E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Título del Trabajo Fin de Grado

Grado en Ingeniería Informática

Trabajo Fin de Grado

Student: Miguel Lucas Lacal

Supervisor: Mikel Galar Idoate

Pamplona, 11/06/2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Contents

ABSTRACT

The learning process is based on different elements like instructors, learners, the learning method or the learning place. Due to its impact in society, teaching has been deeply studied in order to improve its results. In this project we have focused on improving creating and correction of beginner programming assignments.

We have developed a program which is integrated with Jupyter Notebook. Jupyter Notebook is a web-based tool which allows users to create open source software in different programming languages (Julia, Python and R). This platform is used in many universities because it is simple to use, and beginners can start coding without dealing with compilers or IDEs' installations.

Our platform consists of 2 NbExtensions[1] (instructor and student extension) which behave as interfaces for users, and the main instructor program, which generates the corrector engine for assignments. The corrector engine contains the solutions provided by the instructor, the values of the variables used to test each exercise and an optional tip to help students in case they need it. Correction is based on solutions provided by both instructors and students. If an exercise is not correct students will see some feedback.

After correcting the assignment, students must hand the generated result file in order to be evaluated. Instructors may see the results ordered by assignment id. A simple plagiarism check is executed by the instructor program.

Keywords

Automated correction, programming exercises, computer science, teaching, python.

[1] NbExtension: plugin for JupyterNotebook that adds functionality to the notebook

I. INTRODUCTION

Teaching and didactics have formed an essential pillar in every society. It is not possible to imagine human being evolution without the elementary learning that precedes a task. This learning process is formed by:

- The student or learner
- The instructor
- The learning or teaching method
- The place where learning takes place

These are the four key components when teaching and learning and due to their vast impact on society they have been deeply studied. In this project we have focused on exercise correction and feedback retrieval since both are essential in student learning.

The correcting process involves evaluation and feedback retrieval. Evaluation might be pretty fast to perform depending on the exercise type but what increases the time spent when correcting is feedback retrieval. Feedback retrieval is critical since students that are not provided with useful feedback have little chance to improve.

We have focused on beginner programming exercises because full-automated correction for complex programs is at the moment difficult if not impossible to implement.

Main objective

Fair evaluation and useful feedback are essential in the learning process and that is why our objective is to develop a platform which supports assignment creation, automated evaluation, automated feedback retrieval and a results interface for instructors. This platform is integrated in Jupyter Notebook as most of the engineering students in UPNA learn programming through this tool.

Individual objectives

I. Study existing methods and tools for auto-grading
II. Design and implementation of single exercise correction
III. Design and implementation of main corrector
IV. Design and implementation of professor interface
V. Design and implementation of student interface
VI. Feedback for the student
VII. Results interface

Individual objectives will be explained in the next chapter.

II. THE LEARNING PROCESS

A. Learning process

The learning process' quality depends on four main components: students or learners, instructors, the learning method and the learning place.

Students

Student's importance in the education process varies depending on the learning method.

There are systems in which communication is mostly unidirectional. In this case, students must be very self-sufficient in order to learn and this may result into frustration and loss of interest.

On the other hand, systems based on interaction encourage students to communicate and ask in order to solve problems. Interaction leads to pro-active students, a key feature in today's labour market.

Instructors

Instructors are the ones who guide the learning process and therefore have a huge impact in students' improvement. This is why they are believed to be a key element in the learning process. Nevertheless, the significance of the other three elements must not be underestimated.

The learning method

The learning method and the way it is carried out undeniably affects the learning process. Some of the most common teaching methods are lecturing, demonstrating and collaborating [1].

Lecturing

Lecturing is the most extended teaching method due to its efficiency when it comes to teaching large classrooms. Nonetheless, it is often troublesome because of the lack of interaction. Instructors must make a big effort to notice student's problems and encourage them to give verbal feedback.

Demonstrating

Demonstrating consists of teaching through experiments. It is sometimes easier to understand a fact after seeing an experiment and its associated reasoning.

Being lectures the most popular teaching method, demonstrating seems engaging as lecture drabness is removed. Nevertheless, it is sometimes inefficient when individual assistance is needed or when facing time constraints as experiments may take longer than lectures.

Collaborating

Teaching through collaboration means organizing working groups and making students work together to achieve solutions. This creates links among students and between students and the subject.

The main drawback of collaborating is the time it consumes. Even though, collaborating constitutes a solid teaching method as students learn teamwork and communication skills.

The place

The last essential element in teaching and studying is the place where these activities take place. It must be a suitable place for studying where students and instructors feel comfortable. Although this is strongly linked to the student personal preferences, the most appropriate places to learn are normally quiet and calmed places in order to avoid distractions.

Leading countries in education

Some of the leading countries in education are Japan, Finland, South Korea, Denmark, Hong Kong and Germany. Many studies consider these countries to have the best educational systems according to certain factors like the autonomy teachers have, government founding and teacher training[2][3].

The study of these four elements conclude that any educational instrument or tool should promote communication and feedback trade off.

B. Feedback

Efficient learning methods normally involve active feedback. Feedback must exist right from the beginning of the learning process. When students complete tasks it is very important to let them know how they are doing, not only correcting their mistakes but also remarking their success.

Most former learning methods consisted of sending tasks to students and correcting them. In this situation, learners only see an overall feedback. Lack of specific feedback prevents students from learning from mistakes and is very unhelpful in the learning process. Even though this method allows great students to pass it does not involve real improvement as students cannot know exactly what to improve.

Some of the current teaching methods focus on highlighting mistakes. This may cause frustration in the student because mistakes stand out over everything else. It is true that learners may improve by checking their errors but focusing on mistakes might turn learning into a very depressing and unattractive activity.

Focusing on mistakes when retrieving feedback is called negative feedback and remarking success is known as positive feedback. Many of the leading countries in education support positive feedback, which consists of letting students know their successes. The more fulfilled students feel, the more hard-working they will be.

Learners must know whether they are doing good or bad work, highlighting their mistakes in order to let the know how to improve at the same time positive feedback is retrieved to keep them motivated [4].

Another important aspect of feedback we must consider is how often it is given. There is no point on giving feedback at the end of the course, but overwhelming students with constant feedback might be counterproductive.

Feedback must be continuous in time so that students know periodically how their work is being evaluated. Tasks should be evaluated once students have finished them in order not to overwhelm them. An interesting approach is letting learners hand the assessment in again once it has been corrected, allowing students not only to check their mistakes, but also to fix them.

Apart from being able to correct tasks, students should be given the possibility to ask the instructor for advice. In these cases, the instructor must have a deep knowledge of the task in order to provide concrete support and be able to guide the student towards the solution.

C. Correction and its costs

The learning process involves several different processes. Among those processes, we have focused on correction and feedback.

Teachers spend large amounts of time correcting tasks, explaining how to solve problems and retrieving feedback to students. This time is absolutely needed in order to maintain a high-quality teaching system.

There are many ways to help students on top of lectures. Learners are generally allowed to ask questions during lectures, but this is sometimes not enough when it comes to problems which require some more time.

A different approach is personal assessment. Professors devote some time to solve students' individual questions and despite this not being very popular it constitutes a great way to get through difficult courses.

With the advent of technology individual assessment has grown significantly. Professors save time retrieving feedback by using web platforms and students greatly benefit from this, receiving fast and accurate responses.

Despite the fact that technology improves former correction-feedback systems, instructors still invest large amounts of time correcting and reviewing students' assessments. It is true that little can be done when it comes to proofreading large historic texts, but several approaches have been conducted regarding practical tasks and assignments.

D. Correction in programming

One of the main characteristics regarding programming tasks is freedom when it comes to coding. This generates two types of correction:

- Functional correction
- Non-functional correction

Functional correction evaluates whether the program does its work or not, regardless time constraints or code clearness.

Non-functional correction considers features like time constraints, redundant blocks of code, code readability or efficient resource use.

Instructors must decide the weights of these two aspects in programming correction. Since beginner programming exercises may not require much non-functional correction in this project we have focused on the functional side of programming.

Advanced problems require a substantial amount of students' work and so it is for the instructor to evaluate them. These problems generally involve several days of work and require a specialised instructor to check them.

On the other hand, beginner exercises are mostly simple programs designed to teach students some of the elementary programming concepts. It is true that these simple exercises are easy to solve and therefore to correct, yet there is normally a great number of them.

When it comes to correcting large quantities of these short exercises the professor may end fed up of it due to this task's drabness. Therefore, some approaches have progressively emerged for the purpose of avoiding this.

E. Auto-correction

Auto-correction is the main approach when it concerns saving time while grading. Luckily, this is what computer science studies: how to automate repetitive processes. Therefore, many research groups have developed tools and platforms in order to deal with this issue. Some of these approaches are addressed and explained in chapter three. UPNA professors have also tried some of these approaches to save time and improve their performance as instructors.

Nbgrader is probably one of the most extended tools. Nbgrader is a Jupyter Notebook based tool in which professors can generate different types of tasks for students using a simple interface. The correction of these tasks is a test-based procedure in which values returned by a function are evaluated. After completing the assignment, individual assignments are collected by the professor to visualize the results. Nbgrader's functionality is explained in detail in chapter three.

Nbgrader's principal drawback is the insufficient correction scope. As it is a test-based system, instructors cannot do more than checking function return values via the assert python instruction. For this reason, exercises which consist of different tasks like drawing geometrical forms with characters, which are common among beginner exercises, cannot be auto graded.

A useful online tool for beginners to learn is Repl.it. It is an online IDE that accepts more than 40 different programming languages. It is useful when it comes to simple programs since it is online, and it does not require any installation. Beginners benefit from Repl.it because they can test their programs easily without worrying about compiling or IDE installation.

Nbgrader is a useful tool when it comes to solving value-based exercises [2]and repl.it makes learning efficient. Even though there is a wide variety of exercises that must be excluded due to Nbgrader's nature and repl.it does not provide an automatic corrector.

[2] Value-based exercises: programming exercises which consist of returning a determined output (generally by printing something in the console)

F. Motivation

Lecturers invest large amounts of time correcting and evaluating assignments in order to retrieve specific feedback to students. Sometimes feedback is not even retrieved due to time constraints and learners only receive their mark. Besides, automatic grading and evaluation tools do not embrace a very wide variety of programming exercises.

This project's motivation arises from the need to automate correction and feedback retrieval of a wider variety of programming exercises for beginners.

G. Objetive

The objective of this project is to develop a platform based on Jupyter Notebooks through which instructors will save time when creating and correcting beginner programming assignments.

Individual objectives

Study existing methods and tools for auto-grading

The first step is to have a look at how this issue is being approached. Many tools have been developed by research groups and students, but these tools are adapted to their particular workflow, which means that these software solutions might not be useful in different situations. Some companies have also developed tools which are more adaptable but most of them are not free. Studied solutions are explained in chapter four.

All these software solutions may be classified into the following categories:

- Plagiarism check
- Syntax error correction
- Basic programming concepts learning
- Specific code piece analysis
- Redundant code detection

Design and implementation of a single exercise corrector

After studying some tools related to programming exercises correction it was time to figure out how could individual exercises be corrected and evaluated. First, we tried to correct value-based exercises in order to figure out how to determine the correctness of an exercise. Most beginner exercises require the learner to print something in the screen, but the main problem was that programming exercises' code can be written in many correct ways.

After some thinking we decided that the best way to determine a program correctness was checking the program's output as most of the beginner's programs require the learner to print a result.

With this in mind, we proceeded to correct different exercise types and after succeeding in most of them, we defined output-check as the method to determine correctness.

Design and implementation of the main corrector

The core functionality was to develop a corrector from the task created by the instructor so a solution must be supplied for each exercise. Then students supply their own solutions. These solutions are computer programs, so they are compared by running each solution in parallel and checking whether the outputs are the same or not.

We developed a program which executed python programs in parallel so that the output could be compared. These programs existed before the corrector was launched because the instructor's interface to provide the solutions was no developed yet.

Having developed a functional corrector, we discussed the platform architecture. Architecture features are explained in chapter five.

Design and implementation of the professor interface

In order to create a useful platform intuitive interfaces must be created. As UPNA professors were used to using Jupyter Notebooks, a Notebook-based interface was the best option. We developed an interface through which instructors could write the statement of the exercise in a Notebook cell and below a solution to that statement. To execute the solution more information was still needed so instructors had also to indicate the type of each variable and the values they might get. All in all, an exercise required three different cells in order to be defined:

- Statement cell
- Solution cell
- Variables cell

These cell types were identified by their first line, which indicated the type and the exercise number they were referring to. After defining these cell types, we decided to add 3 new ones:

- Function solution cell
- Function variable cell
- Tip cell

The first two represent the same as the solution cell and variable cell but for functions. We had to define them because of technical reasons. The tip cell was used for the professor to provide some tips in case the student found problems solving the exercises. Precise examples will be shown later.

After creating the corresponding cells for the assignment, instructors could create the assignment for students by clicking a button. This button was part of a Notebook extension.

Design and implementation of student interface

Alike instructors, students are also used to Notebook-based interfaces. Learners obtained a directory from the instructor, which contained the assignment to be completed (statements) and the correction engine.

When learners completed the task, they had to click the correction button in order to obtain feedback. Furthermore, an encrypted file was generated. This file was what the students were supposed to hand in for the purpose of being evaluated by the instructor.

Feedback for the student

Feedback retrieval was since the beginning one of the main issues. When completing tasks, students must know how they are doing to let them improve. Feedback was retrieved for each exercise, indicating whether the solution was valid and the tip in case the solution was wrong.

This way the student could work on the solution at the same time they tested it. This is very helpful because instructors saved time, allowing them to focus on students' questions which required bigger explanation.

Results interface

We developed the results interface as part of the instructor's interface. It allowed instructors to visualize students' submitted tasks for the purpose of analysing them. Results were shown in professor's interface inside a Jupyter Notebook.

Tasks must be collected and placed in the corresponding directory in order to make them visible for the application and they were ordered by task identification and student identification.

The complete workflow is elaborated in chapter six.

III. EXISTING SOLUTIONS ANALYSIS

Many solutions have been developed by research groups all over the world. We have gathered and analysed some of the most interesting software solutions related to automated programming exercises correction.

A. Automatic Analysis of Programming Assignments [5]

This solution was developed by Computer Engineering Students in FernUniversität Hagen, Germany.

At(x) - analyse and test for a language x - is a system used to analyse students' programs and retrieve feedback. This tool has been developed for two programming languages: Scheme and Prolog, At(S) and At(P) respectively.

They use a website platform (WebAssign) to support assignment submission. Students can access the tasks and upload them for correction via WebAssign.

This platform has two upload modes:

- Pre-test mode: automatic analyses/test are carried out to retrieve feedback. After feedback retrieval, students can modify, correct and resubmit their solution.
- Final assessment mode: for students to submit the final solution.

WebAssign supports simple task correction via tests, but for complex tasks specific correction modules are needed. At(x) aims to serve as an automatic test and analysis facility in pre-test mode.

Instances of At(x) have a database to store tasks and students' information in order to carry out correction and feedback processes. At(x) has 2 main components:

- The analysis components, that are implemented in the target languages (Prolog and Scheme).
- A Java interface which connects WebAssign and the analysis components.

In the end, analysis components test functions and check outputs to determine the correctness of the program. Feedback retrieved is filtered in order to show errors in the most understandable and simple way.

Analysis components

Inputs needed to carry out the analysis can be split into static and dynamic inputs.

The static inputs (for each task) are:

- Task description
- Set of test cases
- Specifications of the program solution
- Reference solution

Dynamic inputs consist of:

- Task identifier
- Student solution

The analysis components vary according to the language used, that is why a Prolog analysis may be very different from a Scheme analysis implementation.

Developing different language implementations requires particular development for each language which is a time-consuming process, and since the available implementations do not cover the programming languages that we use in UPNA, At(x) is not useful for us.

B. Tinker: A Programming by Demonstration System for Beginner Programmers [6]

By Henry Lieberman - Media Laboratory - Massachusetts Institute of Technology

Henry Lieberman's approach to programming learning consist of leading the student through concrete examples in order to explain specific programming ideas.

Tinker is a system which teaches beginner programmers how to write Lisp programs through examples. It is a very powerful and general programming by demonstration system which provide multiple and augmentable examples like conditionals and recursion.

Students must tell Tinker which parts of the examples are important, in order to record them for the following problems. This is the function concept.

Here is a simple example of use:

Problem: draw a triangle

Solution:

- Forward 100
- Right 120
- Forward 100
- Right 120
- Forward 100
- Right120

But if the teacher wanted the student to use procedures the solution should have been:

- To Triangle
- Repeat 3 (Forward 100 Right 120)
- End

At this point students would execute the program, but nothing would happen, because they do not understand that nothing will happen until the procedure is ended and called. This may be quite abstract for a beginner.

Tinker provide the following: since a procedure is a remembered set of actions, there is a command that remembers a concrete set of actions and gives it a name so that when that name is written, the set of associated actions will be executed just like a procedure does.

This grows in complexity when tackling different programming ideas like recursion or loops, but the idea of programming by demonstration remains the same.

C. Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks[7]

Sahil Bhatia - Netaji Subhas Institute of Technology - Delhi, India

Rishabh Singh - Microsoft Research - Redmond, WA, USA

Lots of techniques for generating automated feedback on programming assignments focus on functional correctness and style features. Unfortunately, it is not possible to generate corrections for student's syntax errors using these methods.

The solution presented by Sahil Bhatia and Rishabh Singh uses Recurrent Neural Networks to model and predict valid token sequences, so that the learnt RNN can determine whether a token sequence is correct and by which piece of code may be replaced in case it is wrong.

The RNN consists of an input layer, a hidden layer and an output layer. The training was carried out by using syntactically correct student submissions to determine how the target syntax is.

This technique has been evaluated using 14000 student submissions with syntax errors, and the RNN repaired more than 30% of them.

D. System for Automated Assistance in Correction of Programming Exercises SAC [8]

Benjamin Auffarth, Maite Lopez-Sanchez, Jordi Campos i Miralles and Anna Puig.
Department of Applied Math and Analysis - University of Barcelona.

SAC is a web-based environment for carrying out unit testing on submitted programs. Instructors must know the **JUnit** tool in order to develop JUnit tests.

This platform focuses on the idea of open source and modularity in order to make the platform more accessible and let SAC work together with other systems, like plagiarism check and style correction systems.

An important limitation of this solution is that students must submit programs which must not fail during execution and without compiling errors.

Some other characteristics are that solutions are uploaded in HTML forms and information about assessments can be exported to .csv or .xslx to generate statistics.

E. Automated semantic grading of programs [9]

Rishabh Singh – MIT CSAIL – Cambridge, MA

Sumit Gulwani – Microsoft Research – Redmond, WA

Armando Solar-Lezama. MIT CSAIL, Cambridge, MA

This approach remarks the main differences between automated semantic grading and debugging techniques.

When correcting programs semantically the complete specification is known, whereas in debugging the space of solutions is extremely large. Semantic mistakes made by students are predictable and programs are generally short. Therefore, the automated grading process is simplified.

This tool has several modules that work together to achieve the solution. Firstly, instructor provide a reference solution to let the system know the problem. The system includes an error model language (EML) which allows instructors to describe a very high-level description of the possible errors the student can make. Now the instructor describes the errors in EML.

Then the EML file and the reference solution are used to rewrite the program in an imperative language called IMP to explain the details of the algorithm. The IMP program includes all possible programs obtained from the reference program and the possible errors.

Afterwards, Sketch uses the rewritten program to synthesize the space of possible solutions. Sketch basically collects incomplete programs (programs with unspecified fragments or holes) and fills them automatically based on a reference implementation (given by the instructor), using the CEGIS algorithm for efficiency.

CEGIS (counterexample-guided inductive synthesis) receives a specification, synthesises candidate programs and verifies them against the specification to then provide feedback for future candidates.

After that, the synthesizer finds a solution by modifying the input program until it is correct. Each correction increases a counter used to retrieve the mark of the program.

Workflow example:

Reference program

```
1 using System;
3 public class Program {
4     public static int[] Puzzle(int[] b) {
5           for (int i=0; i <= b.Length/2; i++){
6                 int temp = b[i];
7           b[i] = b[b.Length-i-1];
8                 b[b.Length-i-1] = temp;
9           }
10          return b;
11    }
12 }
```

EML file:

```
v[a]     → v[a – 1]
a0 > a1  → a0 < a1
a + +    → a – –
```

Students program

```
1 using System;
3 public class Program {
4     public static int[] Puzzle(int[] b) {
5           int front, back, temp;
6           front = 0;
7           back = b.Length-1;
8            temp = b[back];
10          while (front > back){
11                b[back] = b[front];
12                 b[front] = temp;
13                back++;
14                front++;
15                temp = b[back];
16          }
17          return b;
18     }
19 }
```

Feedback

```
1. Change the conditional (front > back) in line 10 to (front < back).
2. Change the increment back++ in line 13 to back--.
```

F. Nbgrader[10]

This tool is based on Jupyter Notebooks and presents a very intuitive way of creating and correcting assignments. It features several ways to evaluate exercises. Nbgrader is completely integrated in JupyterNotebook IDE cells.

The instructor creates the assignment by clicking a button and fills each cell in a specific way depending on the cell type. Only autograded answers accept automated correcting.

Code between "### BEGIN SOLUTION" and "### END SOLUTION" will not appear in the student's assignment. Students must implement their solution between these marks.
Figures 1 and 2 show how instructors provide a solution between these marks, but students do not see it.

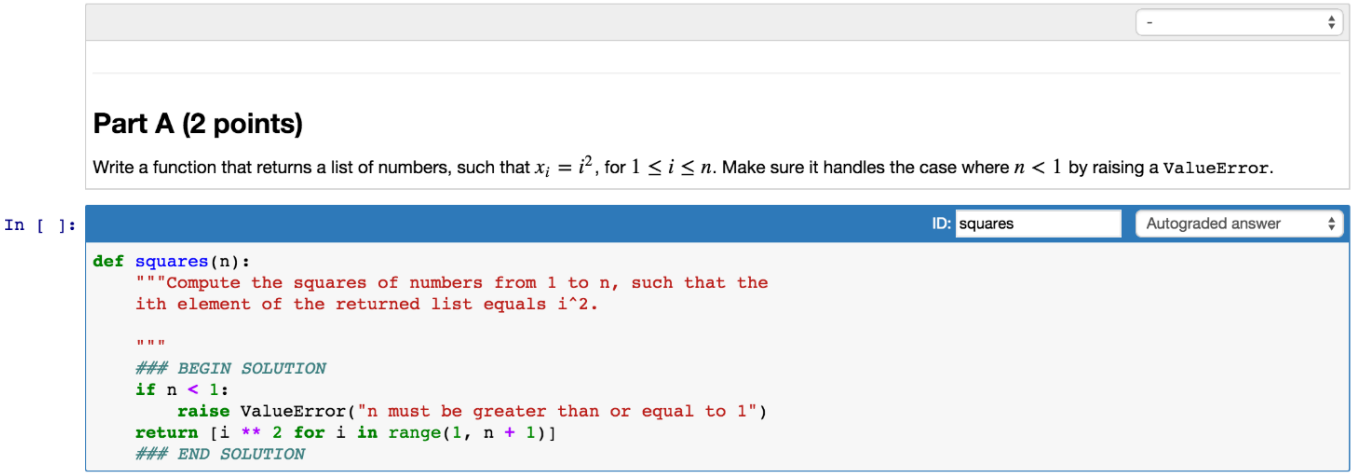

*Figure 1. Example of assessment exercise in **instructor's** interface.*

```
def squares(n):
    """Compute the squares from 1 to n, such that the ith element of the returned list equals i^2.
    """
    ### BEGIN SOLUTION
    ### END SOLUTION
```

*Figure 2. Example of assessment exercise for **students**.*

As shown in Figure 1, there is a select box used to select the cell's type. Each cell type involves a specific grading process.

- **Manually graded answer**: answers which are difficult to correct and require human revision. A statement example could be using Latex notation, write down the equation of the sum of squares given a sequence of integers.

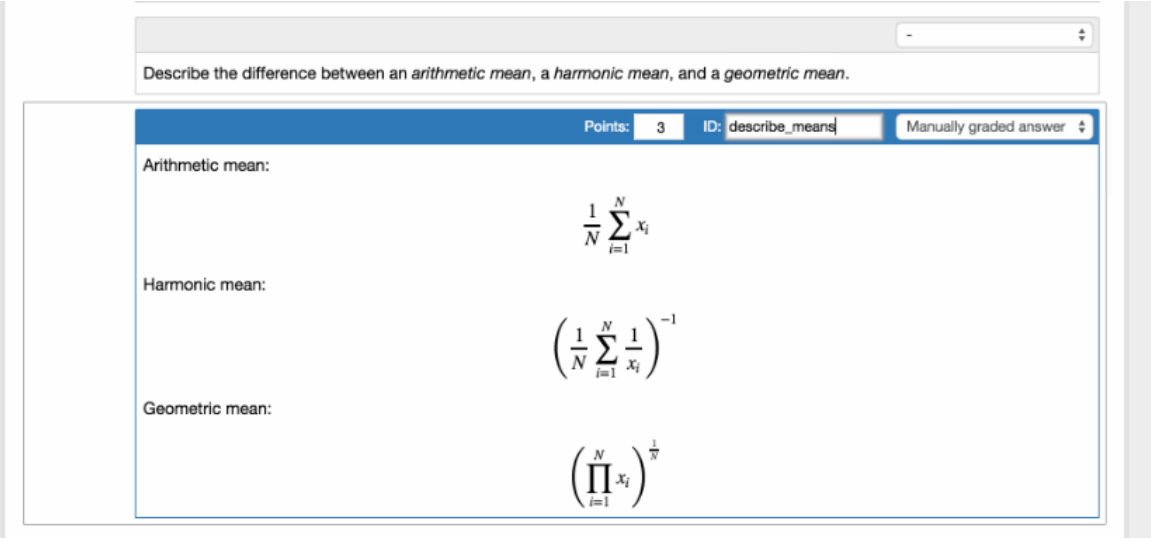

Figure 3. Manually graded answer example

- **Autograded answer**: for tasks which will be graded automatically. For instance: define a function to calculate the sum of squares of a given list of integers. The grading is carried out by executing a set of tests.

- **Autograded tests**: set of tests that will be carried out to grade an autograded answer. These cells are mainly composed of assertions. The test can be set to be visible of invisible for the student.

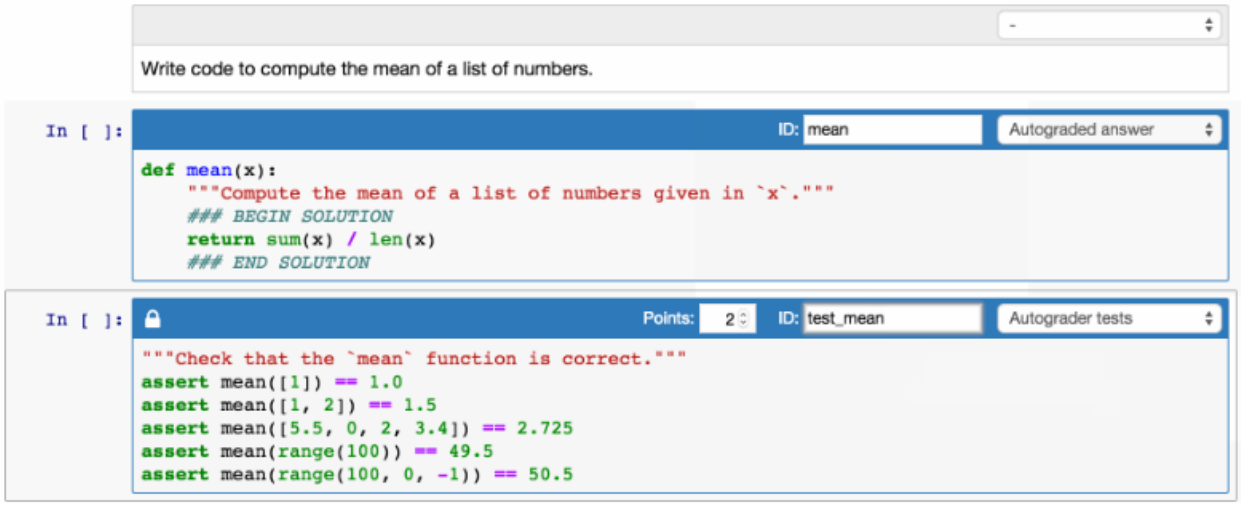

Figure 4. Autograded answer and tests example.

- **Read only cells**: text cells above each code cell. Students cannot modify them as they are exercise statements.

The created assignment is stored in a local database, with its name and identification.

The NbGrader platform allows instructors to register their students and send them the desired assignments (students may belong to different groups or courses).

The submission process can be done manually: sending manually each task file and collecting them manually to then carry out the correction in the instructors Jupyter Notebook. But it can also be done automatically, in a client-server way.

After assignment submission, instructors proceed to correct manually graded tasks (autograded answers are automatically graded).

Comparison table

We constructed a comparison table in order to study and compare the most important features of each tool.

Columns:

- **Language**: programming language the tool works with.
- **Platform**: type of platform the tool uses. If it is empty the platform is unspecified.
- **Type**: correction or grading method.
- **Grading & Feedback**: whether the tool provides grading or feedback functionality.
- **Instructor action**: work instructors must do in order to use the tool.
- **Student action**: work students must do in order to use the tool. If a solution has 'No' in this column it means that students are only required to provide the solution.

	Language	*Platform*	*Type*	*Grading & feedback*	*Instructor action*	*Student action*
At(x)	Prolog, Scheme (Any – requires implementation)	Web	Tests	Yes	Test building	No
Tinker	Lisp	-	Learning through examples	No	No	Coding
RNN	Any	-	Neural network	No	Set of inputs	No
Semantic	Any (requires implementation)	-	Critical pieces of code	Only feedback	Reference program + Rules	No
SAC	Java	-	Tests	No	Test building	No
NbGrader	Python (Any – requires implementation)	Web (Jupyter Notebook)	Tests	Yes	Test building	No

Table 1. Solution analysis characteristics comparison table.

At(x) offers an interesting test-based functionality. However, we find problems when it comes to facing exercises which are not value-based exercises. Instructors still have to correct manually those exercises, which involve abstract tasks, such as figure drawing exercises. At(x) allows developing modules to approach those exercises but implementing new modules may be time-consuming, especially because At(x)'s technology and implementation is new to us.

At(x) and Nbgrader share the correction method as they are both test-based correcting systems. The principal difference is that Nbgrader is completely integrated with Jupyter Notebook. Therefore, Nbgrader is straightforward to include in most of professors' methodology.

As we have already said, Nbgrader's principal drawback is the lack of variety in corrections. Thus, automated correction for abstract exercises is needed.

SAC does not bring anything special as it is a test-based autocorrection system through JUnit. In the end JUnit brings the same functionality as NbGrader or At(x).

Tinker implements a system to help instructors teach elemental programming concepts. Notwithstanding being a useful complement for professors, it does not fulfil the need of automated exercise correction.

Artificial intelligence is always a possibility when dealing with large repetitive issues. Hence Recurrent Neural Networks solve the tedious task of detecting syntax errors. It is true that most compilers retrieve some feedback wherever syntax errors appear, but this system is able to not only to detect them but also to correct several syntax errors. Nonetheless, syntax error detection and correction has little to do with automated programming exercise correction.

Among the studied solutions, the most appealing one is **Automated semantic grading of programs.** This software solution analyses instructor and student's solutions, finding possible errors and focusing on those pieces of code. Afterwards, it grades the program regarding how many changes have been needed to make the student solution work. Despite being such an interesting tool, the quantity of modules required is overwhelming. Although complexity means being able to tackle a larger variety of exercises, some of these modules are not available and developing substitutes would be very time-consuming.

After conducting an analysis about some of the most known solutions related to automated programming exercises correction, we conclude that despite some of them being really interesting and useful, none of them fulfils our interests completely. Our goal is to develop an easy to use platform for teachers and students for the purpose of accelerating and improving exercise correction and feedback retrieval.

Different grading tools

Most of the above mentioned software solutions focus on the functional side of programming. We also want to consider different aspects which may be useful for grading.

Plagiarism check

Regarding the grading objective, plagiarism is an interesting issue to take into account. There are lots of tools in the internet because plagiarism is unfortunately very spread.

Unicheck[11]
This tool was firstly thought to check plagiarism in text like essays or papers but given its success among the users it has expanded and now it also detects plagiarism in programming code.

It checks in web pages and educational data bases and provides very complete reports about the results.

Unicheck is the most used plagiarism check tool but, unfortunately, it is not free.

CloneDR[12]
Developed by Semantic Designs, its objective is to reduce the spread of bugs in duplicated code. It has become quite renowned because programmers normally reuse and copy code to avoid implementing the same functionality again.

It is a very complete tool which analyses exhaustively whole systems and finds both exact matches and partial cloned pieces of code. It also supports large program comparison utilizing parallel computing and it is integrable in Eclipse.
Available languages are Java, C#, C++, COBOL, JavaScript and PHP.

MOSS[13]
MOSS (Measure Of Software Similarity) is a tool for detecting similar pieces of code. Instructors must then decide whether those pieces of code are plagiarism or not.
It has been used over the years by educational institutions, because it is free for this kind of entities. The use of MOSS for commercial purposes is restricted.
It supports a wide variety of programming languages: C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN ...

Style-Check

Coding standards are especially important when developing large systems. One of the most interesting tools which tackles programming style is Checkstyle[14]. Checkstyle is a Java tool that ensures programs are developed following a coding standard. This Way, programs are more legible and understandable, which accelerates the grading task.
Despite the importance of style in large systems, programs we are working with (beginner programs) do not exceed 15-20 lines. Therefore, we will not go deep in this matter.

Performance testing

With performance testing we have a similar case to Style-check in Style-Check: beginner programs are simple and do not take advantage of coding efficiency. Since performance is not an issue beginners should take into account, we will go ahead without giving weight to this aspect.

IV. TOOLS & TECHENOLOGY

Along the development we have used several tools in order to accomplish the objectives mentioned in chapter two.

A. Jupyter Notebooks

Since this platform is based in Jupyter Notebook, we have made continuous use of this tool. Jupyter Notebook is a web-based programming tool which allows the user to code in different programming languages.

"Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages" - [15]

We based our correction platform in Jupyter Notebooks because it is used in beginner programming courses in UPNA. Therefore, beginner students will not have problems learning how to use our platform as it is integrated in Notebooks.

Notebook files are JSON objects, which are similar to python dictionaries, so reading and modifying their content is straightforward. We have read and modified Notebook contents for the purpose of analysing student and professor solutions and to retrieve feedback to students.

B. NbExtensions

NbExtensions are a useful complement while working with Notebooks and can be easily installed. They are like plugins which bring functionality to Notebooks. Once NbExtensions is installed, extensions may be acquired via an interface inside Jupyter Notebook.

Developers can easily create their own NbExtensions locally to improve their performance but NbExtensions may also be uploaded and published in order to make them accessible for everyone.

We have developed two different NbExtensions which will be explained deeply in chapter six:

- Instructor extension
- Student extension

C. Anaconda

Anaconda is an open source distribution of the languages Python and R, very popular among data science and automated learning developers.

It is the package distribution system we have used to get Jupyter Notebooks and Python.

D. NbGrader

NbGrader is a NbExtension developed for the purpose of creating and grading assignments. It is a very popular tool among educational institutions because it provides useful functionality for grading.

In UPNA it is used in some courses to evaluate value-based exercises because correcting large batches of these exercises is time-consuming. Although having demonstrated its usefulness there are some problems which cannot be handled by NbGrader.

We used NbGrader to figure out how was correcting being conducted and understand which characteristics might be improved.

E. GitHub

In order to keep a coherent version control through the platform development we have used GitHub. GitHub is a git version control system used by many research groups, institutions and individual developers.

Along the platform development we have generated different versions, all of them functional. Each version has been developed from its predecessor version, adding functionality to the platform. Therefore, it has been an iterative and incremental development.

F. Programming languages

The platform has different modules written in different languages. We used different languages depending on the requirements of the module and where it has to be run.

JavaScript, HTML & CSS

Students and instructors NbExtensions have been developed in JavaScript, HTML and CSS since Jupyter Notebook is a web platform.

Python & JSON

Python has been the main development language because this way the corrector engine can run in the user's Notebook. This is the most efficient way to implement the corrector engine since the corrector code already has an environment to run in.

An alternative would have been developing an external corrector engine, but this seemed unnecessary having already a Python environment ready to use.

The core functionality, the corrector engine, is a python file. This python file also uses JavaScript, HTML and CSS in order to return feedback to learners.

JSON has also been an essential technology because the principal python data structures, dictionaries, are based on JSON objects. Besides, Jupyter Notebooks files are stored as JSON objects, what makes it straight forward to work with them in Python.

V. AUTOCORRECTOR & ARCHITECTURE

The development of the corrector has been an iterative and incremental development. This means that the aim was to get functional versions of the project based on older ones.

A. Version 0.1

The beginning of the development consisted of evaluating beginner exercises. We used several beginner assessments in order to figure out what were students asked to do. Our aim was to develop a simple corrector engine able to evaluate beginner exercises separately, without considering usability.

The first corrector version read student's solutions, created .py source files for each solution and then executed them comparing the execution output with the expected output. This expected output had already been defined previously to the correction process.

As we have already said, these beginner exercises include value-based exercises and drawing-based exercises. Figure 5 shows an example of a value-based exercise:

```
### STATEMENT 1
##### Make the sum of 2 integers set by the user
```

```
### CODE 1
n = input ('Please, intrduce a number:')
m = input ('Please, intrduce another number:')
print ('The sum of the introduced numbers is :',int(n) + int(m))
```

```
Please, intrduce a number:2
Please, intrduce another number:5
The sum of the introduced numbers is : 7
```

Figure 5. Value-based exercise example and execution.

These exercises are simple and there are a few possible solutions. Nonetheless, value-based exercise correction can be tricky. Although conducting a sum may be straight forward, the output of the program can be written in many ways. The instructor just asked for the sum, so printing "The sum of the introduced numbers is : 7" and printing "Sum: 7" should be equally correct. This is the reason why plain output comparison is not a viable approach.

As a solution we used cipher comparison so the part of the output which is complementary to the solution is ignored and only numbers are considered.

This approach leads to the problematic of retrieving output in an order different than expected. It may happen when students are asked for several solutions. Figure 6 shows an example which asks for more than one value.

```
### STATEMENT 3
#### Ask the user for 2 values: value 1 and value 2. Make first the sum, then
#### the substraction(value 1 Ã· value 2) and finally the multiplication of them.
```

```
### CODE 3
n = float(input("Please, introduce a number:"))
m = float(input("Please, introduce another number:"))
print("The sum is ", n + m ,", subtraction is ", n - m ," and multiplication is ", n * m )
```

```
Please, introduce a number:2
Please, introduce another number:5
The sum is  7.0 , substraction is  -3.0  and multiplication is  10.0
```

Figure 6. Example of multivalued-based exercise.

In the example shown in Figure 6 the instructor has asked for three different values. The corrector engine expects the student to write the solutions in order so that it can compare them with professor solutions. In case the student prints the subtraction before the sum, the corrector engine will consider it as an error and the exercise will not pass the test.

Statements must tell precisely in which order the solutions must appear for the corrector to work.

The other exercise type we have worked with is drawing-based exercises. These exercises are widely used to teach how to use loops. Figure 7 shows an example of drawing-based exercises:

```
### STATEMENT 4
#### Ask the user for a value and build a rectangle triangle with height
#### and base equal to the user's number. Here is an example:
#Introduce a number: 4
#*
#**
#***
#****
```

```
### CODE 4
n = int(input("Introduce a number: "))
for i in range(n):
    for j in range(i+1):
        print("*",end='')
    print()
```

```
Introduce a number: 4
*
**
***
****
```

Figure 7. Drawing-based exercise example

Having figured out how to evaluate value-based exercises we tried to extrapolate it to drawing-based exercises. Student output was captured and compared with the expected output, so the correcting method was basically the same.

Being able to evaluate some simple beginner exercises we concluded the first version of the corrector.

B. Architecture

The next step was to consolidate the architecture of the platform because until now we only had a simple exercise corrector.

Client – server corrector

The first idea was to build a client-server-based corrector. This architecture allows synchronisation between students (client) and instructors (server). This synchronisation would have been used to prevent users from executing the corrector itself, improving client's performance. Furthermore, a client-server architecture would have allowed us to implement some sort of attempt control. This attempt control could have been used to consider the number of attempts students make when fulfilling the assignment.

Despite these advantages, a client-server architecture involves several drawbacks which made it impractical. First, running a server is expensive and involves developing a secure and robust system to keep the service available. Furthermore, internet connexion becomes essential so the platform would become unavailable without it.

After some discussion we excluded this architecture due to its several disadvantages.

Global corrector – V0.2

The next candidate was a global corrector which included every existing assessment corrector. This would have been like the client-server architecture, but without the internet connexion problem or the server availability and security issue.

The global corrector architecture was implemented using python classes and objects. There were a principal corrector class, which had the basic corrector functionality (read student's complete assignment and generate encrypted results file ready to hand in). This encrypted results file would be used by the instructor to get students' results. The core functionality fell to specializations of this main corrector, which had the correction logic of each assessment. Figure 8 shows of the Global Corrector V0.2 architecture.

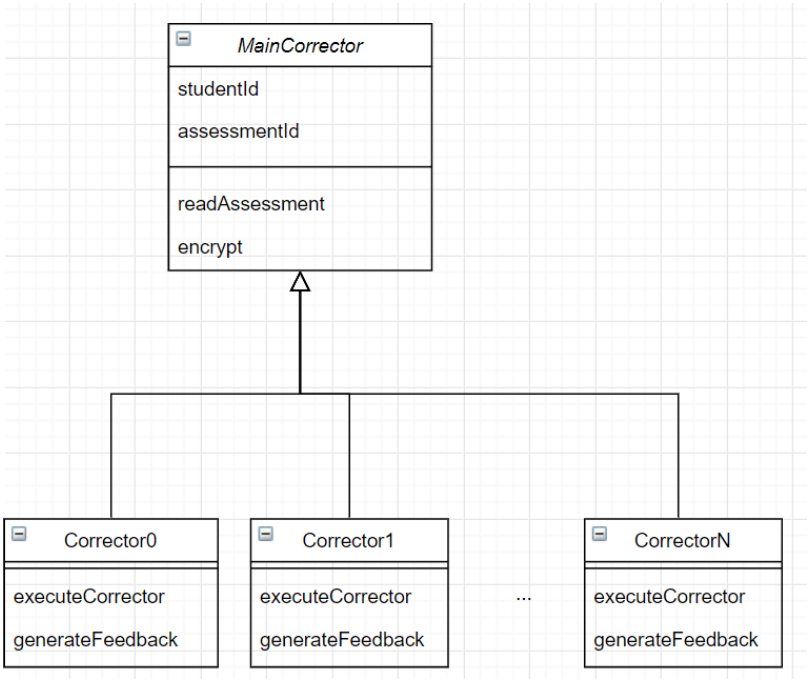

Figure 8. Global corrector architecture

The correcting process had to be executed manually, importing the corrector in the notebook an executing the correction function. After evaluating an assignment, the corrector would generate some feedback showing which exercises were wrong. The corrector would also generate the encrypted results file ready for evaluation.

```
import correctorFile
correctorFile.execute()
```

Figure 9. Notebook cell which imports and executes the corrector and shows exercises feedback.

The corrector generation interface was not developed yet, so the correcting functionality was developed manually. Each exercise was corrected through python code which had been manually written. In the future this correction logic would have been created by instructors through an interface

Nonetheless, the corrector itself would have been rather heavy as it would have included every single assessment correction. Apart from that, this architecture requires solutions to remain static in the correction code. This means that whenever instructors wanted to make any subsequent change they would have been forced to generate and redistribute the whole corrector again.

Therefore, we decided to use a different architecture in order to improve user's experience and corrector performance.

Solution-based corrector – V0.3

The solution-based architecture consists of generating a specific corrector engine for each assignment. This way instructors would generate a corrector engine each time they create an assignment.

The corrector engine is completely based on solutions provided by instructors. These solutions are gathered when the assignment is created, in the instructor's interface. We decided that this was the best possible architecture for the platform since no internet connexion is needed to complete assignments, solutions are created dynamically for each different assignment and the correction engine was not heavy enough to impair students' user experience.

We started developing the instructor's interface since it is the start of the workflow. Instructors create assignments through a Notebook Extension following these steps:

- Add the extension
- Fill assignment cells
- Run the assignment generator

When the assignment is created it can be handed in along with the corrector engine program. Afterwards, students shall complete the assignment and run the corrector in their assignment Notebook. Then students must hand the generated results file in for instructors to get the results. Instructors see the results through the instructor's interface.

Functionality details and main workflow are explained in the following chapter.

VI. NOTEBOOK INTEGRATION & WORKFLOW

We selected the solution-based architecture because it brought some interesting features like a light corrector engine and no need for internet connection. Here we will explain the final platform development, functionality and integration. The platform usage is deeply elaborated in chapter 7.

A. Instructor's Interface

Since the workflow would start by creating an assignment, the first module we developed was the instructor's interface.

Each cell type is identified by its first line. There are 7 cell types:

- Id cell
- Statement cell
- Code cell
- Variable cell
- Function code cell
- Function variable cell
- Tip cell

The id cell is required to identify and create the assignment. Statement cells are used to write the exercise statement, code cells contain the instructor solution to that exercise and variable cells contain information about the possible values each input may get. Function code cells and function variable cells represent the same but for function-based exercises. Tip cells are bind to an exercise and are used to provide some help to students in case they find problems solving the assignment.

Variable cells are essential when evaluating critical pieces of code. The corrector engine will generate values for input variables based on the information written in variable cells. Each line of a variable cell gives information about a single input. The pieces of information of each variable cell line are separated by spaces. The first piece of the line is the variable type (integer, string…). The second piece of line states to which interval the variable belongs. At the moment, only strings and integer variables are supported. The third piece is the interval states those values that cannot be taken by the variable (only for numbers). Variables can also receive static values, instead of ranges. As an exception, variables in function exercises also support dictionary and list static variable definition.

STATEMENT 1

Make the sum of 2 integers entered by the user

```
### CODE CHECKNUMBER 1
n = input ('please introduce a number:')
m = input ('please introduce another number:')

print('The sum of the introduced numbers is: ',int(n) + int(m))
```

```
### VAR 1
1=int -inf:inf 5:8
2->int 4 5
```

```
### TIP 1
To sum integer variables they must be casted to
integers with 'int(<variable>)'
```

Figure 10. Instructor assignment exercise example.

In Figure 10 we can see the statement cell, the code cell, the variable cell and the tip cell. Each cell is identified by its first line.

The code cell's first line includes the modifier 'CHECKNUMBER', which means that the result of the exercise is a number (sum of 2 integers) and that the corrector engine must ignore every non-cipher character. In this way we consider a wider variety of correct solutions since only output numbers are taken into account. If an exercise does not include the 'CHECKNUMBER' modifier the whole output is considered and checked.

The variable cell informs the corrector engine that the first input is an integer, which can take values from minus infinite to plus infinite, excluding the [5,8] interval. The tip cell contains the text that will be shown in case students need help. The second input means static input for that variable ('->' instead of '='). It will receive the values 4 and 5.

Figure 11 and 12 show similar examples but for an exercise which requires a string as input. Figure 11 represent a static input exercise whilst in Figure 12 bounds are given to generate random strings as inputs. The string's length will be within the defined interval (between1 and 10 characters).

STATEMENT 5

Ask the user for a string and build a list

with the caracters of that string. Print the list.

```
### CODE
string = str(input("Enter a string: "))
print([char for char in string])
```

```
### VAR
1=str brownFox myString  boo
```

```
### TIP
Loops might be necessary
```

Figure 11. Instructor assignment exercise example 2

```
### VAR
1=str 1:10
```

Figure 12. Instructor string bounds definition in variable cell.

Instructors can also define function-based exercises. This exercise definition is similar to the ones we described above; the only difference is the identification of the cell. Variable cells are defined the same way and accept the same types.

STATEMENT 6

Create a function which receives 2 integers and returns the sum of them

```
### FUNCTION
def sumNumbers(a,b):
    return a+b
```

```
### VAR FUNCTION
1=int -inf:inf
1=int -inf:inf
```

```
### TIP
Use the 'return' instruction inside the function to return the sum of the parameters
```

Figure 13. Instructor assignment function-based exercise example.

These function exercises also accept different variable types as static inputs. When a static input is defined, type is omitted and the static value (or values) is used to test the function later on, along with students' solutions.

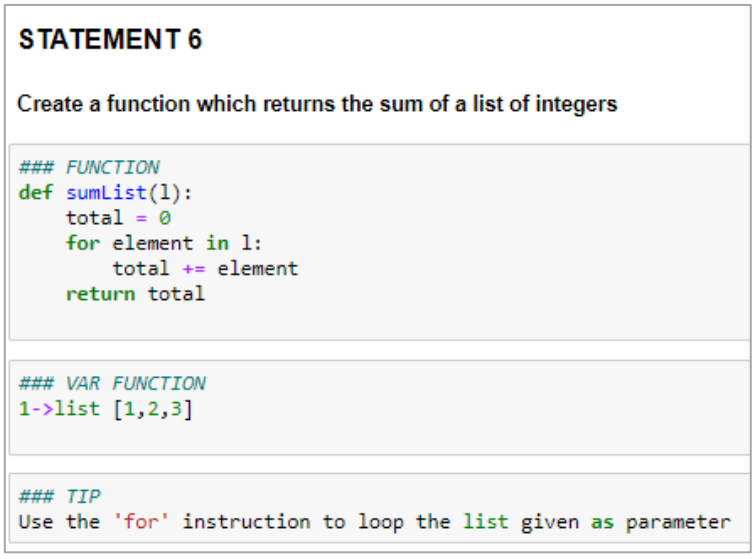

Figure 14. Static list input example.

When defining static inputs (in either code cell or code-function cells) instructors can only define a maximum of 3 values for each variable. The corrector engine conducts 3 tests for each exercise so if only 2 static inputs are defined the first one will be used for the third test. If only one is defined it will be used for every test.

For exercises in which the program asks for specific inputs from the user static inputs must be defined (maximum of three possible inputs). An example would be an exercise that asks the user to choose an option from a menu.

Variable cells in function quads behave similarly to non-function variable cells. Tip cells may be empty. The assignment may have several of these quads.

After creating the assignment instructors must click the generation button given by the instructor's NbExtension. This extension is used by the professor to create assignments for students. It mainly consists of a simple button that when clicked, the assignment content is read and a ready to fill assignment is created. The extension also includes buttons to create statement, code, variable, tip, function code and function variable cells. A compiled python corrector for the assignment is also created.

After the above mentioned, professors just have to hand the created assignment in and the corrector to each student.

Some instructors may not be able to install the extension because they are using Jupyter Notebooks' online version. These users would have to create the assignment cells manually and execute the instructor's program manually. Figure 15 shows the content of the cell which executes the assignment generator.

```
from InstructorProgram.professor import *
genAssignment(<assignmentId>)
```

Figure 15. Assignment generator execution.

Instructors might make mistakes when defining the assignment. To enhance user experience, we have added an input check module to guide them. This input checks are triggered every time instructors click the assignment generation button. Here are some examples:

Figure 16 shows a tip cell with missing id, so the interface warns the user through an error tag. The assignment generation will not conclude if any errors are found.

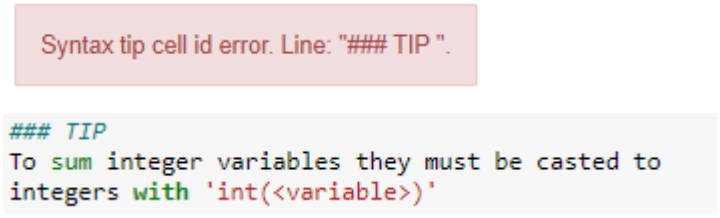

Figure 16. Tip id error.

Figure 17 shows an empty variable cell. It is not possible to generate an assignment if any variable cell is empty.

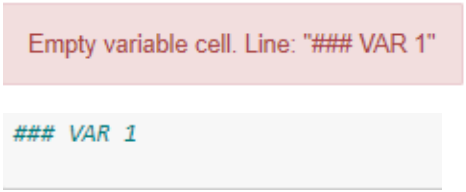

Figure 17. Empty variable cell error.

B. Corrector & assignment generation

The instructor's NbExtension creates a cell at the bottom of the notebook which runs the instructor's program. This program is a python file which generates the ready to fill assignment, the compiled assignment corrector and the JavaScript feedback file from the instructor's assignment.

Instructor's program gathers the assignment content and creates a corrector engine from a model corrector engine. This corrector engine model already contains all the functionality related to exercise correction, feedback generation and encrypted result file generation. The JavaScript feedback file is also generated from a template filled with feedback obtained from the instructor's assignment.

The instructor's program basically adds a line to the corrector program, which contains every solution to the assignment exercises and their corresponding input variable values information. The ready to fill assignment is generated from the statement cells in the instructor's interface.

As we can wee in Figure 18, instructor's interface is composed by the NbExtension and the instructor's program, and generates the files "assignment.ipynb" and "corrector.pyc". The "assignment.ipynb" file is the ready to fill assignment and the "corrector.pyc" file is the python assignment corrector.

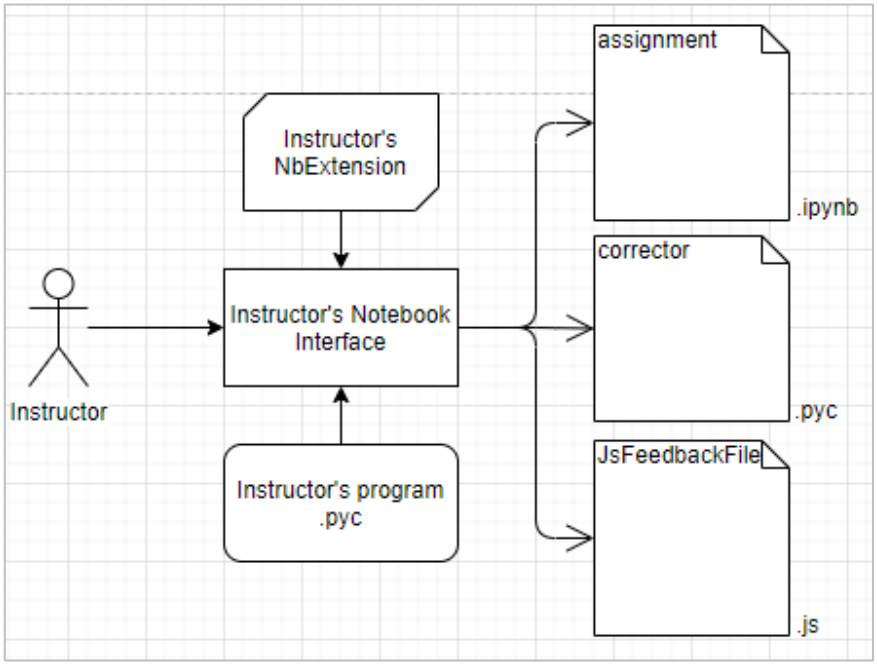

Figure 18. Instructor's interface structure and workflow.

C. Student's interface

Once students possess the ready to fill assignment and the other assignment files (corrector engine and JavaScript feedback file) they only have to complete it and run the correction button. Figure 19 shows how a ready to fill assignment looks like.

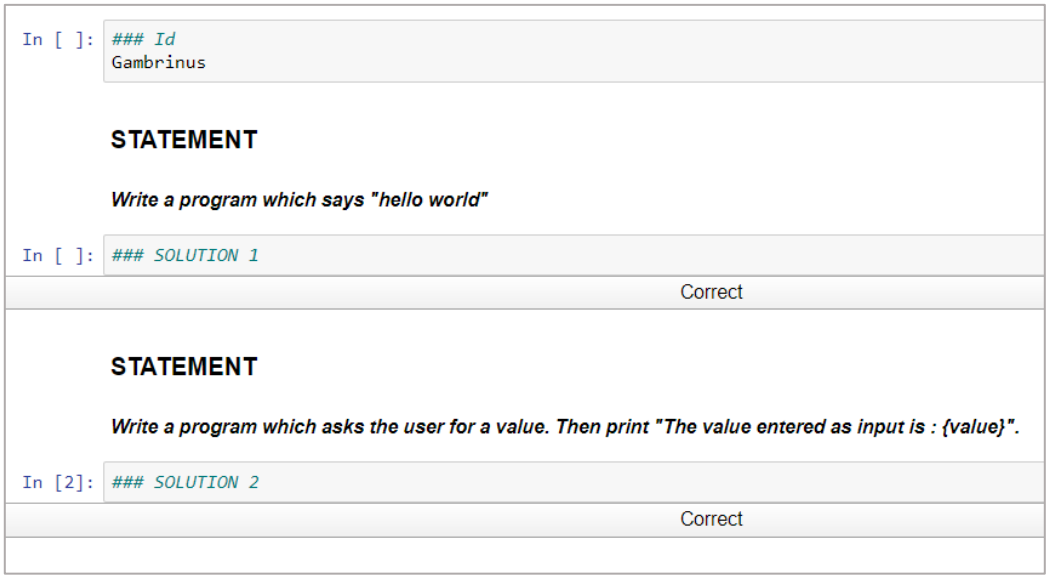

Figure 19. Example of ready to fill assignment. The id cell has already been filled by the student.

The student's extension is more complex as it performs several tasks. Students must have the assignment and the compiled corrector before starting to use the extension.

It mainly consists of a button that works the same way as the instructor extension's button. Once the assignment is complete the button must be clicked, starting the correcting process.

When the correction is finished, students will see in the assignment notebook some feedback related to their exercises. An encrypted file containing students' results is also generated.

Students may also run his solutions as many times as they want since what matters is the solution code. For the same reason, solution cells may not be run. This way students can fix their solutions before handing the assessment results file in. The correction button must be clicked each time any solution is modified in order to update it. Students can also correct their solutions individually using a button provided by the NbExtension, but in this case the result file will not be updated.

Students must then hand this encrypted file in in order to be evaluated.

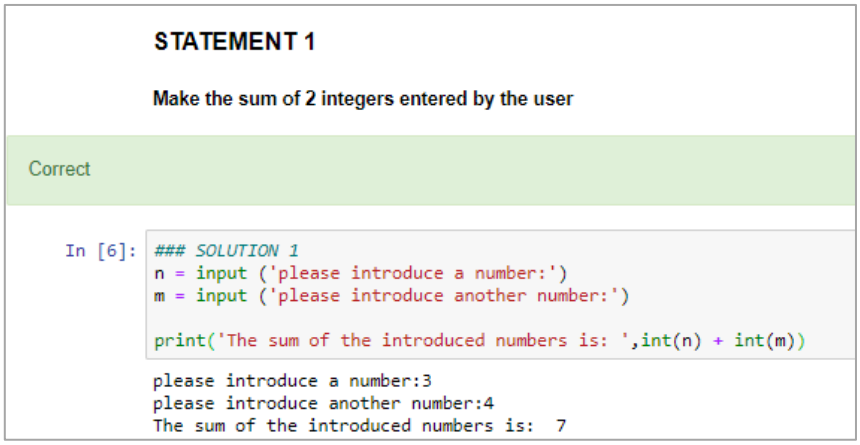

Figure 20. Positive feedback example

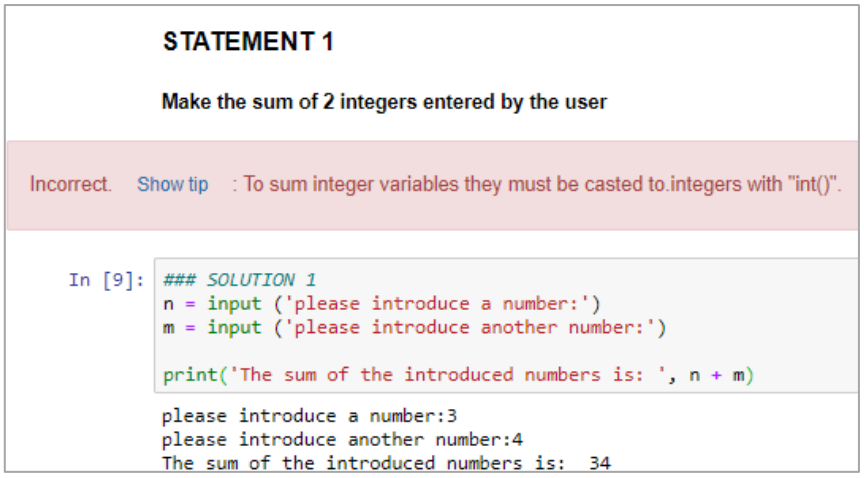

Figure 21. Error feedback example.

Figures 20 and 21 show examples of the feedback received by students when correcting exercises. The tip message is only shown in incorrect exercises.

Some students may not be able to get the NbExtension because they may be using the online Jupyter Notebook version. They can still run the corrector by running the cells shown in Figure 22 and 23.

```
# Single exercise corrector
import correctorFile
correctorFile.execute_single(6)
```

Figure 22. Run correction for exercise 6.

```
# All exercises
import correctorFile
correctorFile.execute()
```

Figure 23. Run correction for the while assignment.

Alike instructors, students may make mistakes when using the interface, so student's input is checked, and feedback is retrieved to help users using the platform.

Figure 24 shows an assignment without student identification. An error tag is showed and assignment correction is stopped. This input checks are triggered every time students click the correction buttons.

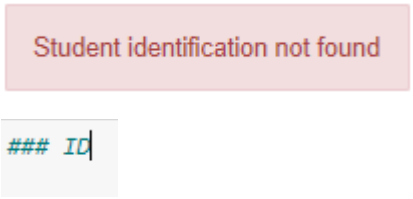

Figure 24. Empty student id error.

D. Corrector engine

The corrector generated in the creation of the assignment performs three main tasks:

- Read student's assignment
- Run solutions in parallel
- Retrieve feedback
- Generate encrypted result file

The solution cell content from the student's assessment is read from the "student_assignment.ipynb" file as a python dictionary. Now the corrector engine has student's and instructor's solutions in python dictionaries.

The engine loops these dictionaries generating a several python files containing each solution. These ".py" files are then executed in parallel and their outputs captured and stored.

Outputs obtained from the parallel execution are compared in order to evaluate student assignment and JavaScript feedback is retrieved regarding these results. It is called JavaScript feedback due to the fact Jupyter Notebooks is a web platform. The encrypted result file is now generated.

Figure 25 shows how the assignment generated by the instructor, the assignment corrector and the student's NbExtension shape the student's interface. The interface outputs are the JavaScript feedback and the results file.

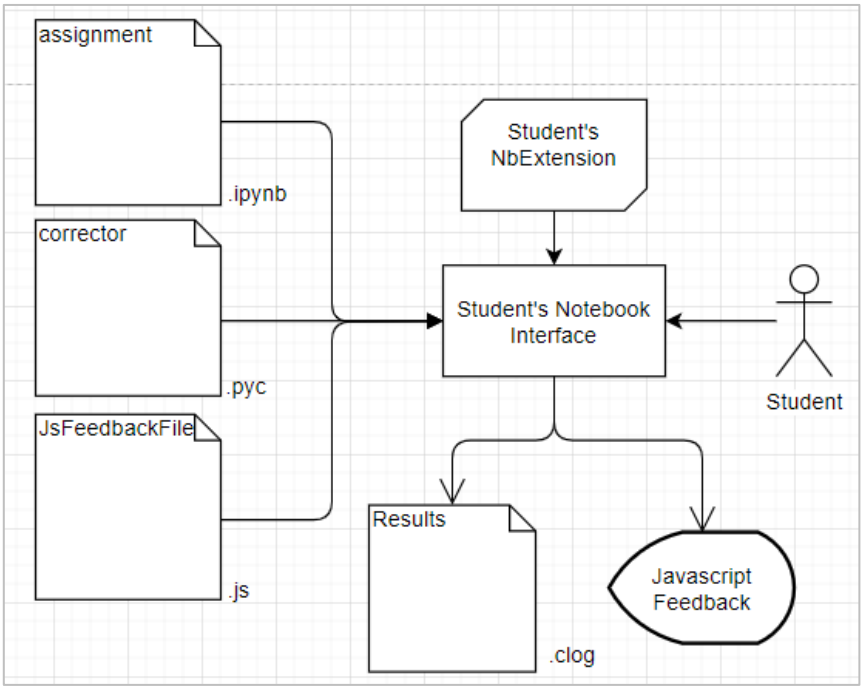

Figure 25. Student's interface structure and workflow.

E. Results interface

After handing the results file in, instructors can read it through the instructor program. The results file is decrypted and read, and results are retrieved as shown in Figure 26. If student solutions are identical, instructors will see it in the result interface. This plagiarism mechanism has been implemented by hashing students' solutions and comparing the hashes. As shown in Figure 26, instructors have to determine which assignment's results are being shown.

```
from InstructorProgram.instructor import *
readMarkFiles("SecondTest")
```

```
student Rodri solution 0 is identical to student Pascu solution 0
student Rodri solution 3 is identical to student Pascu solution 3
student Rodri solution 4 is identical to student Pascu solution 4
student Rodri solution 5 is identical to student Pascu solution 5
```

Assignment SecondTest

> Rodri

Exercise 0 - 0
Exercise 1 - 0
Exercise 2 - 0
Exercise 3 - 1
Exercise 4 - 1
Exercise 5 - 1
Mark: 3/6

> Pascu

Exercise 0 - 0
Exercise 1 - 1
Exercise 2 - 1
Exercise 3 - 1
Exercise 4 - 1
Exercise 5 - 1
Mark: 5/6

Figure 26. Results of assignment 'SecondTest'.

VII. USER GUIDE

In this chapter we will elaborate a user guide. The guide is divided into two parts, one for student users and the other one for instructor users.

A. Instructor User

The workflow starts in the instructor's interface. Before assignment creation, instructors may install the instructor's NbExtension in order to facilitate assignment creation. If instructors decide not to use the NbExtension they will still be able to create assignments by following the steps described in the 'Without NbExtension' section.

Instructors create the assignment by creating an empty notebook and opening it. Instructors must define and fill the 'ID' cell as shown in Figure 27. Instructors must also create the assignment Jupyter Notebook in the same directory the instructor's program directory is. The assignment must be named 'assignment_<assignment_id>'. The assignment id must match the id defined in the id cell inside the notebook.

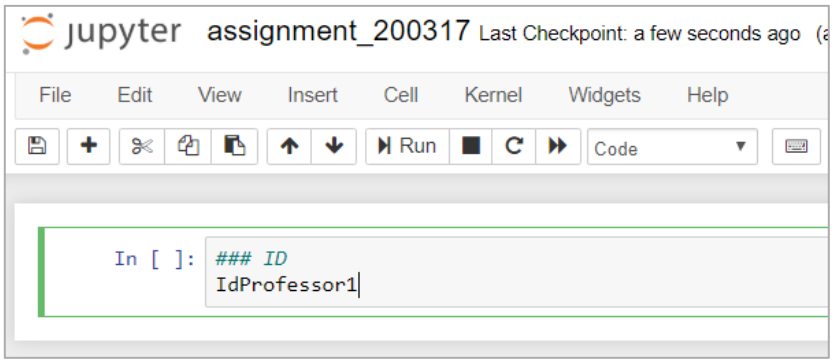

Figure 27. Instructor's identification cell.

With NbExtension

Figure 28 shows the buttons added by the instructor's interface.

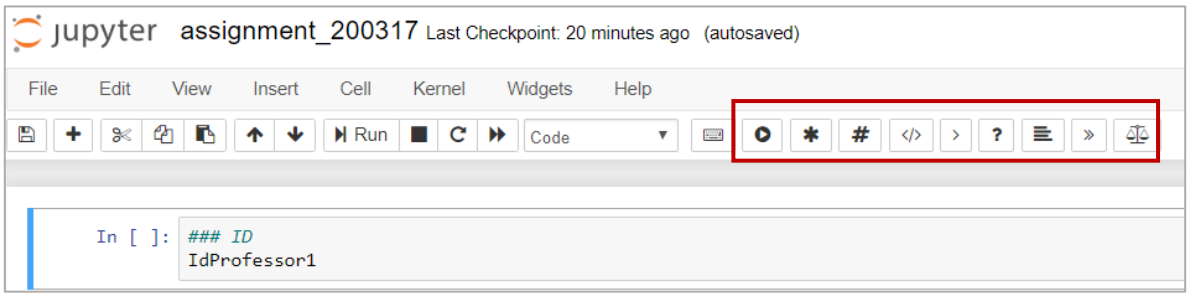

Figure 28. Instructor's interface.

From left to right buttons bring the following functionality:

- Add assignment generation cell: adds a cell which executes the python code that gathers the assignment information and creates student files. Figure 29 shows the content of this cell. The '<assignment_id>' string must be replaced by the id of the assignment. After defining the assignment id (which must match the assignment Jupyter notebook file name) the cell must be run.

```
from InstructorProgram.professor import *
genAssignment(<assignment_id>)
```

Figure 29. Instructor cell to generate the assignment.

- Add cell trio: four different cells are added to the notebook. These cells are a statement cell, a code cell, a var cell and a tip cell. Figure 30 shows the result of clicking the add cell trio button.

```
### STATEMENT

### CODE

### VAR

### TIP
```

Figure 30. Add cell trio result.

- The following four buttons are: add statement cell, add code cell, add var cell and add tip cell. They add the same functionality as add cell trio but separately.

- The next two buttons add function cells and function var cells, as showed in Figure 31.

```
### FUNCTION

### VAR FUNCTION
```

Figure 31. Function cell and var function cell.

- The last button adds a cell used to view student results. The code added by this button is showed in Figure 32. After creating this cell in the notebook, it is executed so that results are shown.

```
import professor
aux = professor.readMarkFiles()
print(aux)
```

Figure 32. Results cell.

After running the assignment generation cell, the 'Assignments' directory will be created (if it did not exist). The 'Assignment directory' will contain each assignment directory. For instance, if an instructor creates an assignment with id 'Test1', the directory 'Assignments/Assignment_Test1 will be created. This directory contains the following files:

- correctorFile.pyc: compiled corrector engine. This python file is the one which will be imported in students' notebooks in order to conduct the correction.
- js:FeedbackFile.js: contains JavaScript code and it is used by the corrector engine to provide feedback to students. This file is an unstructured JavaScript file so that students will not understand the code even if they are familiar with JavaScript. Nonetheless, this file has no effect on the result file so it cannot be used as an exploit.
- student_assignment<assignment_id>.ipynb: ready to fill Notebook for the student. It contains the whole set of statements which constitute the assignment.

Students must receive this whole directory to complete the assignment.

Without NbExtension

If instructors decide not to use their NbExtension they can still use the assignment generation program since instructor's interface doesn't contain the assignment generation engine. The only thing instructors have to do is to create the cells described in the previous section manually.

Student user

Once students receive the assignment directory they must complete the student_assignment<assignment_id>.ipynb Notebook file.

With NbExtension

While completing the assignment students may check their exercises by clicking the correction button as shown in Figure 33. Each button generates a cell at the bottom of the Notebook that executes exercises' correction and shows feedback to students. This cell is removed once the correction has finished.

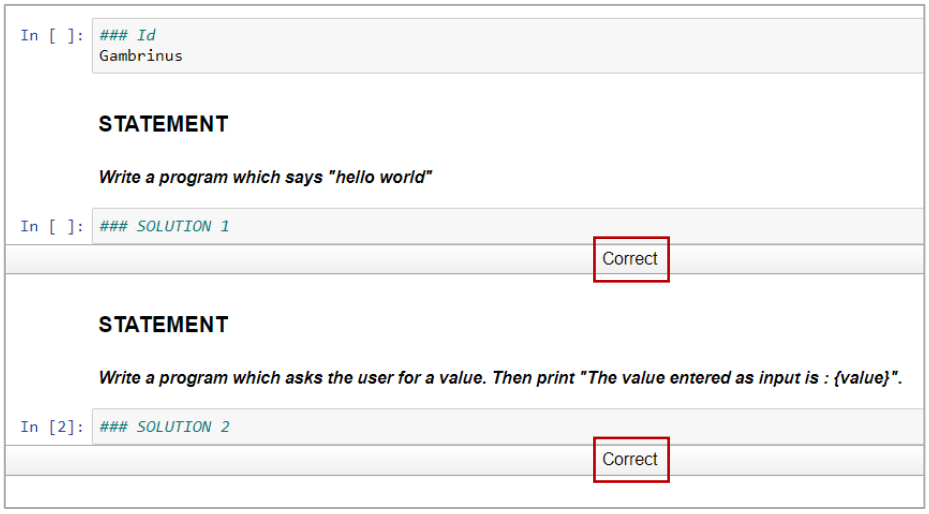

Figure 33. Ready to fill assignment with correction buttons.

```
# Single exercise corrector
import correctorFile
correctorFile.execute_single(4)
```

Figure 34. Single exercise correction cell for exercise four.

After completing the whole assignment students must click the global correction button as shown in Figure 35.

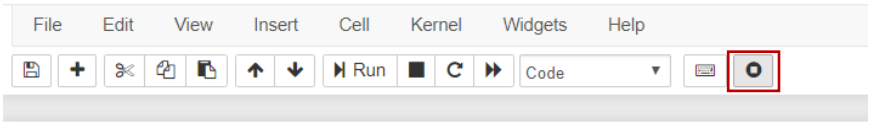

Figure 35. Global correction button.

This button adds a cell at the bottom of the notebook and executes it, correcting the whole assignment, retrieving feedback and generating the results file. It is important that students **save the assignment** notebook before they execute the global correction. This cell is removed once the correction has finished.

```
import correctorFile
correctorFile.execute()
```

Figure 36. Global exercise correction cell.

Without NbExtension

If students cannot make use of Students' NbExtension they can still use the single correction and global correction functionality of the platform. To achieve this, they will have to create and execute the cells described in the previous section (Figures 34 and 36).

VIII. CONCLUSION

The learning process' importance is undeniably significant. It does not matter the place or the time, there is no improvement without the learning that precedes a job. Therefore, teaching has been deeply studied and a wide variety of teaching methods have appeared. Some teaching methods are based on lectures since it allows professors cope with large groups of learners while others make high use of communication and feedback retrieval, since it has been proved that collaborative groups achieve big advances and learn useful aptitudes.

In this project we have focused on enhancing evaluation in the field of computer science. Improvement is needed while teaching beginner programmers since professors spend large amounts of time correcting, evaluating and trying to retrieve useful and individual feedback to students.

After studying some different existing solutions, we figured out that there was a lack of automation and feedback retrieval. For this reason, we have developed a platform to automate assignment creation and correction for beginner programmer exercises.

Our platform is completely integrated in Jupyter Notebook because many students are already familiar with this tool. At the moment the only programming language it supports is python but implementation for different programming languages can easily be done. We have developed two different NbExtensions, one for students and the other one for instructors. These extensions help users create and correct assignments through the program we have developed but are not indispensable to use the platform.

Making the platform work in a server-client way may have caused some problems so the platform itself runs completely locally, featuring complete availability. Users only need communication when instructors provide assignments and when students hand the results of them in. Correction is carried out thanks to solutions provided by instructors, which are used to compare outputs with students' solutions outputs. The corrector engine uses parameters given by instructors during the definition of the assignment to test the proposed solutions with different values for the variables in the code. Instructors may also provide some tips in order to help students if they find problems completing the assignment. Once students complete the assignment an encrypted file is created, which must be handed in to instructors so that the assignment can be evaluated. Users without access to NbExtensions are also able to generate and correct assignments.

We have not deployed our platform in a realistic environment yet since the platform should be used from the beginning of the course. We are planning to test and improve the details explained in the following chapter until the next academic year start.

Nonetheless, users have tried the platform and we have gathered some interesting user experience feedback.

IX. FUTURE LINES

Having developed a functional autocorrection tool is enough for this project's scope. Nonetheless, there is still work to do.

The first issue that may be improved is the feedback system. Students receive feedback while completing exercises, but we are thinking about developing some mechanism to analyse students' code in order to provide them more specific feedback.

Due to time constraints, professor's and student's interface are not as usable as we would have liked. Regarding instructor's interface improvement can be made identifying cell types, since they are now being identified by the first line, which may not be very fancy. Instructor's result interface may also be improved by remodelling the way results are shown, so that information retrieved is clearer. We are also planning to add a functionality to download assignment results as '.xlsx' files because so that instructors can export the results and integrate the platform with other tools.

Our platform processes python programs, but the correction idea may be extrapolated to other programming languages. It would require some time for implementation, but other programming languages can be easily included in the platform.

At the moment, NbExtensions require to be installed locally in order to use them. In the future we are planning to upload them to Jupyter NbExtension repository so that anyone will be able to access them.

We are now focused on solving and adding the mentioned issues and functionality so that our tool can be used in the next academic year.

X. BIBLIOGRAPHY

[1] "Teaching_method @ en.wikipedia.org." .

[2] "worldbesteducationsystem @ worldtop20.org." .

[3] "world-top-20-data-base @ worldtop20.org." .

[4] J. Wong and H. Z. Waring, "'Very good' as a teacher response," *ELT J.*, vol. 63, no. 3, pp. 195–203, 2009, doi: 10.1093/elt/ccn042.

[5] C. Beierle, M. Kulas, and M. Widera, "Automatic analysis of programming assignments," *DeLFI 2003 Tagungsband der 1. e-Learning Fachtagung Inform.*, pp. 144–153, 2003.

[6] "Tinker." .

[7] B. Sahil and R. Singh, "Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks," *Autom. Correct. Syntax Errors Program. Assignments using Recurr. Neural Networks*, p. 11, 2016, doi: 10.1145/1235.

[8] B. Auffarth, M. Lopez-Sanchez, Jordi Campos i Miralles, and Anna Puig, "System for Automated Assistance in Correction of Programming Exercises," *V Int. Congr. Univ. Teach. Innov.*, pp. 104 (1–9)., 2008.

[9] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated Semantic Grading of Programs," pp. 1–12, 2012.

[10] "index @ nbgrader.readthedocs.io." .

[11] "index @ unicheck.com." .

[12] "CSharpCloneDR @ www.semdesigns.com." .

[13] "f84a8cfcff881d61c36c1944a9930ce0e48d4e3a @ theory.stanford.edu." .

[14] "index @ checkstyle.sourceforge.io." .

[15] "about @ jupyter.org." .