Introductory Computer Science Course by Adopting Many Programming Languages

Francisco de Assis Zampirolli[1]*, Fernando Teubl, Guiou Kobayashi, Rogério Neves, Luiz Rozante, Valério Ramos Batista

[1]Federal University of ABC (UFABC)
Av. dos Estados, 5001 – Santo André – 09210-580 – SP – Brazil

Abstract. *Teaching programming logic by means of a single Programming Language (PL) may lead the whole process to a particular syntax and specific libraries. In order to let every student choose their preferred PL we have developed a method that includes didactic material in many PLs by means of notebooks in Colab. We created a filter that generates Lecture Notes in different combinations of PLs from these notebooks. Moreover, each student can choose different PLs to practice with exercises and send their solutions as programming codes, which are individualized because of the parametric questions generated with MCTest+Moodle+VPL. Herewith we present our method, which is easily adaptable, validated with 5 remote classes comprising a total of 223 students, whose average pass rate was 90%.*

1. Introduction

Conceived by D. Knuth in 1984, the Literate Programming is a paradigm that fosters the alternation between codes and documentation, so that programming can result in either a literary work or a scientific paper [Knuth 1984]. "The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer". [1]

The content development environments *Jupyter* (`jupyter.org`) and *Google Colab* (`colab.research.google.com`) also follow this concept by grouping codes and documentation in a JSON file with `ipynb` extension. These files combine cells of text and of code, and the latters can be formatted in Markdown, LaTeX, HTML or JavaScript. In these environments Python is the native PL for the code cells but they can also run others. The `ipynb` file is called *notebook*, the name we shall use throughout from now on.

These environments not only allow for literate programming, but can also be used in order to produce material complementary to didactic books, which normally include a specific PL in their examples. For instance, in [Géron 2019] the author created a notebook for each chapter, with which the reader can easily reproduce all of the examples described in the book by opening files on Jupyter and running the cells.

Our present work complements [Géron 2019] because our method generates notebooks in many PLs by customizing the first line of text and code cells, which define the PLs they are referring to. We have also implemented a filter that automatically generates specific notebooks in different PLs. Moreover, this filter converts the notebooks into other formats like HTML and PDF, and it also assembles many notebooks in digital book format. In this way, professor and students can each choose their preferred PLs independently, be it

*Grant #2018/23561–1, São Paulo Research Foundation (FAPESP).
[1]`www-cs-faculty.stanford.edu/~knuth/lp.html`

to lecture or to study. Additionally, the integration MCTest+Moodle+VPL enables each student to choose the PLs in which they will solve activities individualized by parametric questions. These solutions consist of program codes, whose corrections are then carried out automatically on Moodle [Zampirolli et al. 2020]. We have applied our method as part of a fully remote course called Introduction to Computer Science (ICS), which has been giving from May to August 2021 and encompasses 5 classes. Didactic material includes the book [Neves and Zampirolli 2017] with lecture notes in the form of notebooks.

2. Background

The Brazilian Syllabus Guidelines for Computer Science (`goo.gl/35CmzT`) determine that the five types of graduation disciplines in the country must prepare graduates to "solve problems through programming environments". Therefore, a PL represents one of the means to solve problems but it is not the main purpose of a course by itself. This is specially important for heterogeneous classes, in which students will follow different Areas of Knowledge. Herewith we present an important resource to make this new teaching paradigm viable, once it is based upon competencies.

2.1. Programming Environments in Many PLs

Regarding web platforms and online services, they both have been under sunstantial development over the years. For instance, modern users can write codes in a web window with their preferred PL, and they do not even have to install software therefor. Many of these Programming Environments (PEs) offer various PLs and a resourceful text editor, all built-in on a web page that also displays feedback whenever the user debugs or runs the source-code remotely.

Some well-known PEs are Studio App Center (MicroSoft Azure), `replit.com`, and `codepen.io` among others. Differently from services devoted to developers, like GitHub and jetbrains, those enable beginners to try diverse PLs and get familiar with their special features by means of tutorials, examples and practical experience.

However, the hitherto available online and printed didactic material have both been adapted to each PL. As a consequence, students face distinct experiences among PLs. Also, the adopted teaching methods bring along discrepancies in levels of difficulty and thoroughness.

Both Jupyter and Colab project teams have been offering multiplatform tools and services, which aim at standardizing the learning process and also make programming available to a broader public. Any other method similar to the one presented in this paper is still incipient.

2.2. Automatic Correction of Codes

The Virtual Programming Lab (VPL – `vpl.dis.ulpgc.es`) is a Moodle extension introduced in [Rodríguez del Pino and et al. 2010], namely one of the most important works regarding automatic correction in code programming activities.

Comprising various PLs, VPL has been developed consistently over the years with improvements as the one achieved in [Rodríguez-del Pino and et al. 2012]: an integrated plagiarism detector to increase the reliability on authorship in scored activities.

Later in [Thiébaut 2015] the author shows the application of VPL to evaluate automatically two classes of PLs. One the one hand the results turned out to be positive but one the other hand the paper revealed some characteristics that had not been supported by VPL yet. For instance, it could neither deal with random input nor allow for penalizing marks in case of delayed submissions. Now we have overcome these two limitations with our method.

In [Christian and Trivedi 2016] the authors compare eleven tools devoted to evaluating codes of programming exercises, specially regarding the diversity of supported PLs, the architecture and the grading metrics.

2.3. Problems of Adopting a Single PL

Adopting a single PL to a whole class will make each student face different challenges. For example, some can have difficulty in memorizing either its syntax or structuring, others may want to use resources that only exist in another language, and so on. There are numerous works that aim at making things easier for beginners. In [Sorva et al. 2013] the authors present an extensive survey of generic program visualization systems devoted to introductory courses.

In a class where some of the students have already worked with another PL, these can oppose the adopted PL because their skills diverge therefrom. Moreover, some PLs are simpler but more restrictive than others, and there is no ideal choice for a heterogeneous class. However, many works [Dingle and Zander 2000, Wainer and Xavier 2018] compare differences between two or more PLs in the same discipline as an attempt to identify which one is the most adequate for each course.

Our present paper differs from the aforementioned works because we propose many PLs for the same programming course. Namely, every student is free to choose one of the available PLs in each activity. Our experiment involved only parametrized activities, which generate individualized exercises in order to curb plagiarism. Moreover, correction is fully automatic and includes feedback to the student. Our method is original to the best of our knowledge.

3. Method

Here we contextualize the application of our method to the course Introduction to Computer Science (ICS), and also present the notebook filter and the format converter that both produce notebooks for the lecture notes of the course.

3.1. Contextualization

ICS is offered by our university and consists of three plus two weekly hours of theoretical and practical lectures, respectively. The latters are devoted to computer lab, and ICS takes 12 weeks in total. Due to the covid-19 pandemic all courses became fully remote, and our institution allowed professors to choose either synchronous or asynchronous lectures. Moreover, in order not to hinder low income students, who normally have little access to computer and internet, every activity that counts for the final mark remains available for at least 72h.

Therefore we planned eight lists of exercises, each list for a specific week of the course, so that another four weeks are devoted to revision and exams. Individualized through parametric questions, the lists are generated in PDF and emailed to each student automatically by MCTest, an open source system that also performs automatic correction of the students' solutions (see `vision.ufabc.edu.br` for details).

However, MCTest corrects multiple-choice questions, whereas ICS requires the student to send program codes. These are corrected automatically via Moodle+VPL, and for that we have prepared lists and exams consisting of four parametrized questions each.

During every week students can watch the lectures that help solve the corresponding list. Either synchronous or asynchronous lectures refer to typical classroom's five weekly hours, which now mostly include emailed questions answered by professors and TAs. Each list was scheduled to be sent on a Tuesday and the submission deadline of student's solutions on the following Monday, in both cases at 6am for a technical reason: at this time the three VPL servers set up for the automatic corrections do not get overloaded.

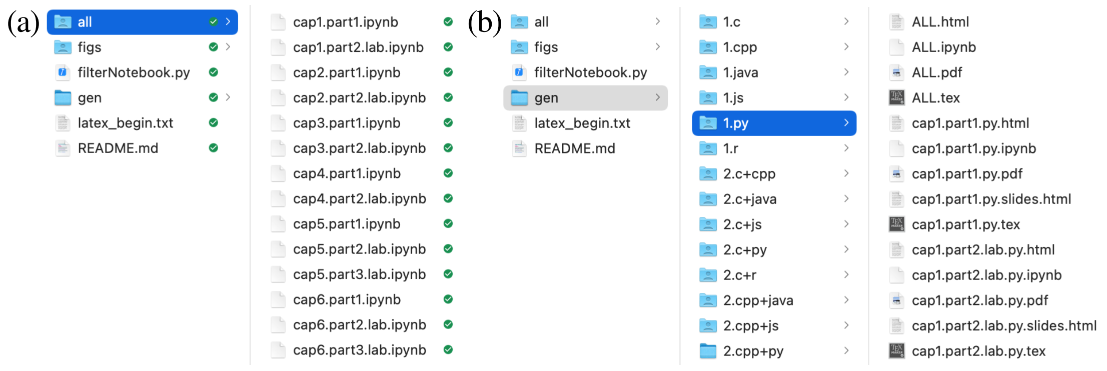
Concerning how to create these lists of exercises with parametrized questions, for details we refer the reader to [Zampirolli et al. 2020]. Exams follow the same format as the weekly lists. The programme of the course can be checked in Table 1.

Table 1. Course Programme.

Week	Subject	Evaluations
1	Introduction	List 1
2	Organizing a Code (functions)	List 2
3	Simple and Composed Conditionals	List 3
4	Loops	List 4
5	Revision and Exercises	Exam 1
6	Arrays - Part 1	List 5
7	Arrays - Part 2	List 6
8	Matrices - Part 1	List 7
9	Matrices - Part 2	List 8
10	Revision and Exercises	Exam 2
11	Revision and Exercises	Replacement Exam
12	Revision and Exercises	Retake Test

3.2. Didactic Material

The content in Table 1 was adopted to publish a book for the course Introduction to Computer Science [Neves and Zampirolli 2017]. There we have seven chapters that cover Foundations, Code Structuring, Conditionals, Loops, Arrays, Matrices, and Advanced Topics in this order. For each of the first six chapters we supplied an `ipynb` file, namely a notebook with the lesson notes that deal with the concepts and their respective examples in the following PLs: Python, R, Java, JavaScript, C++ and C. Besides these conceptual notebooks we have also created others for the lists of exercises in the practical lectures. All these notebooks were stored in the folder `all` depicted in Figure 1a.


(a)

all	cap1.part1.ipynb
figs	cap1.part2.lab.ipynb
filterNotebook.py	cap2.part1.ipynb
gen	cap2.part2.lab.ipynb
latex_begin.txt	cap3.part1.ipynb
README.md	cap3.part2.lab.ipynb
	cap4.part1.ipynb
	cap4.part2.lab.ipynb
	cap5.part1.ipynb
	cap5.part2.lab.ipynb
	cap5.part3.lab.ipynb
	cap6.part1.ipynb
	cap6.part2.lab.ipynb
	cap6.part3.lab.ipynb

(b)

all	1.c	ALL.html
figs	1.cpp	ALL.ipynb
filterNotebook.py	1.java	ALL.pdf
gen	1.js	ALL.tex
latex_begin.txt	1.py	cap1.part1.py.html
README.md	1.r	cap1.part1.py.ipynb
	2.c+cpp	cap1.part1.py.pdf
	2.c+java	cap1.part1.py.slides.html
	2.c+js	cap1.part1.py.tex
	2.c+py	cap1.part2.lab.py.html
	2.c+r	cap1.part2.lab.py.ipynb
	2.cpp+java	cap1.part2.lab.py.pdf
	2.cpp+js	cap1.part2.lab.py.slides.html
	2.cpp+py	cap1.part2.lab.py.tex

Figure 1. (a) Structure of files and folders; (b) Cutout showing the structure of files and folders generated automatically. Here we detail the subfolder `1.py` for Python.

Figure 1a also shows how theoretical and practical activities were named: `cap⋆.part⋆.ipynb` and `cap⋆.part⋆.lab.ipynb`, respectively. The folder `figs` contains all pictures employed to the notebooks.

3.3. Cell Filter

In each of the `ipynb` file from the `all`-folder the first line of either text or code cell contains the chosen PL for that cell content. It follows the syntax `#[`$\texttt{type}_1$`]#[`$\texttt{type}_2$`]#`···`#[`$\texttt{type}_n$`]#`, in which $\texttt{type}_i$ is the file extension of the code in the chosen PL. For instance, Table 2 illustrates code cells in the six PLs. On the left handside we have the code and on the right handside its compilation/execution on Colab. For some PLs like Python, R, Java and JavaScript one can execute the code inside the very cell depicted on the left, so it is not necessary to save the code in a file (with the command `%%writefile` and then run it). In Figure 1a the file `filterNotebook.py` filters such cells from the `all`-folder into the folder `gen`, whose content is then generated automatically (see Figure 1b).

Table 2. Code cells before applying the filter. After executing `filterNotebook.py` the first line containing `#[`$\texttt{type}_i$`]#` will be removed. Moreover, cells whose first line indicate another PL will not appear in the notebooks created automatically.

code	compile/run

`#[py]#`
`%%writefile cap01exem01.py`
`print("Hello, World!")`
	`#[py]#`
	`!python cap01exem01.py`

`#[r]#`
`%%writefile cap01exem01.r`
`cat("Hello, World!")`
	`#[r]#`
	`!Rscript cap01exem01.r`

`#[js]#`
`%%writefile cap01exem01.js`
`console.log("Hello, World!")`
	`#[js]#`
	`!node cap01exem01.js`

`#[java]#`
`%%writefile cap01exem01.java`
`class cap01exem01 {`
`  public static void main (String[] args) {`
`    System.out.println("Hello, World!");   }}`
	`#[java]#`
	`!javac cap01exem01.java`
	`!java cap01exem01`

`#[c]#`
`%%writefile cap01exem01.c`
`#include <stdio.h>`
`int main(void) {`
`  printf("Hello, World!"); return 0; }`
	`#[c]#`
	`!gcc cap01exem01.c -o output`
	`!./output`

`#[cpp]#`
`%%writefile cap01exem01.cpp`
`#include <iostream>`
`using namespace std;`
`int main(void) {`
`  cout << "Hello, World!" << endl; }`
	`#[cpp]#`
	`!g++ cap01exem01.cpp -o output`
	`!./output`

In Figure 1b the subfolders of `gen` are named $\texttt{n.type}_1$+···+$\texttt{n.type}_n$, which indicates the `n` chosen PLs with their signatures, as filtered by `filterNotebook.py` (see details in Subsection 3.4).

3.4. Converters

Every subfolder of `gen` contains files in several formats besides the notebook with extension `ipynb`, exemplified as `ALL.ipynb` in Figure 1b. There we see that the implemented filter also converts notebooks into HTML, LaTeX and PDF. These conversions are performed by the following command:

```
jupyter nbconvert file.ipynb --to format
```

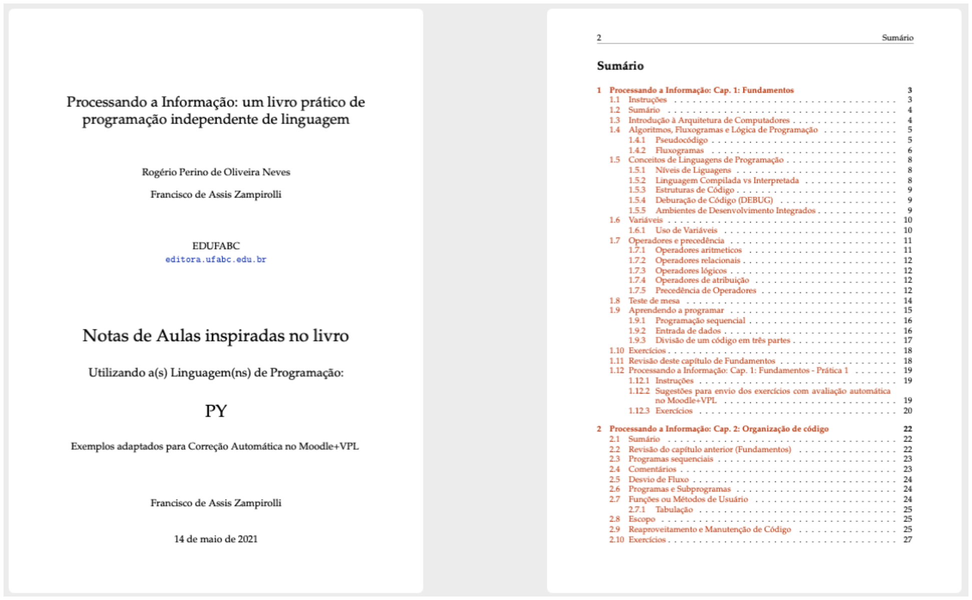
The parameter `format` can assume many formats defined at the beginning of the file `filterNotebook.py` by means of the variable `Formats`. For example, `Formats = ['html', 'slides', 'latex']`.[2]

In order to personalize the generated PDF we resorted to another conversion command, namely from LaTeX to PDF:

```
pdflatex --shell-escape -interaction nonstopmode file.tex
```

[2]These are some formats into which conversion is possible with the command `jypyter nbconvert`, see `nbconvert.readthedocs.io/en/latest/usage.html`.

Figure 1a indicates the file `latex_begin.txt`, which contains a header to the `tex` file. The header includes a cover and a summary into the automatically generated PDF. Figure 2 illustrates it for Python.



Processando a Informação: um livro prático de programação independente de linguagem

Rogério Perino de Oliveira Neves

Francisco de Assis Zampirolli

EDUFABC
editora.ufabc.edu.br

Notas de Aulas inspiradas no livro

Utilizando a(s) Linguagem(ns) de Programação:

PY

Exemplos adaptados para Correção Automática no Moodle+VPL

Francisco de Assis Zampirolli

14 de maio de 2021

2 Sumário

Sumário

Figure 2. Cover and contents of the PDF for Python generated automatically.

Next we give the full explanation of the command to run the implemented filter in `filterNotebook.py`:

```
python filterNotebook.py file type format

file: can be a file as all/cap1.part1.ipynb or even the whole folder all;
type: can be one of the extensions py, js, java, c, cpp, r or even all (the six exten-
sions).  The user can also choose type_1+···+type_n as explained above, e.g.  py+js, but they
will come in alphabetical order js+py;
format: html, slides, latex, all or none (if omitted).
```

For instance, `python filterNotebook.py all py` will filter all text and code cells in Python from the whole content of the folder `all`, hence generating the `ipynb` files inside the folder `1.py` (see Figure 1b).

In the previous example, by replacing `py` with `all` we also get the files `gen/*/ALL.ipynb` together will the entire set `gen/*/*.ipynb`, as indicated in Figure 1b. Moreover, these files will also be converted to HTML, LaTeX and PDF. As an example, the first two pages of `gen/1.py/ALL.PDF` can be seen in Figure 2.

The reader may use this `filterNotebook.py` and other files to simulate examples like the one just presented here. Such files are available at `https://github.com/fzampirolli/filterNotebook` and also at `https://editora.ufabc.edu.br/matematica-e-ciencias-da-computacao/58-processando-a-informacao`.

As indicated in Figure 1b, the number N of combinations of PLs that can generate subfolders depends on the number of PLs defined by the variable `Types` in the header of `filterNotebook.py`. For instance, with `Types = ['py', 'js', 'java', 'c', 'cpp', 'r']` we generated 63 subfolders. The last one was named `6.c+cpp+java+js+py+r`, which comprises the cells of all six PLs per notebook contained in the folder `all`. It is easy to see that $N = \sum_{p=1}^{m} \frac{m!}{(m-p)!p!}$, where m is the number of PLs in `Types`.

4. Results and Discussion

Here we detail the context in which the course ICS was conducted, specially regarding our experiments.

4.1. Experiments

Both the method and didactic material presented in Section 3 were adopted by five classes, just one evening class thereof, with a total of 223 matriculated students. To these classes were assigned four professors, who agreed on evaluation criteria informed to the students already in the first remote lecture. They consist of students that will follow many different Areas of Knowledge because the course belongs to the Bachelor in Science and Technology, which is multidisciplinary. Thereafter, students may choose other bachelor programs, engineering, etc.

Regarding the three professors who were each assigned to only one class, they opted for weekly synchronous lectures (both theoretical, 3h, and practical, 2h). Since program codes were submitted to an automatic corrector, which included a plagiarism detection tool, these professors required their students to record short explanatory videos for the activities after Week 2 in Table 1. The videos were worth 50% of the activities' mark. Their three classes consisted of 137 students (referred to as **Group 1**).

The fourth professor posted his asynchronous lectures every Monday, prepared with Colab and slides, and he forfeited the explanatory videos. His two classes consisted of 86 students (**Group 2**). However, the remaining was identical for the five classes: programme of the course, weekly evaluations, exams, unified corrections, and the PL Python (in fact, the students preferred Python to the others).

Next we are going to analyze these two groups statistically, and see whether their distinction had any influence in the teaching strategies.

4.1.1. Questionnaire: Student's Profile

At the very beginning of the course students were asked to fill out an online questionnaire devoted to identifying the profile of each class.[3] This enabled each professor to elaborate their teaching strategies. For instance, classes C_1 to C_5 had a total of 39, 40, 30, 35 and 38 responses, respectively.

The questionnaire asked for: the student's personal data, information about their access to computer and internet, their previous knowledge of PLs, which one they preferred for ICS, and their interest in synchronous or asynchronous lectures in either case (theoretical and practical). Only 23% preferred theoretical synchronous lectures compared with 34% for practical ones, while 21% remained indifferent in this case.

Regarding previous knowledge of some PL we had 48% Python, and 45% willing to learn this language in the course. Other PLs they either know or want to learn are scattered among JavaScript, Java, C, C++ and R. It is worth mentioning that ICS is offered in the third academic term. In the first one the students attend the introductory course Foundations of Computer Science, for which we attempted to unify the didactic material and evaluations in 2020 by means of Colab, Python and its library Pandas, devoted to handling CSV files.

[3]Questionnaire in Portuguese: `https://forms.gle/DQMrGpNvg4QQvpdd7`

4.1.2. Questionnaire: Student's Feedback

During Week 9, right before the last exams, the students received another questionnaire. Now this one aimed at detecting potential flaws in the coordination of the five classes.[4]

They were asked about general topics, didactic material and evaluations as follows:

```
01. My PREVIOUS knowledge of PL was already quite good before the course
02. My knowledge of PL improved a lot AFTER the course
03. Handling several PLs can be useful for my ACADEMIC LIFE in OTHER COURSES
04. Handling several PLs can help in my PROFESSIONAL CAREER
05. SYNCHRONOUS THEORETICAL lectures are the best choice for learning
06. SYNCHRONOUS PRACTICAL lectures are the best choice for learning
07. It is important to offer the DIDACTIC MATERIAL in many PLs at the student's wish
08. COLAB is an important DIDACTIC MATERIAL for the course
09. I would recommend this course to my colleagues because it offers DIDACTIC MATERIAL in
    many PLs
10. Regarding the DIDACTIC MATERIAL in many PLs, the one offered to my class is very good
11. It is important to offer EVALUATIONS in many PLs at the student's choice
12. It is appropriate to use Moodle+VPL in the EVALUATIONS with automatic correction and
    immediate feedback
13. I approve of the 2min-explanatory VIDEO applied to every EVALUATION, because this
    improved my learning (if your class was discharged, would it have been useful?)
14. The INDIVIDUALIZED EVALUATION helps curb plagiarism and therefore improves learning
15. The Individualized Weekly EVALUATION is important
16. I would recommend this course to my colleagues because it offers EVALUATIONS in many PLs
17. Regarding the EVALUATIONS in many PLs, the ones offered to my class are very good
```

Moreover, the following query was added separately:

```
Suggestions you would give to improve our teaching-learning method for this course. If you
disagree with some of the questions 1-17, could you propose improvements?
```

Since only three questions can really be a watershed between **Groups 1** and **2**, namely 5, 6 and 13, we are going to show their answer graphs from both groups. At the end of this section we present the overall averages of both groups, together with some statistical analysis.

Figure 3a illustrates the BoxPlot graphs of answers to questions 1-17. There the 1st quartile is the threshold that separates the lowest 25% of the scores; the 2nd quartile is the median; the 3rd quartile separates the highest 25% of the scores; the white square stands for the average [Jw 1977]. Among these graphs we highlight questions 3 and 4 about the importance of knowing several PLs in academia and in the lab our market, respectively. They both got the best scores. We also emphasize Figure 3b, which shows the 88% average approval rate of questions 7 and 8. Average scores of questions 11-17 are summarized in Figure 3c. They dealt with Evaluations, and we point out question 13, which attained an approval rate of just 56%. Although only 53 students responded to this questionnaire, maybe due to the final exams of various courses, we were able to carry out the following analysis.

We adopted Student's T Test [Lowry 2014] to compare the averages of the two independent groups, with questions in the questionnaire with confidence level $\geq 90\%$: 1, 6, 9, 13, 14, 16 and 17. For example, in question 6, with population averages $\mu_1 = 4.000$ (**Group 1**) and $\mu_2 = 3.368$ (**Group 2**). These groups were defined in Subsection 4.1. The hypothesis test is $H_0 : \mu_1 = \mu_2$ vs $H_1 : \mu_1 \neq \mu_2$. **Groups 1** and **2** consist of 34 and 19 students, respectively. H_0 was rejected when the T Test was performed at the 94.827% confidence level, with p-value$= 0.052$. Therefore, we assume that the method applied to **Group 1** is statistically better. Only in question 13 is **Group 2** statistically better. [5]

The pass rate of the five classes remained around the historical values, namely all close to 61%. Hence the two groups do not differ statistically [Zampirolli et al. 2021].

[4]Questionnaire in Portuguese: `https://forms.gle/ak2KLtaA7Nyme3jc7`

[5]For details of statistics: `vision.ufabc.edu.br/MCTest/public/filterNotebook`

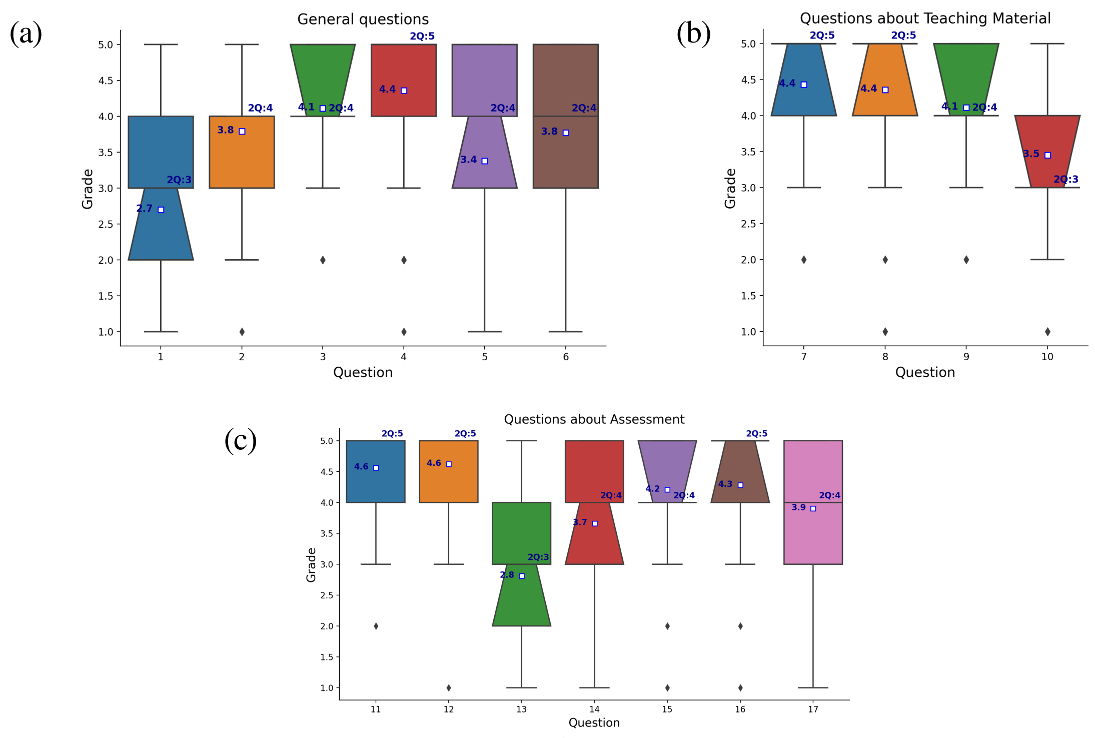


Figure 3. Answers to the questionnaire Student's Feedback separated by question topics: (a) General, (b) Didactic Material and (c) Evaluations.

4.2. Discussion

Besides the previously given technical exposition, one of the professors reported his experience with some morning students, who privately revealed how important our method had become for them. The Covid-19 pandemic compelled them to have a job besides studying, as evening students typically do. Particularly, these morning students now urgently need to learn PLs, which vary from company to company.

We have no knowledge of how many students face this challenge, but one of the greatest advantages of our method is precisely the participant's freedom to choose their preferred PL for the same course. Even after overcoming the pandemic, which may spare such students from working, that advantage will remain because students will be able to quickly revise the course in any PL required in a future job.

Namely, before the pandemic each whole class used the same PL during a course, but the evening students were losing the chance of profiting the time when another PL was required in their jobs. That was an unfortunate drawback, because workplaces vary a lot in terms of their needs and speciality.

5. Conclusions and Future Work

In the paper we presented a method to generate didactic material in many PLs by converting JSON files from Colab/Jupyter into many different formats. In the first line of each text/code cell one inserts a tag that describes which PLs are to be used. This method was applied to five classes of PI with a total of 223 students that underwent the same weekly individualized evaluations that were corrected automatically via MCTest+Moodle+VPL, but with a distinction between two groups: **1** with synchronous lectures on Colab and explanatory short video for the submitted codes, and **2** asynchronous lectures forfeiting the video. Didactic material and evaluations were supplied in six PLs: Python, Java, JavaScript, C, C++ and R, at the student's choice. In this experiment all five classes opted mostly for Python. In order to validate the method we sent the students a questionnaire that in average approved it for 83% (without questions 1, 5, 6 and 13). Regarding didactic material and evaluations in many PLs, 88% (Question 7 in Figure 3b) considered them satisfactory. Didactic material in many PLs and evaluations with automatic correction attained the best average, namely 92% (Questions 11 and 12 in Figure 3c).

In future work we plan that professors who prefer certain PLs should produce and publish more explanatory videos of the whole content of the course. Hence we would have a collection of recorded theoretical and practical lectures in other PLs besides Python. In this way, the student could choose not only a PL but also the corresponding lectures. In our experiment most of the videos deal with Python because of the students' choice.

Moreover, our experiment revealed that we need to produce more didactic material compatible with the level of difficulty faced by the students during the evaluations.

References

Christian, M. and Trivedi, B. (2016). A comparison of existing tools for evaluation of programming exercises. In *ICTCS*, New York, NY, USA. Ass. for Comp. Machinery.

Dingle, A. and Zander, C. (2000). Assessing the ripple effect of cs1 language choice. *Consortium for Computing Sciences in Colleges*, 16(2):85–93.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Jw, T. (1977). Exploratory data analysis. *Reading: Addison-Wesley*.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.

Lowry, R. (2014). Concepts and applications of inferential statistics. `http://vassarstats.net/textbook`. [Online; accessed 19-July-2021].

Neves, R. and Zampirolli, F. (2017). *Processando a Informação: um livro prático de programação independente de linguagem*. EDUFABC.

Rodríguez del Pino, J. and et al. (2010). VPL: laboratorio virtual de programación para moodle. In *Jornadas de Enseñanza Universitaria de la Informática*, pages 429–435.

Rodríguez-del Pino, J. and et al. (2012). A virtual programming lab for moodle with automatic assessment and anti-plagiarism features. *CSCE*.

Sorva, J., Karavirta, V., and Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.*, 13(4).

Thiébaut, D. (2015). Automatic evaluation of computer programs using moodle's virtual programming lab (vpl) plug-in. *J. Comput. Sci. Coll.*, 30(6):145–151.

Wainer, J. and Xavier, E. (2018). A controlled experiment on python vs c for an introductory programming course: Students' outcomes. *ACM Trans. on Comp. Edu.*, 18:1–16.

Zampirolli, F., Borovina Josko, J., Venero, M., Kobayashi, G., Fraga, F., Goya, D., and Savegnago, H. (2021). An experience of automated assessment in a large-scale introduction programming course. *Computer Applications in Engineering Education*.

Zampirolli, F., Pisani, P., Josko, J., Kobayashi, G., Fraga, F., Goya, D., and Savegnago, H. (2020). Parameterized and automated assessment on an introductory programming course. In *Anais do XXXI SBIE*, pages 1573–1582. SBC.