Math.Comput.Sci. (2023) 17:15
https://doi.org/10.1007/s11786-023-00560-z

Mathematics in Computer Science

# Self-Paced, Instructor-Assisted Approach to Teaching Python Programming

**Pavel Solin · Alexander Freyer**

Received: 28 September 2022 / Revised: 9 January 2023 / Accepted: 23 January 2023


**Abstract** We present a novel self-paced, instructor-assisted approach to teaching Python programming in a college classroom environment. Instead of listening to a lecture and doing homework on their own later, students work actively 100% of the time, using an advanced interactive learning platform combined with real-time individual assistance of their instructor. Instead of lecturing, the instructor walks through the classroom and assists students individually. This makes a huge positive difference for the students, and it also allows the instructor to understand much better how each student performs and where they need help. After numerous years of using this approach, we are convinced that nothing can replace a one-to-one interaction between the student and his/her instructor. Unfortunately, traditional classroom lectures are not consistent with this teaching style. On the contrary, the mostly uni-directional flow of information from the instructor to the class effectively shields the instructor from interacting with the students. We describe various aspects of the self-paced, instructor-assisted method and show that it has a major positive impact on the students. Students find it extremely helpful being able to digest the material at their own pace. They also get a lot more practice compared to the standard lecture + homework model, develop good programming habits, and become skilled and experienced programmers. This method makes students better independent learners, problem solvers, and communicators. Our findings are based on 10+ years of teaching Python programming courses to diverse audiences in this way. Results of a sample student survey and student testimonials are presented.

**Keywords** Computer programming education · Python · Competency-based education · Self-paced course · Learning by doing · NCLab

**Mathematics Subject Classification** Primary 97U50; Secondary 97B40 · 97D40

Pavel Solin is the founder of NCLab (http://nclab.com).

P. Solin (✉)
Department of Mathematics and Statistics, University of Nevada, Reno, USA
e-mail: pavel@nclab.com; solin@unr.edu
URL: http://nclab.com

A. Freyer
University of Lübeck, Lübeck, Germany
e-mail: alexanderfreyer98@gmail.com

Published online: 20 July 2023

Birkhäuser

## 1 Introduction

In the era of Industry 4.0, automation, and big data, computer programming is an essential career skill. The findings of this study are applicable to teaching computer programming in general, but we focus specifically on Python which crosses the borders of traditional Computer Science and reaches into many other areas including Data Analytics and Data Science, Artificial Intelligence, Machine Learning, Robotics, Advanced Manufacturing, Engineering, Sciences, Business, Finance, Healthcare, to mention just a few.

Computer programming currently is among the most sought-after job skills in the U.S. job market, given the shortage of qualified applicants to fill available Computer Science-related jobs. The U.S. Bureau of Labor Statistics (BLS) projects that computer and IT-related jobs will grow 11% between 2019 and 2029, which is much faster than national job growth overall. According to a recent Gallup poll [19], 63% of White students, 60% of Black students, 61% of Hispanic students, and 71% of Asian students are interested in pursuing a career in Computer Science.

However, computer programming education still has significant issues which are heavily discussed in the literature (see, e.g., [3,6,13,15,16,18,20,24,26,27,30,33,37,38,40] and the references therein). Research shows that failure and dropout rates are high in both introductory and advanced computer programming courses [24,40], as well as that many students who finish the courses successfully do not possess sufficient and expected level of programming skill [13]. These problems are usually attributed to:

1. Abstract nature of computer programming courses [18].
2. Challenges related to combining syntax and logic [33].
3. Lack of practice [16].
4. Lack of motivation of students to learn computer programming [3].

To combat problems 1 and 2, a number of simplified visual educational programming languages have been created such as Logo [11], Karel the Robot [25,36] or Scratch [32], but research shows that programming novices have problems with learning even those [18]. Some authors go as far as to conclude that no programming language is suitable, and cannot be suitable, for novice programmers [33].

On a more positive note, Tan et al. [38] shows that practice is a preferred way of learning computer programming for many students and that it increases student success. Kinnunen and Malmi [17] finds that motivation is one of the most important factors in student success. Ben-Ari [4] proposes that constructivism should be used when designing computer programming courses, meaning that the learner should be taken on an active path where s/he is deeply involved in the learning process and s/he also builds new knowledge on top of existing knowledge. Fincher et al. [12] argues that computer programming courses should be designed to accommodate individual needs and learning preferences of students.

We also agree with [38] that practice is key when learning computer programming, with [17] that motivating students is crucial, with [4] that constructivism should be used when designing computer programming courses, and with [12] that individual needs and learning preferences of students must be taken into account. We also agree with [18] that, unfortunately, traditional ways of teaching computer programming do not work well.

Our findings presented in this paper are very much consistent with the last two paragraphs. Nothing can replace a one-to-one interaction between the student and his/her instructor who can not only help with technical questions but also do a much better job understanding individual needs of the students, and motivating them. This is practically impossible in the traditional classroom setting, but it can be achieved with the help of an advanced interactive learning platform such as NCLab [23].

## 2 Outline

Section 3 explains why neither traditional lectures nor the flipped classroom model are suitable for teaching computer programming. Section 4 presents a self-paced approach where students learn by doing, at their own pace, getting significantly more one-to-one interaction with their instructor compared to traditional lectures.

A prerequisite for this approach is the availability of an interactive learning platform which provides carefully designed short tutorials, examples, exercises, autograded practical tasks, quizzes, and discussions. This is discussed in Sect. 5. The software platform must be able to check the student's solutions and answers in real-time and provide instant feedback, help and guidance as explained in Sect. 6.

Section 7 describes another important component of the learning platform: a real-time student progress monitoring dashboard that is available to the instructor. Sections 8 and 9 present the results of a sample student survey and representative student testimonials.

In this study we use NCLab (Network Computing Laboratory) [23] but any other software platform with the above traits would work as well. In Appendix A the second author describes his personal experience with learning programming in a self-paced way, and compares it to the traditional instruction at his University. Appendix B briefly presents the self-paced Python curriculum in NCLab, and Appendix C explains in detail how Python exercises are graded on a remote cloud server.

## 3 Lectures, Homework, and Flipped Classroom

Lectures have been the traditional means of passing on knowledge from instructors to students since the founding of Western universities in the middle of the eleventh century. Much has changed since then though, in particular the access to information through the Internet and other media. Yet our educational system today still teaches students to sit, listen, take notes, and wait for somebody to tell them what they need to know. This does not make them independent learners and problem solvers though. Instead, students should become active learners who can find and filter new information. The instructor is still very much needed in this process, but s/he does not necessarily have to serve as the primary medium which delivers elementary information to students.

Speaking about computer programming in particular, this is a skill, and any skill requires many hours of hard work and practice [16]. However, sitting in a lecture for 50, 75 or even 90 min, listening to an instructor, and taking notes does not invite students to be active, and it definitely does not give them an opportunity to practice what they are learning in real time.

An overwhelming number of authors agree that student attention during lectures tends to wane after approximately 10–15 min (see [5,10,14,21,39] and the references therein). These exact numbers and their origin are sometimes questioned [7], but there is other heuristic evidence suggesting that people's attention span is limited. For instance, TED talks have an 18 min limit, based on the notion that 18 min is long enough to have a "serious" presentation but short enough to hold a person's attention.

A traditional lecture typically introduces several new concepts, but the students do not have a chance to practice and master one before being exposed to the next. Even worse, often the next concept builds upon the previous one, which resembles building a second floor of a house with the first floor not being there yet. Achieving mastery of the concepts, if at all, happens later at home, with no instructor to be there when the student actually needs help. This not only leads to frustration and the development of bad programming habits, but students often seek help online, where a number of commercial services such as [8,9] gladly "help them" for a small subscription fee, by facilitating access to homework solutions uploaded by their peers.

Another drawback of traditional lectures is uniform pacing. The instructor sets a pace which works best for most students, but for some students it will always be too fast and for some others too slow. This leads to the former group getting lost and the latter getting bored. The instructor must present a given amount of material in the given timeframe (typically there are about 20 lectures per semester), which leaves almost no time for spontaneous instructor-student interaction. In fact, the uni-directional flow of information from the instructor to the students actively shields the instructor from such interactions.

Let's conclude this Section with mentioning the flipped classroom [2,31]. Flipped classroom is structured around the idea that lecture or direct instruction is not the best use of class time. Instead, students learn on their own before class, freeing class time for activities that involve higher order thinking. However, by "learning on their own before class" most authors mean watching pre-recorded video lectures, which by itself is not the best way to learn computer

programming. Learning practical skills such as computer programming requires hands-on practice and problem solving. While learning, students very much benefit from real-time access to their instructor, which studying in advance at home does not provide. Also, in our experience, group activities such as class discussions are not the best for learning computer programming.

## 4 Self-Paced Learning by Doing

Frustrated by experiencing the problems described in the previous Section, around 2010 we started to teach computer programming using a self-paced approach where students learn at their own pace, by doing rather than listening to someone or watching videos. Not having to lecture frees the instructor's time which s/he can use to interact with students and help them individually. We found this active learning approach to be enormously successful with every audience including elementary school kids, middle and high school students, college students, and even working adults, both with and without prior programming experience.

In 2019 we started to use the self-paced approach to teaching college-level Linear Algebra which is a much more theoretical and abstract subject. To our surprise, the experiment was enormously successful again, with students achieving up to 20% better overall scores and consistently reporting that this is how they would like to learn in other math courses as well [35].

Carefully designed interactive course materials play a critical role in the self-paced, instructor-assisted learning model. Let's therefore discuss them next.

## 5 Interactive Course Materials

The interactive course materials are split into individual lessons, where each lesson corresponds to one traditional classroom lecture plus the corresponding homework. A lesson typically covers several new concepts. These concepts are clearly separated from each other, so that students only deal with one new concept at a time. Importantly, the software platform does not allow students to move on to the next concept until they have proved mastery of the current one. Each lesson includes the following components:

1. Short, bite-sized tutorials presenting one concept at a time.
2. Live examples for the students can tweak and experiment with.
3. Graded exercises whose score does not count towards the course score.
4. Graded tasks whose score counts towards the course score.
5. Both the graded exercises and tasks come with hints and discussions.
6. Side-by-side comparisons of the student's code and the master solution.
7. Quizzes which provide detailed explanations for incorrect answers.

### 5.1 Tutorials

Every lesson begins with a short, bite-sized tutorial which includes text, code, and/or illustrations. NCLab does not use video tutorials, because we believe that our brain is in a passive state when watching a video, and this is not consistent with our active approach to learning. Tutorials usually contain a review of previous topics, because we learned that students appreciate that very much. Who knows the material well can simply skip the review, but for other students it can be extremely helpful. This is completely different from a traditional classroom lecture where, when the instructor does a review, the entire class must listen to it.

### 5.2 Examples

In our tutorials we use examples which are "live" in the sense that students open them, run code, and see the results in real time. They are guided to tweak the code, try various things, until they become comfortable with the concept that the example demostrates. Examples are not autograded, meaning that there is no feedback telling the students whether or not what they did was correct. For this there are Exercises.

### 5.3 Exercises

Exercises are the next step after live examples. Students are given tasks which are designed for them to apply the new concept they just learned. Exercises are autograded, meaning that the students see instantly if their solution is correct or not. If their solution is incorrect, then they are helped to identify and fix the problem. Exercises do not contribute to the overall course score.

When writing a program, students can run it as many times as they need without penalties. The grading only occurs when they decide that their program s final, and submit it for grading. If the grader discovers problems with the solution, the students are given feedback, and asked to correct their work. If the program passes the grader, the student is allowed to proceed.

### 5.4 Tasks

When students successfully solve the exercises, and build enough confidence in their understanding of the new concept, they proceed to the next task. Tasks are similar to exercises, they are also graded by the software, but the score counts towards the students' total course score. Sometimes multiple tasks are queued before a new concept is introduced.

### 5.5 Hints

Every task comes with a hint which students can use if they are not sure how to solve the task. Using a hint costs them 3% of the score for the task. If the hint is not sufficient, and if the instructor is not around at the moment (for instance when the student works in the evening at home), then students can use a Fast Forward (FF) button to proceed. The instructor sees the usage of hints and the FF button, and can intervene if there is too much of it. We will talk about student progress monitoring in more detail in Sect. 7.

### 5.6 Discussions and Side-by-Side Code Comparisons

After completing each practical task, students see a side-by-side comparison of their code with the master solution. This allows them to get used to correct code formatting, which is something that they often are not able to develop on their own. If their solution differs from the master solution, usually it is because their solution is sub-optimal. If there is more than one way to solve the task, there is a post-solution discussion where various advantages and disadvantages of the different solutions are compared. Students consistently report that they find the side-by-side code comparisons and the discussions extremely helpful.

### 5.7 Quizzes

Each lesson includes one of more quizzes whose role is to ensure that students read the tutorials carefully, and learn the underlying theoretical concepts along with the practical skills. The quizzes include free answer questions,

multiple choice questions (where only one option is correct), checkboxes (where none, one, or multiple options may be correct), code cells where students are asked to complete missing code segments, and more. A detailed description of quizzes is given in our Linear Algebra paper [35].

Quizzes can be retaken after 12 h, and the better score counts. The purpose of this feature is to motivate students to return to the material, and learn what they missed. During the retake, students are presented with similar but not the same questions.

## 6 Importance of Real-Time Feedback

The importance of feedback in the learning process cannot be stressed enough. Students do not learn when sitting in a lecture and listening to an instructor. They do not learn when watching a video or reading a tutorial. While doing this, they are merely just trying to store as much of the new information as possible, hoping to somehow make sense of it later. Even when solving a task, they do not learn until they receive a confirmation that their solution was correct.

THE "magical moment" occurs when they receive feedback. In the best case scenario this feedback comes from another human being directly - the instructor - but this is extremely rare in a college-level classroom environment. The validation also can come from an instructor in the form of a graded homework, but this typically takes too much time. A software platform on the other hand can provide an instant feedback which either tells the student that their solution is incorrect and why, or that their solution is correct. Only when they succeed and have their success validated, the students gain the confidence they need for the new concept to fall in place.

## 7 Real-Time Progress Monitoring

In order to keep track of the students' progress, the instructor must have access to a dashboard which displays student progress data in real time. In NCLab, the instructor dashboard provides three levels of detail:

- Class level: This is an overview of the entire class which shows the progress of all students visually, along with other useful information such as when each student was last active.
- Student level: When clicking/tapping on a particular student, the instructor can view all graded tasks and quizzes solved by the student including where they used hints, the Fast Forward button, etc.
- Task level: When clicking/tapping on a particular graded task of quiz, the instructor can see how much time the student spent solving the task, how many attempts they needed, and all versions of their code (whenever the student runs their code, a copy is stored in the instructor's dashboard). If the student has retaken a quiz, the instructor can see where they made mistakes in each submission.

## 8 Results of a Student Survey

Below we present a sample survey of 27 students from the Spring 2021 semester.
1. What is your overall level of satisfaction with the course?

| Satisfaction level | Number of students |
|---|---|
| Very satisfied | 22 |
| More than satisfied | 5 |
| Satisfied | 0 |
| Somewhat satisfied | 0 |
| Not satisfied | 0 |

2. How would you rate the difficulty of the material?

| Difficulty level | Number of students |
|---|---|
| Very difficult | 0 |
| Difficult | 5 |
| Just right | 15 |
| Easy | 2 |
| Very easy | 0 |

3. How do you compare your efficiency when learning with this self-paced course versus traditional classroom instruction (lectures + homework)?
5 = the self-paced course was much more productive, 4 = the self-paced course was somewhat more productive, 3 = the experiences are about equal, 2 = traditional lectured courses are somewhat more productive, 1 = traditional lectured courses are much more productive

| Learning efficiency | Number of students |
|---|---|
| 5 | 19 |
| 4 | 2 |
| 3 | 1 |
| 2 | 0 |
| 1 | 0 |

4. In other courses you take, would you prefer to learn by doing at your own pace like in this course, or using the traditional lecture + homework format?
5 = strongly prefer the self-paced learning by doing used in this course, 4 = somewhat prefer the self-paced learning by doing used in this course, 3 = no preference, 2 = somewhat prefer traditional lectures + homework, 1 = strongly prefer traditional lectures + homework

| Learning preference | Number of students |
|---|---|
| 5 | 20 |
| 4 | 2 |
| 3 | 0 |
| 2 | 0 |
| 1 | 0 |

In summary, the survey shows that students clearly prefer the competency-based model over traditional classroom instruction.

## 9 Representative Student Testimonials

“This class was extraordinary. [Instructor’s] use of NCLab helped me to actually learn and understand the applications of Python, rather than memorizing and regurgitating code. It let me learn by doing, making mistakes, and presenting everything in a very approachable way. I feel like I could adequately use basic Python in a job and I am very grateful to have learned it in such a hands on, low stress way.”
“[Instructor] and the NCLab platform are so effective at teaching various subjects in math and data. I’ve taken linear algebra and Python, and feel like I’ve retained more knowledge on these classes then any other I’ve taken.”
“This course is very well designed and makes getting help for yourself easy, but [Instructor] is also always available to help and is happy to listen to and work through your issues! He has designed the course exceptionally well.”
“I would not have changed anything about this class, it was perfect in presenting totally new material in a way none of my CS classes have ever been able to teach it.”

"This course is the best self directed course I have ever taken. There are built in features that make it easy to solve your own issues when you need, but also explains the topics well enough for you to have a strong foundational knowledge of the material that make getting outside help easy. It also provides access to fur her material if you want to further your own knowledge."

"I loved the class format. If I weren't graduating I would have taken SQL with NCLab next semester. I really enjoy the fact that this format allows us to fit the course work into our busy schedules whenever is best, including weekends or late nights. I really wished that more CS/Math classes at [University] would have been this same way, it would have created a lot less stress. Also asking for help on a problem was easy due to the professor having weekly office hours and also being open to answering questions through email. Lastly, some of the students at [University] also work and have extracurriculars that we need to juggle on top of school work. This format really helps with bringing flexibility to our schedule by allowing us to work during the day or fulfill our internship hours throughout the week more easily."

"I really enjoyed the course for a few reasons: I liked how it incrementally built on previous sections and it helped me learn by doing. I wasn't necessarily unsuccessful in previous CS classes I have taken that were lecture based, but I got very frustrated and did not have easy access to help. They were discouraging and I felt like I didn't learn anything and really couldn't be expected to actually solve a problem using C or C++. In the NCLab Python course, I only had to reach out for help a handful of times and it was very empowering to be able to solve a problem on my own and actually have a true understanding of why my solutions were correct. I recently took geography 416 where I learned R, but I didn't actually learn R, I learned how to copy and paste code from lecture and tinker until it worked. This class was totally different, I feel like I could actually use Python in a job and be proficient from day 1. Another reason why I liked this class is because I have a chronic illness and taking a self paced course took so much stress away. I could work on it when I felt well, but there was no pressure to attend a lecture or study for an exam. Accessibility is a huge problem at [University], and just having this one class that I didn't have to worry about was so positive for my mental health. I didn't have to worry about finding a ride to and from school, I didn't have to worry about needing a bathroom break during class, I didn't have to find a note taker to revisit material, I could repeat a section as many times as I needed to if I couldn't remember a previous topic. Taking this class was one of the best decisions I made, and I have recommended it to many people because I got so much out of it. Thanks for designing an awesome course."

"The NCLab Python course taught me more than I've learned in any other class. The ability to learn by doing helped me out so much since you basically have to write different queries to pass every section and every section builds upon one another and at the end you get a final task that puts it all together. Python is a skill that employers are in need of so I really appreciate the certificate at the end of each section for me to put in my resume. Highly recommended."

"I wish more courses were offered with a self paced format. I found that I am able to challenge myself more and learn more effectively from this method."

"I really enjoyed this course and would absolutely take any similarly formatted courses available. Thanks for the great semester!"

"For computer science courses I greatly preferred the self-paced format of this course. I also appreciated the option to reach out for help and feedback. However, the help within the course was extremely helpful in itself. I loved how I could truly format my schedule in a way that I could continue my growth and learning while also making it to the other duties in my life."

"The course is going great! I've been through about seven or eight DataCamp courses for learning R but feel I did not retain the information very well, potentially because they are not very challenging. With your course, the problems we do at the end of each lesson are challenging enough that I feel I retain the concepts very well. So far, very much enjoying the course!

## Appendix A: Traditional Instruction Versus Self-Paced Learning By Doing From a Student's Point of View

My name is Alexander Freyer, and I am studying medical informatics/computer science at the University of Lübeck in Germany. This year, I took a one-semester break from my university study to expand my Python skills with the help of NCLab's Python Developer career training. I decided to do this because good knowledge of Python is essential for my future success, and I felt that my university was not giving me those skills. I am grateful for this opportunity to discuss how my experience with the NCLab's self-paced teaching method compares to my university studies.

Earlier this year, I was preparing for my winter exams. Some of the modules I had to study were 'Introduction into Programming' with Java as the given language, as well as 'Medical Image Reprocessing,' which used Python. In all honesty, I felt completely overwhelmed. I did not understand anything, nor did I know where to start. All I knew about were if-else statements, the while-loop, and how to write the famous 'hello world' program. That was it.

Let me explain the cause of my struggles. The class consisted of 90-min lectures or seminars in which the professor or tutor would explain concepts with the help of Powerpoint slides filled with large amounts of code. We were sometimes encouraged to apply the information and program in real-time which was quite helpful. However, once I lost focus, I had no clue how to approach the next line of code. That struggle continued for an entire semester.

Knowing that I needed to do something about it, I googled how to begin coding and which language I should start with. In one YouTube video, an experienced programmer suggested that newbies start with Karel. "Learning a language is like driving a vehicle," he explained. Once you get your driver's license, it doesn't really matter what car you drive. Although the experience may vary slightly, the traffic rules remain the same.

So, I went searching and found the famous book, *Karel the Robot: A Gentle Introduction to The Art of Programming*, by Pattis [25]. I also came across the short free Karel sampler course [22] from NCLab.

Who is Karel the Robot? Karel is a simple robot that can be programmed to solve mazes using very basic commands such as 'go' (move one square forward), 'left' (turn 90 degrees left), 'right' (turn 90 degrees right), 'get' (collect an item from the ground beneath you), and 'put' (place an object from your backpack on the ground where you stand). It also has the counting loop 'repeat', the conditional loop 'while', 'if-else' and 'if-elif-else' conditions, the keyword 'def' to define new commands, it can use functions, variables, recursion, even Python lists, etc.

The Karel language is almost just abbreviated English, but that makes it really efficient for teaching algorithmic thinking, problem solving, and programming logic. This language was designed in the 1980s at Stanford. Back then it was similar to Pascal, but a few years ago Dr. Solin adjusted it and expanded to be similar to Python, because Pascal is no longer a leading programming language today. Dr. Solin's textbook *Learn How to Think With Karel the Robot* is freely available online [34].

After completing the sampler course, I regained confidence in my knowledge and skills. But why was this the case?

For instance, I finally really understood the concept and purpose of the 'while' loop. Sometimes Karel was in front of a wall, and had to do something while the wall was in front of him, such as to look for a corner. Sometimes he had to walk straight to the next wall (while a wall was not in front of him). Or, he needed to place objects from his backpack on the ground until the backpack was empty (= while it was not empty). At another time, he had to turn left until he faced North (= while he did not face North).

In one of the tasks, I even wrote unknowingly the First Maze Algorithm (FMA), which was a great feeling. For illustration, Fig. 1 shows a sample maze where the FMA is used. In this maze, Karel's objective is to collect all tulips and roses (collectible objects), avoid water (obstacle), and enter the home square (Karel's destination):
Fig. 2 presents a program that uses the FMA. This algorithm guides Karel to pass through an arbitrary labyrinth where the path is one square wide, and can only go straight, to the left, or to the right. The program below will guide Karel through the maze from Fig. 1, collecting all tulips and roses, until he enters the home square:
The sampler course proved to be extremely helpful, therefore I decided to enroll in the full-length Computational Literacy course from NCLab. This was an extraordinary experience. Let me tell you why.

**Fig. 1** Sample maze which includes water (obstacle), tulips and roses (collectible objects), and the home square (Karel's destination)

```
while not home
    go
    if rose or tulip
        get
    if water
        right
        if water
            repeat 2
                left
```

**Fig. 2** Program based on the FMA which guides Karel to the home square, collecting all tulips and roses

In this course I felt guided. I had a starting point. There weren't dozens of slides to investigate. The information was delivered in a concise and comprehensive way, and examples for concepts were always provided. We were able to step through the code line by line and see what actually happens. Sometimes, solving tasks required me to think deeply and come up with creative solutions. I remember a task where Karel should find out whether there is a gem in each aisle or not. It took me quite a while before I could come up with an idea. It actually came to me when I was lying in bed. I grabbed my notebook, wrote it down, and tried it out. It worked!

These moments of excitement, clarity, and inspiration never happened in my university's programming course. There, often I just hoped that some friend would give me the code so that we could get done with the task as quickly as possible. As a result, I did not learn much. Even though the NCLab's Karel course was challenging at times, I never felt lost. I could review my notes, use the hints, and even reach out to the support team which answered my questions within a few hours.

In my university's programming course, the professor would tell us the solution, as well as the most common mistakes. But when you have no base, it is hard to understand and to follow. For example, he recommend doing W3Schools tutorials or buying some textbook. However, the information was quite overwhelming for a beginner. Also, it always was about the language itself and the syntax, rather than about the logic of programming. This I found weird because the module was called 'Introduction to Programming'. How should one think to find the solution of the task? What is actually happening behind the scenes? These questions were answered for me in the NCLab's Karel course.

In conclusion, the NCLab's Karel course helped me discover that I really enjoy computer programming. After finishing it, I enrolled into the NCLab's Python Developer career training. This training was every bit as good as the Karel course. It gave me lots of practice, and made me job-ready. This training is aligned with the PCEP™ (Certified Entry-Level Python Programmer) exam [29] by the Python Institute. It prepared me for the PCEP exam

really well. After finishing it, I passed the exam with a score of 96%, and landed an entry-level Python position at a big company located in Germany.

Because the NCLab's self-paced teaching method works really well for me, currently I am enrolled in the Advanced Python Developer career training. Here I am learning object-oriented programming, event-driven programming, and various advanced Python and Computer Science concepts. When I finish it, I will be ready to take the next-level PCAP™ (Certified Associate in Python Programming) exam [28] of the Python Institute.

## Appendix B: Python Curriculum in NCLab

NCLab provides two Python career training programs which produce job-ready Python programmers. The Python Developer career training is aligned with the Python Institute's PCEP certification [29], and the Advanced Python Developer career training is aligned with the Python Institute's PCAP certification [28]. The former consists of the following self-paced courses and software projects:

- *Introduction to Python*: This course provides a detailed and comprehensive overview of the Python programming language. Students learn Python by solving programming problems of gradually increasing complexity, using simple calculations, loops, conditions, local and global variables, functions, ternary conditional expressions, and basic exceptions handling. Students also become proficient in working with fundamental Python data structures, including tuples, lists, and dictionaries. Throughout the course, students are developing a good Python coding style and other good coding habits.
- *Working with Text in Python*: More than 80% of work computers do is processing text. In this course students learn how to process, analyze, and manipulate text strings with Python.
- *Plotting and Drawing with Python*: Python is known for its powerful graphic capabilities. In this course students learn how to use the powerful Python library Matplotlib for plotting and drawing.
- *Software Project 1 (Graphics Editor)*: Students build their own Graphics Editor based on Matplotlib. The Graphics Editor is able to create shapes such as squares, triangles, rectangles and circles, fill objects with color, move, scale and rotate shapes, and combine them to make complex drawings. In addition to substantial programming practice, this Software Project provides students with a valuable insight into the principles of good software design.
- *Working with Files in Python*: Most data are stored in files. Therefore, this course teaches students how to open files, read data from them, process the data, and write back to files.
- *Software Project 2 (Image Viewer)*: Students build their own Image Viewer in Python. The Image Viewer is able to read bitmap images from files, store them as 2D Numpy arrays, and visualize them with Matplotlib. In this Software Project students practice working with files, text strings, and the Numpy and Matplotlib libraries.
- *Data Visualization with Python*: The world we live in is driven by data. Therefore, in this course students learn how to visualize data in the form of simple graphs, bar charts, pie charts, color maps, surface plots, wireframe plots, and contour plots. Students also learn how to visualize data on 2D Cartesian grids and unstructured triangulations.
- *Data Analytics Minimum*: Most real-life applications of Python are to some extent related to Data Analytics (DA). Therefore, the DA Minimum course teaches students how to use the Pandas library and perform elementary Data Analytics with Python.
- *Computer Science Minimum*: Every Python developer must know the basics of Computer Science (CS) including the binary, octal, and hexadecimal numeral systems, and bitwise operators. These are also required for the PCEP exam. That's exactly what students learn in this course.
- *Intermediate Topics in Python*: This course covers selected intermediate Python concepts including variadic function, anonymous (lambda) functions, built-in functions any(), all(), map(), filter(), reduce(), eval() and exec(), iterables and iterators, and generator functions and generator expressions. Students also gain a deeper insight into mutability, shallow and deep copying, and exceptions handling.

- *PCEP Preparation Course*: This course includes several PCEP [29] practice exams and prepares students to score high on the PCEP exam. PCEP is an industry-recognized certification from the Python Institute that will add a significant weight to students' resumes. Students are encouraged to take the PCEP exam before starting to work on their Capstone Project.
- *Capstone Project*: Students choose one of two options:
  - *Option 1*: Look up open source projects on Github, find one that they like, and contribute to it by submitting a pull request. Their contribution must be consulted and approved by their instructor in advance.
  - *Option 2*: Implement their own program in Python and upload it to Github. The topic of the program is chosen by the student, but must be consulted and approved by their instructor in advance. Typically, a more substantial program is required compared to Option 1.

  In both cases students are required to create a free Github user account, and to install a Python IDE on their own computer or laptop.

The Advanced Python Developer career training comprises the following self-paced courses and software projects:

- *Software Project 3 (Digital Computer)*: In this software project students use Python to simulate logic gates, binary adders and multipliers, and implement their own simple model of a digital computer in Python.
- *Object-Oriented Programming 1*: This course covers the basics of OOP including the history of programming languages, how OOP evolved from procedral programming, the concept of encapsulation, classes, attributes, methods, and instantiation.
- *Software Project 4 (Turtle Graphics)*: In this software project students implement their own version of the Python Turtle Graphics.
- *Object-Oriented Programming 2*: This course covers inheritance and advanced concepts of OOP such as abstract classes and methods, polymorphism, multiple inheritance, mixin classes, the Diamond Problem, and the Abstract Base Classes (ABC) module.
- *Software Project 5 (OOP Upgrade of the Graphics Editor)*: Students use inheritance, polymorphism, and multiple inheritance to upgrade their Graphics Editor from Software Project 1 to an OOP design.
- *Event-Driven Programming*: Students learn even-driven programming while implementing the well-known desktop game Othello (Reversi).
- *Advanced Topics in Python*: This course covers selected advanced Python concepts including advanced applications of recursion to Polish (prefix) notation and binary trees, names and namespaces, packages and modules, decorators, making Python classes callable, working with JSON and XML data, etc.
- *PCAP Preparation Course*: This course includes several PCAP practice exams and prepares students to score high on the PCAP exam [28].
- *Capstone Project*: Substantial Python software project based on student's own choice. Instructor's approval of the topic, and working in Linux is required.

All of NCLab coursework is ADA compliant [1].

## Appendix C: Sample Autograded Python Exercise

Providing instant feedback to students, and validating their understanding of the subject is absolutely crucial in their learning process. Therefore, below we present a sample exercise and show how NCLab performs grading. The exercise is related to recursion:

*Write a recursive function permute(L) which for a list L returns a list of all permutations of L (i.e. a list of lists)! Do not use itertools. The Math and Numpy modules also provide functions to calculate permutations, but do not use them either. Do not change the main program. Expected output:*

Then follows the expected output of the main program:

[1, 2, 3, 4] [1, 2, 4, 3] [1, 3, 2, 4] [1, 3, 4, 2] [1, 4, 2, 3] [1, 4, 3, 2] [2, 1, 3, 4] [2, 1, 4, 3] [2, 3, 1, 4] [2, 3, 4, 1] [2, 4, 1, 3] [2, 4, 3, 1] [3, 1, 2, 4] [3, 1, 4, 2] [3, 2, 1, 4] [3, 2, 4, 1] [3, 4, 1, 2] [3, 4, 2, 1] [4, 1, 2, 3] [4, 1, 3, 2] [4, 2, 1, 3] [4, 2, 3, 1] [4, 3, 1, 2] [4, 3, 2, 1]

Matching this output successfully is the first test which allows the students to see immediately if their function produces an incorrect result. But even if they match the expected output successfully, additional tests will be performed.

There is a large number of various grading criteria a course designer can choose from. One of them is a list of forbidden keywords. In this particular exercise, the forbidden keywords include "import" which prevents students from importing itertools, math or numpy and using recursion from there.

In the code input cell the students find the main program (which they know they must not change). They also know that the ellipsis "…" is to be replaced with their function *permute(L)*:

```
...

# Main program (do not change):
numbers = [1, 2, 3, 4]
for p in permute(numbers):
    print(p, end=' ')
```

Some students will be able to figure out the solution on their own, others will need help. For the latter group, there is always a hint. In this case, it walks them through the algorithm for permutations of a three-item list:
Here is the main idea, illustrated on a 3-item list L = [1, 2, 3]:

- Extract from [1, 2, 3] the first item 1. The remainder of the list is [2, 3]. Obtain all permutations of this shorter list recursively. These are [2, 3] and [3, 2]. In both cases insert the item 1 back at the beginning, obtaining [1, 2, 3] and [1, 3, 2].
- Extract from [1, 2, 3] the second item 2. The remainder of the list is [1, 3]. Obtain all permutations of this shorter list recursively. These are [1, 3] and [3, 1]. In both cases insert the item 2 back at the beginning, obtaining [2, 1, 3] and [2, 3, 1].
- Extract from [1, 2, 3] the third item 3. The remainder of the list is [1, 2]. Obtain all permutations of this shorter list recursively. These are [1, 2] and [2, 1]. In both cases insert the item 3 back at the beginning, obtaining [3, 1, 2] and [3, 2, 1].

The solution to this exercise is as follows. Students will see it in a side-by-side comparison with their own code once their function passes the grader:

```
def permute(L):
    """Creates a list of all permutations of list L"""

    # Obtain the length of L:
    n = len(L)

    # If n == 1, return a one-item list containing L:
    if n == 1:
        return [L]

    # Create an empty list R for the result:
    R = []
```

```
    # Use a for loop with n cycles:
    for i in range(n):

        # Read ith item from L:
        item_i = L[i]

        # Create a remainder without the ith item:
        remainder = L[:i] + L[i+1:]

        # Use a for loop to go over all permutations p of the remainder:
        for p in permute(remainder):

            # Append to R the ith item (as a list) + p:
            R.append([item_i] + p)

    # Return the result:
    return R
```

The students can run their program as many times as they want without penalties. But when they press Submit, their code is sent to a remote server for grading. Below we show the main parts of the autograding code for this particular exercise.

The grader begins with checking if the name "permute" is defined in the first place:

```
try:
    permute
except NameError:
    lab.grade(False, "Name 'permute' is undefined.")
    return False
```

As a next test, the grader checks if the name *"permute"* is a callable function (in this code snippet as well as in the following ones, lines are artificially wrapped with the backslash symbol '\' to fit the page width):

```
if not callable(permute):
    lab.grade(False, "It seems that 'permute' is not a callable function.")
    return False
```

Then follows the master solution *permute_sol_(L)* which is used to check the results of the student's function (the code is the same as above, therefore we won't repeat it here):

```
def permute_sol_(L):
    ...
```

Finally the student's function is called with several test lists, and the result is compared to the result produced by the master solution *permute_sol_(L)*:

```
for p_ in [['a'], ['a', 'b'], ['a', 'b', 'c'], \['a', 'b', 'c', 'd'], ['a', 'b', 'c', 'd', 'e'], ['a', 'b', 'c', 'd', 'e']]:
    try:
        t_ = permute(p_)
    except Exception as e:
        lab.grade(False, f"Test call permute({p_}) caused an exception:")
        lab.grade(False, str(e))
        return False
    t_sol_ = permute_sol_(p_)
    if sorted(t_) != sorted(t_sol_):
        lab.grade(False, f"Test call permute({p_}) returned an incorrect result.")
        lab.grade(False, "Your result: " + str(t_))
        lab.grade(False, "I expected: " + str(t_sol_))
        return False
```

## References


1. ADA: Information and technical assistance on the Americans with disabilities act. https://www.ada.gov/2010ADAstandards_index.htm. Accessed 20 September 2020
2. Akçayır, G., Akçayır, M.: The flipped classroom: a review of its advantages and challenges. Comput. Educ. **126**, 334–345 (2020)
3. Alaoutinen, S., Smolander, K.: Student self-assessment in a programming course using Bloom's revised taxonomy. In: Laxer, C. (Ed.) Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education, pp. 155–159 (2010)
4. Ben-Ari, M.: Constructivism in computer science education. ACM SIGCSE Bull. **30**(1), 257–261 (1998)
5. Benjamin, L.T., Jr.: Lecturing. In: Davis, S.F., Buskist, W. (eds.) The Teaching of Psychology: Essays in Honor of Wilbert J McKeachie and Charles L Brewer, pp. 57–67. Lawrence Erlbaum Associates, Mahwah (2002)
6. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming. ACM SIGCSE Bull. **39**(2), 32–36 (2007)
7. Bradbury, N.A.: Attention span during lectures: 8 seconds, 10 minutes, or more? Adv. Physiol. Educ. **40**(4), 509–513 (2016)
8. Chegg: Find solutions for your homework. http://chegg.com. Accessed 20 September 2020
9. CourseHero: Find online study resources—better grades start here. http://coursehero.com. Accessed 15 September 2022
10. Davis, B.G.: Tools for Teaching. Jossey-Bass, San Franciso (1993)
11. Feurzeig, W., Papert, S., Solomon, C.: Logo, an introductory programming language for beginners. Created in (1967)
12. Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., Tutty, J.: Predictors of success in a first programming course. In: Tolhurst, D., Mann, S. (eds) Proceedings of the 8th Australasian Conference on Computing Education, vol. 52, pp. 189–196 (2006)
13. Ford, M., Venema, S.: Assessing the success of an introductory programming course. J. Inf. Technol. Educ. Res. **9**(1), 133–145 (2010)
14. Hartley, J., Davies, I.K.: Note taking: a critical review. Program Learn Educ Technol **15**, 207–224 (1978)
15. Hawi, N.: Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. Comput. Educ. **54**(4), 1127–1136 (2010)
16. Jenkins, T.: On the difficulty of learning to program. In: Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, pp. 53–58 (2002)
17. Kinnunen, P., Malmi, L.: Why students drop out CS1 course? In: Anderson, R., Fincher, S. A., Guzdial, M. (eds) Proceedings of the 2nd International Workshop on Computing Education Research, pp. 97–108 (2006)
18. Konecki, M.: Problems in programming education and means of their improvement. In: Katalinic, B. (ed) Chapter 37 in DAAAM International Scientific Book 2014, pp. 459–470. Published by DAAAM International, ISBN 978-3-901509-98-8, ISSN 1726-9687, Vienna, Austria (2014). https://doi.org/10.2507/daaam.scibook.2014.37
19. Marken, S., Crabtree, S.: U.S. Students' Computer Science Participation Lags Interest. Gallup Poll. 30 September 2021
20. McCartney, R., Moström, J.E., Sanders, K., Seppälä, O., Simon, B., Thomas, L.: A multi-national study of reading and tracing skills in novice programmers. ACM SIGCSE Bull. **36**(4), 119–150 (2004)
21. McKeachie, W.J.: Teaching Tips: Strategies, Research and Theory for College and University Teachers. Heath, Lexington (1986)
22. NCLab Free Sampler Courses: http://nclab.com/samplers. Accessed 15 September 2022
23. Network Computing Laboratory (NCLab): http://nclab.com. Accessed 15 September 2022
24. Nikula, U., Gotel, O., Kasurinen, J.: A motivation guided holistic rehabilitation of the first programming course. ACM Trans. Comput. Educ. (TOCE) **11**(4), 1–38 (2011)
25. Pattis, R.E.: Karel the Robot: A Gentle Introduction to the Art of Programming. Wiley, Hoboken (1994)
26. Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., Paterson, J.: A survey of literature on the teaching of introductory programming. ACM SIGCSE Bull. **39**(4), 204–223 (2007)

27. Peng, W.: Practice and experience in the application of problem-based learning in computer programming course. In: Yuting, L. (ed.) Proceedings of the International Conference on Educational and Information Technology (ICEIT), vol. 1, pp. 170–172 (2010)
28. Python Institute's PCAP (Certified Associate in Python Programming) exam. https://pythoninstitute.org/pcap. Accessed 15 September (2022)
29. Python Institute's PCEP (Certified Entry-Level Python Programmer) exam. https://pythoninstitute.org/pcep. Accessed 15 September 2022
30. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: a review and discussion. J. Comput. Sci. Educ. **13**(2), 137–172 (2003)
31. Sams, A., Bergmann, J.: Flip your students' learning. Educ. Leadersh. **70**, 16–20 (2013)
32. Scratch. An online platform for learning computer programming http://scratch.mit.edu. Accessed 15 September 2022
33. Smith, P.A., Webb, G.I.: The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers. J. Educ. Comput. Res. **22**(2), 187–216 (2000)
34. Solin, P.: Learn how to think with Karel the robot. http://femhub.com/pavel/work/textbook-karel-new.pdf. Accessed 15 September 2022
35. Solin, P.: Self-paced, instructor-assisted approach to teaching linear algebra. Math. Comput. Sci. **15**(4), 1–27 (2021)
36. Solin, P., Roanes-Lozano, E.: Using computer programming as an effective complement to mathematics education: experimenting with the standards for mathematics practice in a multidisciplinary environment for teaching and learning with technology in the 21st century. Int. J. Technol. Math. Educ. **27**(3), 147–156 (2020)
37. Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. ACM Trans. Comput. Educ. (TOCE) **13**(4), 1–64 (2013)
38. Tan, P.H., Ting, C.Y., Ling, S.W.: Learning difficulties in programming courses: undergraduates' perspective and perception. In: Jusoff, K., Othman, M., Xie, Y. (eds) Proceedings of the IEEE International Conference on Computer Technology and Development, vol. 1, pp. 42–46 (2009)
39. Wankat, P.C.: The Effective Efficient Professor: Scholarship and Service. Allyn and Bacon, Boston (2002)
40. Yadin, A.: Reducing the dropout rate in an introductory programming course. ACM Inroads **2**(4), 71–76 (2011)