2. Total user distribution (8-80).

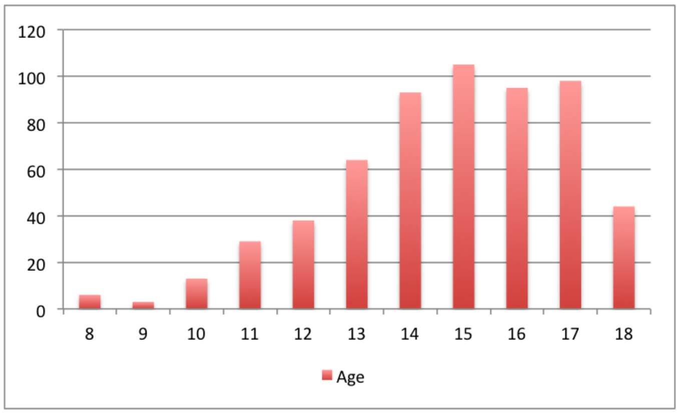


Fig. 3. User distribution of the actual target group (8-18).

taxonomy [2] the exercises can be classified in the categories: knowledge, comprehension, and application. Figure 1 shows such a building block—exemplary for the first video in week one. All others followed the same structure. The deadline for the graded exercises of each week was at 10:00pm(CEST) on the following week's Monday.

The course had about 7400 enrolled participants. The specified target group were children from the age of 11 to 17, but the course attracted users from a wider range of ages. Some parents enrolled in the course together with their children, also teachers enrolled with their classes. Additionally, the course attracted a vast variety of programming beginners of all ages. Many of them have expressed in the forum, that the promise of understanding basic computer programming, persuaded them to join this course. Figures 2 and 3 show the distribution of the participants age, as far as it is known. The participants are not obliged to enter their age—in the case of Python2014 we have this information for 2379 participants (32%). This course provided the users with a tool to write, run, and evaluate programs in the browser. From here on we will refer to this tool as WebPython. It was very well accepted amongst the participants. Table I shows some of the numbers in the context of user activity with regard to this tool. In the context of our regular courses, active participants are defined as those users, who have submitted at least one discussion post or one assignment [3]. In previous courses the percentage of active participants during the first week was about 40% (see e.g. [4]). In Python2014, due to the different setting, we defined an active participant as a user that at least has started one exercise in the respective week. According to this definition, Python2014 sported 68.73% of active users during the first week (derived from the number of registered users (Table III) and active users (Table I).)

	Users	Started	Submitted
Week 1	4961	51209	48764
Week 2	3696	37597	34115
Week 3	3078	28862	26118
Week 4	2392	23780	21725
RoA	2523	-	-

TABLE I. Active users, started exercises, and exercises with submitted (not necessarily correct) solutions per week. The last row shows the number of issued Records of Achievement.

	Total	**Active**
Average openHPI courses	18.3%	51.11%
Python2014	34.22%	50.86%

TABLE II. COMPLETION RATES IN RELATION TO THE TOTAL OF REGISTERED USERS AND THE NUMBER OF ACTIVE PARTICIPANTS DURING WEEK ONE

Figure 4 shows the decrease of the number of active participants over time. High dropout rates are a common issue that is shared by most MOOCs. Courses on openHPI have generally rather low dropout rates compared to other MOOCs [3]. To compare Python2014 to other courses on openHPI, we employed the same methodology as Meinel et al [3]. They observed the participants' engagement throughout the course and took the number of submitted homework assignments as measured value. The submission number for the first week's homework is taken as a reference (100%) [3]. In this course multiple submissions per participant and exercise were possible. We, therefore, normalized the number of submissions by counting only one submission per participant. We calculated the average value of the courses Meinel et al have listed and compared this average to the number of active participants in Python2014 based on their methodology. The dropout rates of Python2014 seem to be a little higher than in the average openHPI courses especially in week four (see Figure 5).

Comparing the completion rates of the average, regular openHPI course and Python2014 shows a different image: The completion rate in relation to those users that have been active in week one is almost the same as for the average openHPI course. The completion rate in relation to the total of registered users is even better (see Table II.) At the end of the course we also conducted a survey amongst the participants. The results of this survey indicate two explanations for this phenomenon:

1) A substantial amount of participants took the course with their school class, some of them not by choice. Teachers required a Record of Achievement to pass the class. So, as soon as those students had achieved 50% of the points, they quit.
2) The workload of the course was perceived as getting increasingly higher from week to week. More details will be discussed in one of the following sections.

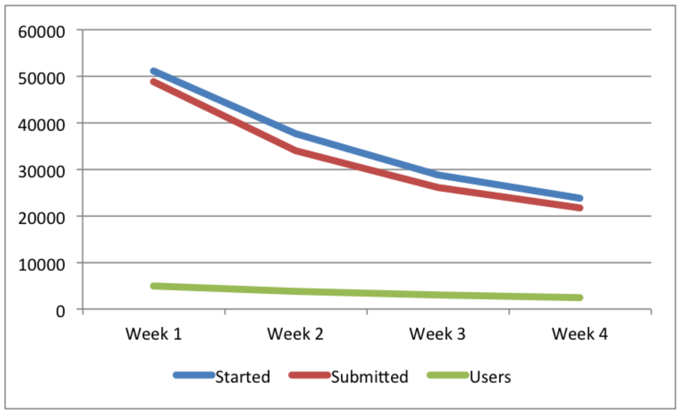


Fig. 4. Decrease of active participants from week one to week four.

III. SCALING THE TECHNOLOGY

As participants kept dropping in during week one, we decided to extend the deadline for the exercises of this week until the end of week two. During week two, we faced increasing load problems, culminating into a tightened version of another common problem that we are facing at openHPI: load peaks before deadlines. In this case the participants had to hand in the exercises for two weeks instead of one, additionally they did not have to hand-in simple multiple choice tests but running code. Particularly, in week two—the *loop* week—this caused technical problems, which forced us to extend the deadline for the first two weeks for another day until Tuesday. Figure 6 shows the load peaks on that particular Tuesday. At some point we had a CPU outage but could take measures to recover before the end of the deadline.

Figure 7 compares the number of started exercises, submitted solutions, and active users before and after the deadline of week three. The numbers on the left have been extracted a day before the exercises' deadline. The exercises were available

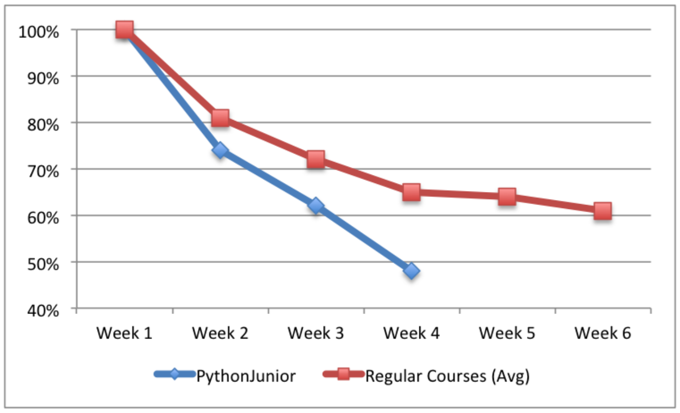


Fig. 5. Active participants in Python2014 compared to average openHPI courses.

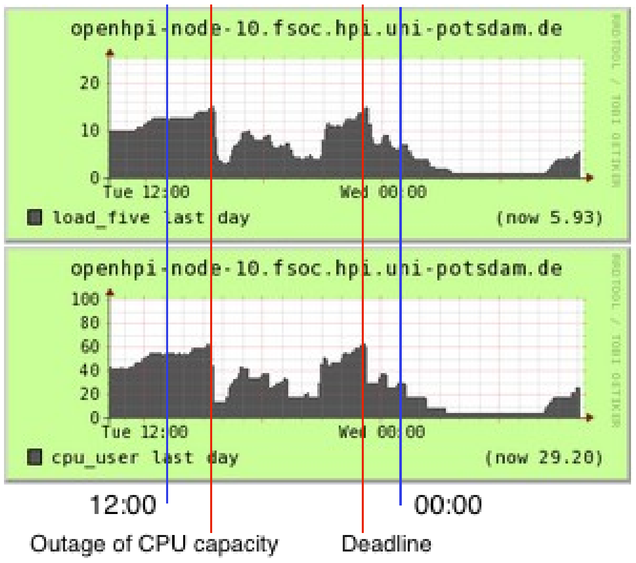


Fig. 6. CPU usage and load on a node shortly on the day of the deadline.

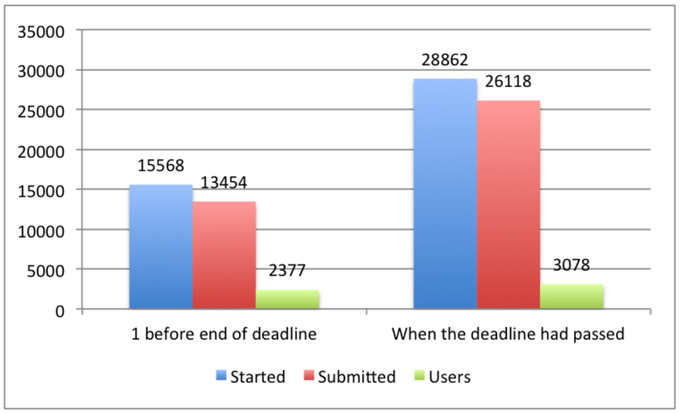


Fig. 7. Submission peak before deadline (week three, Python2014).

for exactly one week. During the remaining last day before the exercise's deadline the number of submissions almost doubled, which increased the load on the system remarkably. Unfortunately, we do not have these numbers for the first two weeks, but the trend would undoubtedly be the same there.

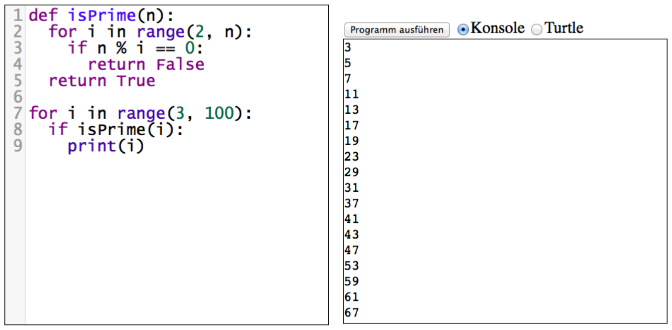
A. *Python Programming Tool*

The course objective was to teach programming with Python to people with no prior programming experience. As programming is learned primarily by exercise, we set a list of requirements for the infrastructure:

- There should be practical exercises along with the video teaching material.
- Participants should not be required to install additional software on their systems, to avoid hassle with broken software installations.
- There should be support for graphical output, following the turtle graphics programming model.
- There should be fully automated assessment of the programming exercises.

The second requirement resulted in the necessity to offer Python programming in the web browser. Our system features a web server that allows to edit Python source code in a programming editor, based on the CodeMirror[1] package. In

[1]CodeMirror: http://codemirror.net

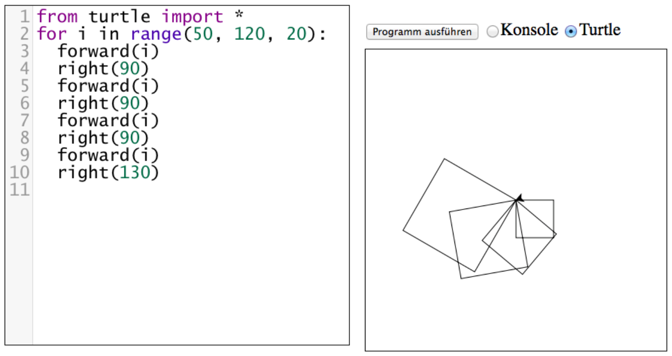


```
def isPrime(n):
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

for i in range(3, 100):
  if isPrime(i):
    print(i)
```

```
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
```

Fig. 8. WebPython in console mode.



```
from turtle import *
for i in range(50, 120, 20):
  forward(i)
  right(90)
  forward(i)
  right(90)
  forward(i)
  right(90)
  forward(i)
  right(130)
```

Fig. 9. WebPython in turtle mode.

order to test their programs, users can submit their source code to the server for execution and view the results in a console output window (see Figure 8). After completing the exercise, users can submit the code for assessment, resulting in a grade for the exercise. To allow for long-running Python processes that produce output in an ongoing manner, while also consuming user input of their own, a web socket connection is established between the web browser and the Python process.

As the course title and the target group suggest, we intend to attenuate prejudices against computer science amongst children. Therefore, we designed exercises with visual—and not only numerical or string—output. Turtle graphics [5] is a programming model developed by Seymour Papert in the 1970s, which targets this goal. Various projects have demonstrated its practicability in teaching programming to children (see e.g. [6] and [7]). Turtle graphics is a core part of the Python standard library and Python programming literature for children typically leverages this support [8] [9]. An exemplary output of a turtle script is shown in Figure 9.

Automated assessment of programming exercises is the most challenging aspect of this project, and previous work has demonstrated inherent limitations of automated assessment when compared to a human teacher's assessment [10]. On the other hand, automated assessment also has advantages over human assessment, such as instantaneous feedback, scalability to a large number of participants, and objectiveness.

We decided to allow participants to submit each exercise as often as they want, grading each attempt, and finally taking the best result into account. The assessment code is written in Python as well, running it, and then evaluating the output (not just the printed output, but also side effects on variables, function definitions, and other changes that were expected from running the program). In addition, for a few exercises, the source code was analyzed, primarily to determine whether the solution was using an undesired shortcut.

Running the Python script either for the user's testing purposes or for assessment requires a working Python implementation. For that purpose, several implementations of Python that allow to run code in the web browser itself are available. Unfortunately, none of them is production-ready in the sense that it provides sufficient compatibility with the *official* Python (aka CPython) installation. Therefore, we opted to provide a server-side installation of CPython.

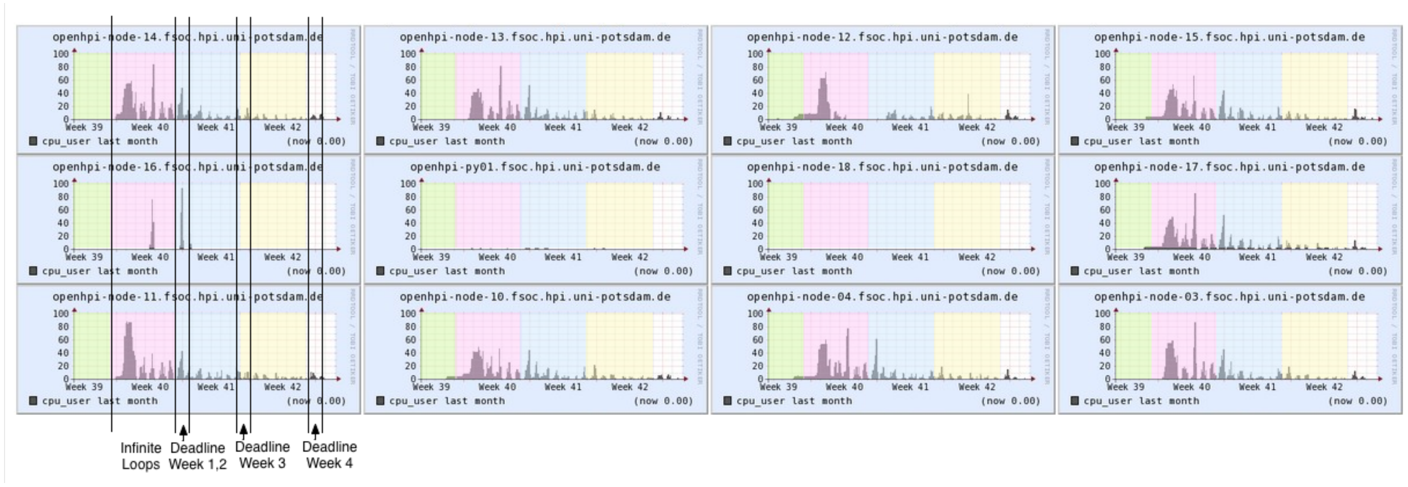


Fig. 10. Load on some of the nodes in our private cloud at the Future SOC Lab during the course. There are two nodes in the middle of the figure that have no or only very little load. The one on the left is the head-node that only runs the web-server. The one on the right did not want to join for unknown reasons.

B. *FutureSOC*

Providing a server-side CPython installation produces two challenges: first, it is necessary to provide sufficient CPU and memory resources to accommodate a large number of concurrent users. Second, the system must be protected against abuse and user mistakes.

To allow for a large number of participants, we set up a private cloud in the HPI's Future SOC Lab[2].

The Future SOC Lab is a cooperation of the HPI and a couple of industrial partners. It provides a complete infrastructure of state of the art hard- and software to researchers—free of charge—for a certain period of time. Python2014 was running in between two lab periods, and therefore, we had the chance to use some of its computing power. The code submission system consists of several server processes as shown in Figure 11. The head node runs the nginx web server, to provide for static files and TLS encryption. It delegates to a number of Python servers written with the Twisted framework[3] which perform user authentication, templating of the exercise descriptions, and delegation to worker processes running on the cluster. Each node in the cluster runs a Twisted Python server, which in turn launches a Docker[4] container that ultimately runs the Python process.

[2]http://hpi.de/en/research/future-soc-lab.html
[3]https://twistedmatrix.com/trac/
[4]https://www.docker.com

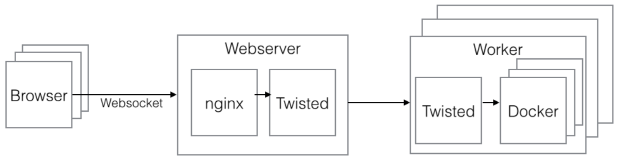


Fig. 11. WebPython's architecture.

The cloud consists of 18 machines in the HPI FutureSOC Lab infrastructure. Each machine has 24 CPU cores and 64GiB of memory.

As kind of a coarse-grained load balancing, we initially had set a limit of 10 processes to be run per core. This arbitrary limit caused a shortage on available process slots within week two, where we dealt with the concept of loops. As expected, several users created infinite loops by error, which blocked a process slot for the maximum allowed runtime of 10 minutes per process. This issue could luckily be solved by setting the limit to 50 processes per core, resulting in a more efficient scheduling, capable of enduring some infinite loops without running out of available process slots. We have not gotten any CPU or main memory outage since this increase. This overload situation can be seen in Figure 10 as the block of higher CPU usage in week two and the pike at the beginning of week three. Afterwards, only minor peaks show up—also due to decreasing numbers of active users—but no more longer lasting high load situations.

As expected and clearly visible in the peaks, most load always occurred close to the closing dates of the respective exercises. With regard to the different charts, it stands out that some nodes hardly show any load at all. Node 18 seemed to be simply broken, while node *py01* was responsible for rendering the web front-end of the python editor, which was a comparably simple task requiring only little CPU capacity.

The protection against abuse and mistakes is primarily achieved by using Linux containers, and the Docker infrastructure that builds on it. Users are isolated against each other and the system, and state is not persisted across program invocations. Docker also allows to limit the CPU share and memory that a process may consume. However, it currently does not provide sufficient support for setting a maximum process run time, which becomes necessary as programming beginners will inevitably write programs that perform endless loops. External monitoring is used to kill processes that exceed the maximum acceptable run time of 10 minutes.

IV. Comparison to Similar Courses on Other Platforms

In this chapter, we provide a short description of similar courses. The selection is based on the following criteria: First, we only consider courses on MOOC platforms as most of the issues we are focussing on in this paper are to some extent specific for this format. Second, we focus on Python programming courses and third we focus on courses that claim to be addressing programming novices. Finally, we do not claim that the following list is exhaustive.

Programming for Everybody[5]—A first programming course, teaching Python, applies an open-source, web-based development environment[6] for writing and assessing practical programming exercises. The tool is based on CodeMirror and Skulpt, an in-browser implementation of Python, providing client-side code execution. Since no request to the server is required, the tool's client-side code evaluation approach has the advantage of short response times. Infinite loops also turned out to be problematic, as they eventually rendered the browser window unresponsive. Affecting the client side only, at least they did not affect the performance of the tool for other users than those that caused the problem. Program errors are reported using native browser alert dialogs. Error messages are reduced to the essential. They neither contain a traceback nor provide additional clues to the error's origin. Whenever code is executed, it is also checked against the exercise specification. The programming tool performs automatic grading, based on I/O matching and basic invocation checks at runtime. The grading approach can only grant full score or zero points. Partial solutions, however, are not awarded. Besides auto-graded programming assignments, the course contains two optional peer-graded essays.

Intro to Computer Science[7]—An introductory Python course aiming to build a basic search engine. The course makes use of a lightweight web-based development tool, which is based on CodeMirror and seamlessly integrates into the Udacity platform. The editor is easy to operate, but it provides no means for editing more than one unit of code. Exercise instructions are provided as comments in the skeleton source code and sometimes in the form of a short introductory video. Learners can restart the exercise from scratch, execute their code for exploration, submit their code for evaluation, or view a sample solution, which is presented in a step-by-step fashion. During program execution, Python's standard traceback is presented in case of an occurring error, which might be too cryptic for beginners. During test execution, basic hints pointing to corrective actions are provided, however. The result of successful test-based code assessment is briefly presented in natural language, which benefits comprehensibility but lacks valuable details, such as expected and actual program behavior.

V. Beta-Testing the Course Design for Learning Environments at Scale

As a course of this kind was new terrain for all members of the teaching team, we conducted a beta test with six high-school students, who participated at one of the HPI's on-site

[5]https://www.coursera.org/course/pythonlearn

[6]https://github.com/csev/tsugi

[7]https://www.udacity.com/course/cs101

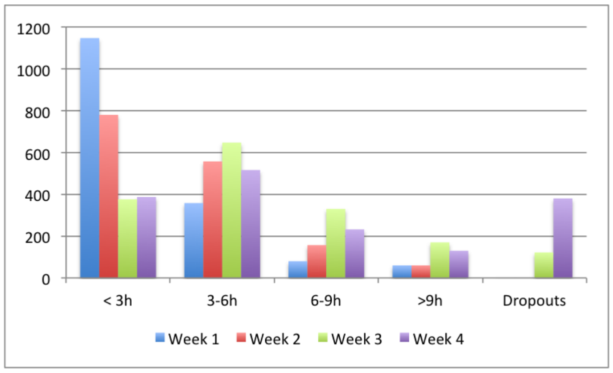


Fig. 12. Workload as perceived by the participants.

youth development activities. The main reason for this effort was to get a feeling about the necessary amount of time that participants would require to cover the workload. The pupils in the beta test were able to work through the exercises of week one in very short time. It took the fastest pupil about ten minutes the slowest had to invest about twenty minutes. Another test was conducted with first semester HPI students. While these students did not notice a higher workload for the following weeks, a survey at the end of the course, and some discussions in the forum, showed that the participants perceived this differently. While in weeks one and two the majority of the users stated that the workload was below three hours, in weeks three and four this shifted towards three to six or even six to nine hours with an increasing dropout rate. See also Figure 12. Therefore, we investigated if our analysis tool[8] would also reveal that the time that the users have spent on openHPI has increased, which was not the case. See Figure 13. Instead, we can see again the recurring pattern of activity peaks shortly before approaching deadlines.

VI. Scaling the Support

At openHPI, the user forum is an important part of scaling the support for the users. The help-desk, openHPI's one-to-one support tool, turns out to be overwhelming the teaching teams, particularly in hands-on courses on beginner's level. Many participants do not realize that personal support is not manageable for a small team facing the massive amount of users. Too often requests cannot be answered during the runtime of a course. Monitoring the forum alone and providing appropriate support is often already taking the teaching teams to their limits. In general, our teaching teams consist of the professor, four to eight research assistants, and one or two

[8]Piwik: http://piwik.org/

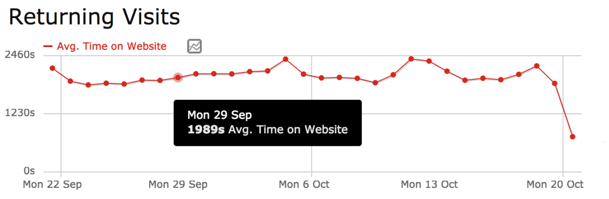


Fig. 13. Average session length during the course.

No. of	Python2014	IMDB2014
Users total	7373	8641
Users active (avg)	about 3700	1191
Users(Q)	700	139
Users(A)	670	56
Users(C)	770	104
Questions	1404 (0.19/2.01)	219 (0.03/1.57)
Answers	2178 (0.30/3.25)	203 (0.02/3.63)
Comments	3571 (0.48/4.64)	385 (0.04/3.70)

TABLE III. COMPARISON OF FORUM AND HELP-DESK USAGE BETWEEN PYTHON2014 AND IN-MEMORY DATA-MANAGEMENT (IMDB2014)—TWO PARALLEL COURSES ON OPENHPI. USERS(Q,A,C): AMOUNT OF USERS THAT POSTED A QUESTION, ANSWER, OR COMMENT. THE NUMBER IN BRACKETS SHOW THE AVERAGE AMOUNT OF POSTS PER USERS TOTAL/POSTING USERS.

students. Participants are encouraged to support each other in the forums. The teaching team only intervenes if discussions get rough, or go down the wrong track. In Python2014 it could be observed that this style of offering support to the course participants has worked very well. Naturally, as the course had way more participants than the on-site youth development activities at HPI, an increased demand for support had already been expected. The factor, however, by which the forum usage of Python2014 has increased in comparison to other openHPI courses really came surprising. Table III compares the forum activity of Python2014 to a IMDB2014, a regular openHPI course that was conducted in parallel. The numbers in brackets show the average amount of forum posts per total participants and the average amount of forum posts per users that actually have posted a question, answer, or comment in the forum.

The forum usage in Python2014 was significantly higher compared to other openHPI courses (exemplary IMDB2014). The average number of questions per participant was six times higher in Python2014, the average amount of answers was even more than fifteen times as much. The amount of comments was about twelve times higher in Python2014. The majority of the questions were about problems with the programming exercises. Often, the participants started with a complaint about the automatic grader while asking for help. Having received an answer enabled them to finally solve their problem. Common complaints were about tests being too restrictive, e.g. being case sensitive.

We came up with a couple of hypotheses for this significant difference between the two courses.

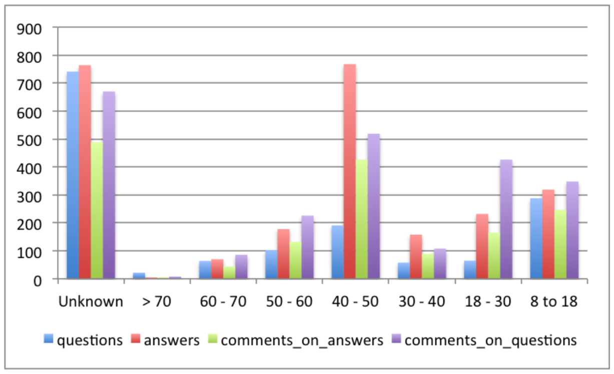


Fig. 14. Age of the participants who were active in the forum

1) The age of the participants: Python2014 sported a high percentage of very young participants. Younger users are more used to expose themselves on the internet and therefore have less problems to publicly post in a forum.
2) More posts due to ”misusage” of the forum, such as duplicated questions due to a ”post first, search for similar topics later (or never)” tactic by the less experienced participants, fun posts or irrelevant posts, and user to user communication via the forums.
3) Optimized forum usage by the more experienced users of IMDB2014.
4) The professional level and background in IT of the participants: IMDB2014 participants are at a higher level of their career and, therefore, have the notion that they have more to lose when they show the gaps in their knowledge by posting.
5) The language barrier: Python2014 participants were mostly German native speakers, while IMDB2014 participants come from all over the world.
6) The possible benefit that could be achieved by actively taking part in the forums was much higher in Python2014 as graded homework could be done over and over again. Asking for help in the forum was likely to improve the grade. Due to the short and strict time limits for graded assignments in IMDB2014, in this course, it was rather unlikely to find direct help in the forum.
7) As the course itself afforded more active participation, the threshold to actively participate in the forums was less high.
8) The forum was identified as the second most important learning material, next to the course videos, in the survey at the end of the course. Way in front of books, even relegating the internet to the third position.

We tried to find indicators for these hypotheses in our data. Table IV gives an overview about the socio-demographic

	Python2014	IMDB2014
Background IT		
Beginner	43.8%	14.7%
Advanced	58.7%	49.8%
Expert	18.7%	35.5%
Professional life		
None / not set	81.6 %	61.6%
Up to 5 years	4.7%	10.4%
Up to 10 years	2,9%	8.5%
More than 10 years	10,9%	19.5%
Highest degree		
High school	23.4%	10.4%
Bachelor, Diplom, Master	42.7%	76.0%
PhD	3.7%	4.9%
other	30.1%	8.7%
Gender		
Male	77.2%	87.5%
Female	22.8%	12.5%
Country		
Native German	96.2%	48.8%
Native English	0.8%	14.2%
India	0.3%	14.9%
Other	2.7%	22.1%

TABLE IV. SOCIODEMOGRAPHIC COURSE DATA.

data of the course participants in comparison to a regular course on openHPI. As mentioned before in the context of the participants' age distribution, we only have this data available for those participants who voluntarily entered this information in their profile page (25.6% in Python2014 and 41% in IMDB2014). Generally, those users, which have a longer relation to the platform and have taken more than one course (and, therefore, are not in the actual target group of this course) are more likely to have completed their profile. So the actual circumstances might be a little distorted. Figure 14 shows the age distribution of the active participants in the forum. Most of the posts are from users that have not completed their profile information. Where we have this information, it seems to be the 40 - 50 year olds who were most active. There were some patterns that we observed amongst the younger participants. For instance, we can say that some inappropriate posts that had to be removed, were definitely written by younger participants. After deleting the threads we wrote to the culprits in private and asked them to stop such behavior. They apologized and the number of this kind of posts was reduced remarkably. Technical or performance problems at peak hours generated a lot of forum posts. We could observe, that especially among younger and not as experienced online-learners, the forum mirrored frustrations over-exceptionally. Discussions in IMDB2014 mainly revolved around three topics: general questions and comments concerning the content, seeking for clarification if the participants understood it correct, in-depth questions on applying the presented principles on real-world problems and organizational questions concerning the course. Since technical issues were mainly targeted towards the help-desk, and organizational questions accounted for only the minority of threads, most discussions were content oriented. The fact that the in-depth questions usually were too specific to be answered by fellow students, such threads typically ended up in a 3-5 post status, with one post asking the question, a teaching team member answering it and the original poster either saying thank you or asking for further clarification of a certain detail and then thanking the team member. Comparing the numbers per active or posting user, the gap becomes noticeable smaller. While the participants of IMDB2014 in average were less likely to be active in the forum, those who did participate did not show a different communication pattern than the participants of Python2014.

A potential reason for the, generally, more reluctant behavior within IMDB2014 is the fact that substantial parts of the audience did not have English as their mother tongue and, therefore, felt uneasy to pose questions potentially exposing them in front of other adults or even professionals and colleagues. Matching our hypotheses to the data presented in Table IV, it becomes apparent that the language hurdle is most likely to be the main reason for the lower post count within IMDB2014. Above 85% of the participants were non native speakers with regard to the course language. Also the second hypothesis, taking the professional career into account, is backed by the data. While the majority of users did not answer this question at all, the fraction of participants with higher progress in their career in IMDB2014 is about twice as high as the corresponding one in Python2014.

VII. Future Work

Coming up next on openHPI is another hands-on programming tool that will allow to set up courses in a wider variety of programming languages. This tool also allows to define test cases for these languages—in the language's native testing framework, e.g., RSpec for Ruby, JUnit for Java, etc. Courses, such as Python2014 require additional forum support and monitoring. A possible solution might be to involve experienced and active participants. In an earlier survey we had very encouraging results from users that would be willing to mentor in future iterations of a course [11]. Making these people recognizable as mentors could ease the high demand for individual support. Already during the registration phase of Python2014 an opportunity to scale the support for participants in future iterations of this kind of courses evolved. We received many requests from teachers who intended to use the course in a flipped classroom setting and asked if the platform supports mechanisms to enroll whole classes or to monitor the results of the students in their class. Enabling teachers to give optimal support for their pupils seems to be a promising way to relieve openHPI's teaching teams from at least a part of their workload. For this use case enabling teachers to create learning groups, supervise their students within the group, and being able to check their results will be very helpful for both sides. A very important step is to enable users to go on from where they are when the course has ended. Particularly, it is important to enable the users to step forward from the restricted learning environment that has been employed during the course to a more self-determined way of coding. Therefore, we plan to enhance our next programming course with a supplementary course introducing the participants to the world of compilers, interpreters, and IDEs.

VIII. Conclusion

The MOOC format certainly offers a valuable means to scale the youth development efforts of an institution such as the HPI. Offering MOOCs for a very young audience—or more general for novices—however comes along with a couple of challenges. Particularly, a significantly increased amount of support needs to be taken into account when planning the human resources for such courses. Despite several technical hiccups at the beginning of the course, we conclude that the course can be considered as successful. It was well received by the participants and had a comparably very high rate of user participation. In the survey at the end of the course, 84.33% of the users stated that they would recommend the course to other people, and 71.10% asked for a sequel of the course. The age distribution turned out to be more widespread than we had intended. Concerns that this could turn out to be problematic did not come true, rather quite the opposite was observed. Participants of all ages worked very well together and supported each other. Particularly, programming courses at a beginner's level should support a tool that liberates the participants from having to install a programming environment of their own to ease the initial pain. Towards the end of the course, the participants need to be introduced to more powerful tools, however, to enable them to transfer the knowledge they gained to real life challenges.

IX. Acknowledgments

We thank Kai Fabian, Nicco Kunzmann, and Hauke Klement for their help with the help-desk, forum, and platform. We additionally thank Bernhard Rabe for supporting us at the Future SOC Lab and the pupils and students who helped to beta test the course.

References

[1] C. Meinel and C. Willems, *openHPI : the MOOC offer at Hasso Plattner Institute, Hasso-Plattner-Institut für Softwaresystemtechnik: Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam ; Nr. 80*. Potsdam, Germany: University Press, 2013.

[2] B. S. Bloom, "Taxonomy of educational objectives," *Cognitive Domain*, vol. 1, 1956.

[3] C. Willems, J. Renz, T. Staubitz, and C. Meinel, "Reflections on enrollment numbers and success rates at the openhpi mooc platform," in *Proceedings of the European MOOC Stakeholder Summit 2014*, no. EPFL-CONF-196608. PAU Education, 2014, pp. 101–106.

[4] T. Staubitz, J. Renz, C. Willems, J. Jasper, and C. Meinel, "Lightweight ad hoc assessment of practical programming skills at scale," in *Proc. CHI 2014*, no. 10.1109/EDUCON.2014.6826135. IEEE, 2014, pp. 475–483.

[5] S. Papert and C. Solomon, "Twenty things to do with a computer," *MIT Artificial Intelligence Memo*, no. 248, 1971.

[6] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.

[7] C. Emihovicha and G. E. Miller, "Talking to the turtle: A discourse analysis of logo instruction," *Discourse Processes*, vol. 11, no. 2, pp. 183–201, 1988.

[8] J. R. Briggs, *Python for Kids*. San Francisco: No Starch Press, 2012.

[9] G. Lingl, *Python für Kids*, 4th ed. Heidelberg: bhv, 2010.

[10] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, 2005.

[11] T. Staubitz, J. Renz, C. Willems, and C. Meinel, "Supporting Social Interaction and Collaboration on an xMOOC Platform," in *EDULEARN14 Proceedings*. IATED, 2014, pp. 6667–6677.