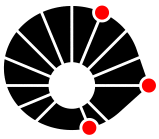
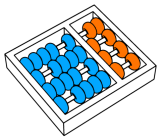




Misconceptions in Correct Code: rating the severity of undesirable programming behaviors in Python CS1 courses

Eryck Silva *Ricardo Caceffo* *Rodolfo Azevedo*

Technical Report - IC-23-01 - Relatório Técnico
January - 2023 - Janeiro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Misconceptions in Correct Code: rating the severity of undesirable programming behaviors in Python CS1 courses

Eryck Pedro da Silva* Ricardo Edgard Caceffo† Rodolfo Azevedo‡

Abstract

Misconceptions in Correct Code (MC^3) are undesirable programming behaviors, in terms of the learning objectives, that students have in code that generates the correct output. We manually analyzed 2441 students' submissions from a Python CS1 course, which were corrected by an automatic grading system (autograder), and identified 45 MC^3, divided in 8 categories. To assess this initial list of the MC^3, we administered a survey to CS1 instructors. The survey was composed of an online questionnaire and a semi-structured interview. The survey had two purposes: to classify the MC^3 by severity, identifying are prone to further investigation; and to shed light on possible educational interventions to address the MC^3, based on different CS1 teaching contexts. In total, 32 respondents answered the online questionnaire and 9 participated in the interviews. We identified 15 MC^3 as the most severe and the instructors provided feedback on automatic detection and Active Learning techniques to address these undesirable programming behaviors in CS1 classes.

1 Introduction

This report presents the process of identification and validation of Misconceptions in Correct Code (MC^3). MC^3 are a set of programming behaviors, coded by novice students in Introductory Programming Courses (CS1), which, even though the code produces the desired outcome, the presence of those behaviors might indicate a misunderstanding of the concepts taught in these courses. We named those behaviors like that because the MC^3 can be seen as a subgroup of general misconceptions, in which, for CS1, researchers identify general misunderstandings the students have regarding the core concepts of programming [1, 3, 11, 9].

The main reason of our study of MC^3 is the widespread use of automatic grading systems (also known as autograders or online judges) [5, 8, 7] which are systems that analyses some aspects of the code and automatically assigns a grade based on those aspects. This process is achieved by comparing the students' solutions using a predetermined set of input and checking if the result is equal to the expected outcome. Autograders are especially used by courses with a vast number of students. However, since autograders generally evaluate only dynamic aspects of the code, such as the desired output and running time [8], the code itself might have other properties that will remain unseen if the instructor does not check the code manually. Our hypothesis is that some of those properties could be potentially undesirable in terms of the learning outcomes.

The research process regarding the study of the MC^3 presented in this report can be divided in two main steps: the identification and categorization of the MC^3, which resulted in an initial list of

*Inst. of Computing, UNICAMP, 13083-852 Campinas, SP. eryck.silva@ic.unicamp.br
†Inst. of Computing, UNICAMP, 13083-852 Campinas, SP. caceffo@ic.unicamp.br
‡Inst. of Computing, UNICAMP, 13083-852 Campinas, SP. rodolfo@ic.unicamp.br

45 MC³ sorted by eight categories; and an assessment of this initial list, made by CS1 instructors. This assessment process was composed of an online survey with two phases: a questionnaire and a semi-structured interview to identify the severity level of the MC³ and the ones that most need to be addressed in formative feedback from the instructor so that the students can have a better understanding of the course material. The interviews had the objective of identifying different contexts of CS1 teaching as well as obtaining insight for future teaching interventions regarding the MC³.

A total of 32 volunteers responded the online questionnaire, and 9 of them participated in the semi structured interviews. The ranking of the MC³ regarding the severity concern was elaborated comparing the frequencies of those who agreed the MC³ was of severe concern to those who did not. We focused on the first 15 to indicate which ones are prone to further investigation. We also identified different CS1 teaching contexts from the interviews. These contexts relate to the use (or not) of automatic grading systems, opinions about whether code quality should be addressed in CS1, and the analysis of two possible teaching interventions: one that uses an automatic grading system that identifies the MC³; and other applied in CS1 classes using an Active Learning [2] technique.

The remainder of this report is organized as follows. In Section 2 we presented the methodology used. Sections 3 and 4 describes the obtained results from the questionnaire and the semi structured interviews, respectively. The established severity ranking of the MC³ is present in Section 5. Finally, we conclude this report in Section 6.

2 Methodology

In this section we describe the process of identification and categorization of the initial list of the MC³, as well as the development of the phases used in the online survey. Figure 1 presents the methodology used in this research.

2.1 Misconceptions in Correct Code

We collected data and evaluated the Algorithms and Computer Programming (MC102) course, a CS1 discipline taught in the State University of Campinas (UNICAMP). The course has an average number of 600 students per semester organized by graduation course. MC102 classes are split into two groups: 1) Bachelor in Computer Science and Bachelor in Computer Engineering; 2) Other STEM related courses. We focused on the latter group, which has the majority of students. All classes are taught in Python since 2018.

MC102 classes are divided into theory and practice lectures. During the semester, students are assigned summative laboratory exercises that counts towards the final grade. The submission of those exercises is done via SuSy[1], an autograder created and maintained by UNICAMP. SuSy grades students' solutions by running a dynamic evaluation that checks the code output with the expected result. Sometimes, other aspects, such as the running time, is also counted towards the grade.

We gathered student submissions to laboratory assignments from 19 MC102 classes that were taught in the first semester of 2020. By using log files generated by SuSy, we managed to filter only submissions that scored maximum grade. In total, we analyzed 2.441 maximum graded submissions for the first part of the course. Table 1 shows a summary of the topics covered by each assignment and the number of submitted, maximum graded and analyzed submissions. We only analyzed half of the maximum graded submissions for Lab06 and Lab07 because those assignments covered a similar topic.

[1] `https://www.ic.unicamp.br/~susy/`

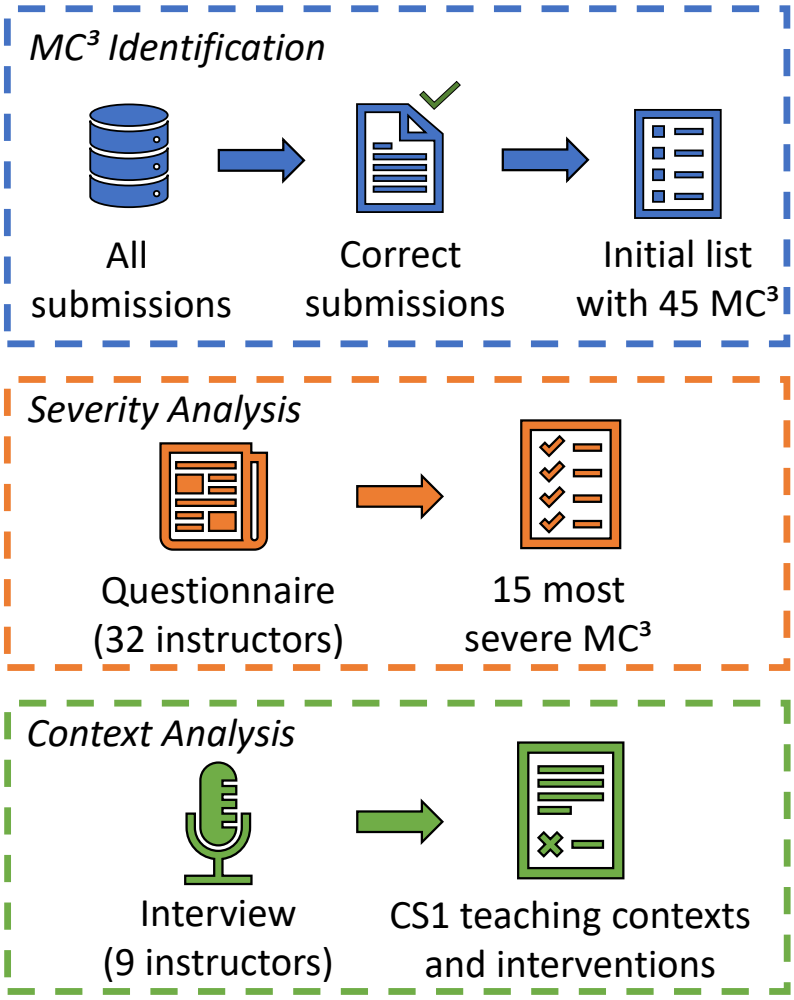


Figure 1: General process of the methodology used in this research.

Table 1: Description of how many student solutions to the laboratory assignments were submitted, maximum graded and analyzed.

ID	Related Topic	Submitted	Max Graded	Analyzed
Lab01	Arithmetic operations: the *int* type	535	529	529
Lab02	Arithmetic operations: the *float* type	499	491	491
Lab03	Logical operations: the *bool* type	511	503	503
Lab04	Conditionals I	499	478	478
Lab06	Simple loops: *while*	459	452	220
Lab07	Nested loops: *for*	456	421	220
	Total	2959	2874	2441

All the submissions were analyzed manually, following the sequential order of the assignments. Worksheets were used to annotate any observations perceived in each submission. Initially, these observations were made just by describing, in an informal manner, what was detected in the code

that potentially could indicate a misconception. Once all the submissions were analyzed, we revised all the annotations and grouped together similar observations.

We used the work of Gama et al. [4] as a starting point for the categorization names, and added new ones for the new created categories[2]. At the end of this process, we identified a total of 45 MC^3 and divided into eight categories, listed below and in Table 2. We describe each one of them in the questionnaire analysis, presented in Section 3.

- A) Variables, identifiers, and scope (8 MC^3 - from A1 to A8)
- B) Boolean expressions (12 MC^3 - from B1 to B12)
- C) Iteration (8 MC^3 - from C1 to C8)
- D) Function parameter use and scope (4 MC^3 - from D1 to D4)
- E) Reasoning (2 MC^3 - E1 and E2)
- F) Test cases (2 MC^3 - F1 and F2)
- G) Code organization (6 MC^3 - from G1 to G6)
- H) Other (3 MC^3 - from H1 to H3)

2.2 Assessment with CS1 Instructors

Although it is possible to say that the identification of the MC^3 has a certain degree of relevance, because of both the number of analyzed submissions and the fact that those were done by students of 19 different graduation classes, we decided to establish an assessment process to mitigate the threats to validity that arises from the fact that only one CS1 course was analyzed. Our primary goal is to identify the MC^3 that most needs to be addressed in formative feedback during CS1 classes to help students understand those misconceptions and mitigate their occurrence in their code.

For the assessment, we desired to understand how CS1 instructors would classify the MC^3 regarding the severity. To do so, we developed an online survey composed by two phases: a questionnaire and a semi structured interview. Both phases were primarily developed to be done online because of the *Sars-Cov-2* pandemic and the practical reasons in reaching a worldwide target audience. Since this research involved human beings, it was previously analyzed and approved by the Ethics Research Committee (CEP) of UNICAMP under the CAAE's number: 51444121.5.0000.5404.

2.2.1 Questionnaire

The questionnaire had the purpose of classifying, using Likert-items, all the 45 MC^3 from the initial identified set.

The invitation period was from January to February of 2022. We invited potential participants using both email discussion lists and direct contact. The lists used were the Interest Group on Computer Education of the Brazilian Computer Society (GIEC-SBC), the main Brazilian Computer Society (SBC) list, and the Special Interest Group on Computer Science Education (SIGCSE) of

[2]On a side note, we had named those programming behaviors as *Programming Issues* initially (and used this first name in the online survey) but as recommended by the thesis panel of the first author, we decided to be more specific about the scope of what we are studying, especially regarding to code that is deemed correct by an autograder. Thus, we changed the name to *Misconceptions in Correct Code*.

Table 2: Initial list of the 45 MC[3] identified with the analysis of MC102 laboratory assignments.

Category	ID	Misconception Name
A: Variables, identifiers, and scope	A1	Unused variable
	A2	Variable assigned to itself
	A3	Variable unnecessarily initialized
	A4	Redefinition of built-in
	A5	Unused import
	A6	Variables with arbitrary values (Magic Numbers) used in operations
	A7	Arbitrary manipulations to modify declared variables
	A8	Arbitrary treatment of the stopping point of reading values
B: Boolean expressions	B1	Redundant or simplifiable boolean comparison
	B2	Boolean comparison separated in intermediary variables
	B3	Arithmetic expression instead of boolean
	B4	Repeated commands inside *if-elif-else* blocks
	B5	Nested *if* statements instead of boolean comparison
	B6	Boolean comparison attempted with *while* loop
	B7	Boolean validation variable instead of *elif/else*
	B8	Non utilization of *elif/else* statement
	B9	*elif/else* retesting already checked conditions
	B10	Unnecessary *elif/else*
	B11	Consecutive distinct *if* statements with the same operations in their blocks
	B12	Consecutive equal *if* statements with distinct operations in their blocks
C: Iteration	C1	*while* condition tested again inside its block
	C2	Redundant or unnecessary loop
	C3	Redundant operations inside loop
	C4	Arbitrary number of *for* loop execution instead of *while*
	C5	Use of intermediary variable to loop control
	C6	Multiple distinct loops that operates over the same iterable
	C7	Arbitrary internal treatment of loop boundaries
	C8	*for* loop having its iteration variable overwritten
D: Function parameter use and scope	D1	Inconsistent *return* declaration
	D2	Too many *return* declarations inside a function
	D3	Redundant or unnecessary *return* declaration
	D4	Function accessing variables from outer scope
E: Reasoning	E1	Checking all possible combinations unnecessarily
	E2	Redundant or unnecessary use of lists
F: Test cases	F1	Verification for non explicit conditions
	F2	Specific verification for instances of open test cases
G: Code organization	G1	Long line commentary
	G2	Exaggerated use of variables to assign expressions
	G3	Too many declarations in a single line of code
	G4	Functions/variables with non significant name
	G5	Arbitrary organization of declarations
	G6	Functions not documented in the Docstring format
H: Other	H1	Statement with no effect
	H2	Redundant typecast
	H3	Unnecessary or redundant semicolon

the Association for Computer Machinery's (ACM). For direct emailing, we gathered authors from the most recent research (years 2020 and 2021) published on SIGCSE and the Institute of Electrical and Electronic Engineers (IEEE) in which the main research topic was CS1. About 185 emails were sent via this method.

The questionnaire was created using Google Forms. All questions, except the ones for the informed consent and an email to send the results, were made optional as required by the CEP. The complete questionnaire can be found in Appendix A in this report, but its basic structure was the following:

1. Informed Consent Form, describing the nature and relevance of the research, and asking for the consent of the respondent to participate in the research.
2. Basic contextual questions about the respondent: name, working institution, time teaching CS1, if has experience teaching Python, and other programming languages the respondent is familiar with.
3. Questions about the 45 MC^3. For each one, a MC^3 was presented with the name, a description, a generic example, and a description of the generic example. Two questions about each MC^3 were asked: one to rate the severity, using Likert items; and an optional text field for the respondent to make any further observations.
4. Invitation for the respondent to participate in the semi structured interview.

It was estimated that the average completion time for the questionnaire was from 15 to 25 minutes. However, we had feedback from respondents who indicated that they took more time than that (ranging from 45 to 90 minutes), so we decided to edit the questionnaire and the invitation message to correct it to 40 to 55 minutes. The questionnaire remained open for answers from January to the end of April 2022. In total, 32 participants responded to the questionnaire.

2.2.2 Semi-Structured Interview

The semi-structured interview was idealized as a further invitation to those who responded to the questionnaire. The main purposes of this phase was to gather information on the structure of the CS1 courses the instructors teach; to discover if the MC^3 can happen in other CS1 courses, and if how the instructors deal with them; and to gather the instructors opinion on teaching and learning interventions that have the intent of helping students understand and mitigate the occurrence of those misconceptions.

The semi-structured interviews happened from March to June 2022. From the 32 respondents to the questionnaire, 18 volunteered to participate in the second phase, from which we managed to interview 9 CS1 instructors. The average estimated time for the semi-structured interviews was 40 minutes (this information also was described in the questionnaire). All interviews were conducted by the main author of this report using Google Meet.

We began by interviewing Brazilian instructors and then proceeded to those located abroad. The unique criteria we used to select each participant was to be as geographically inclusive as possible, that is, we tried our best to invite participants from different institutions and regions. Although we did not fully transcribe the interviews, the compilation of the questions and answers, for each instructor, is detailed in Section 4. The basic structure of the interviews was the following:

1. Brief introduction of the researcher in which he also described the main purpose of his thesis. Solicitation to the interviewee to record the interview.

2. Set of questions about the structure of the CS1 classes the instructor teaches.
3. Set of questions about the MC^3 and how the instructor deals with them.
4. Set of questions about teaching and learning interventions with the purpose of mitigating the MC^3 and how the instructor would use them in class.

3 Questionnaire Analysis

In this section we present the data collected from the questionnaire, composed by basic questions about the respondent, and the classification of the MC^3 severity, which included both the Likert-item questions and the commentaries that respondents made about each misconception.

3.1 Basic Questions

The purpose of the questions in this first section was to gather contextual information about the respondents. It is important to note that we collected names and an institutional e-mail to contact the respondents later if they had volunteered to participate in the semi structured interview. All the data that could identify the participants was anonymized in this report.

3.1.1 Geographical Distribution and Institution Type

We used the name of the institutions that the respondents work and their institutional e-mail addresses to infer the country and the type of the institution (public or private). All 32 answers were considered valid. Most of the respondents were from Brazil and the United States of America (USA), present in Table 3. As for institution types, most of them were public, as shown in Figure 2. One respondent was from a Research and Development company instead of an university.

Table 3: Distribution of the countries from which the questionnaire had respondents. Table sorted decreasingly by the number of respondents.

Countries	**N**
Brazil	18
USA	9
Australia	1
Colombia	1
Finland	1
Slovenia	1
The Netherlands	1
Total	32

3.1.2 Experience in CS1 Teaching

We asked respondents three questions regarding their experience in teaching CS1: for how long they had taught the course, if they had taught Python, and with which other programming languages they were familiar. All the three questions had 32 valid answers.

About half of the respondents said they had more than 10 years of CS1 teaching experience, followed by instructors with 6 to 10 years, and 3 to 5 years. As shown in Figure 3, no respondents

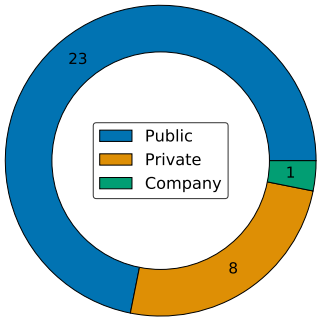


Figure 2: Types of the institutions of the respondents.

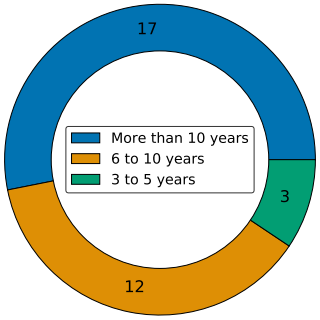


Figure 3: Years of experience in teaching CS1.

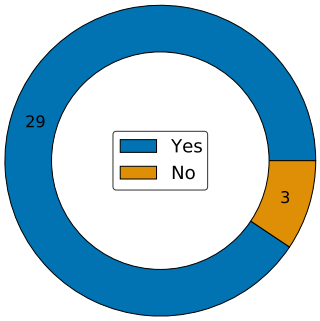


Figure 4: Experience in teaching Python.

had less than 3 years of teaching these courses. Regarding the experience in teaching Python, only three instructors stated that they had never taught this language, as shown in Figure 4. Finally, as there were many other programming languages the respondents were familiar with, we decided to create a word cloud to represent the answers: C++, C, and Java were among the most cited, as shown in Figure 5.

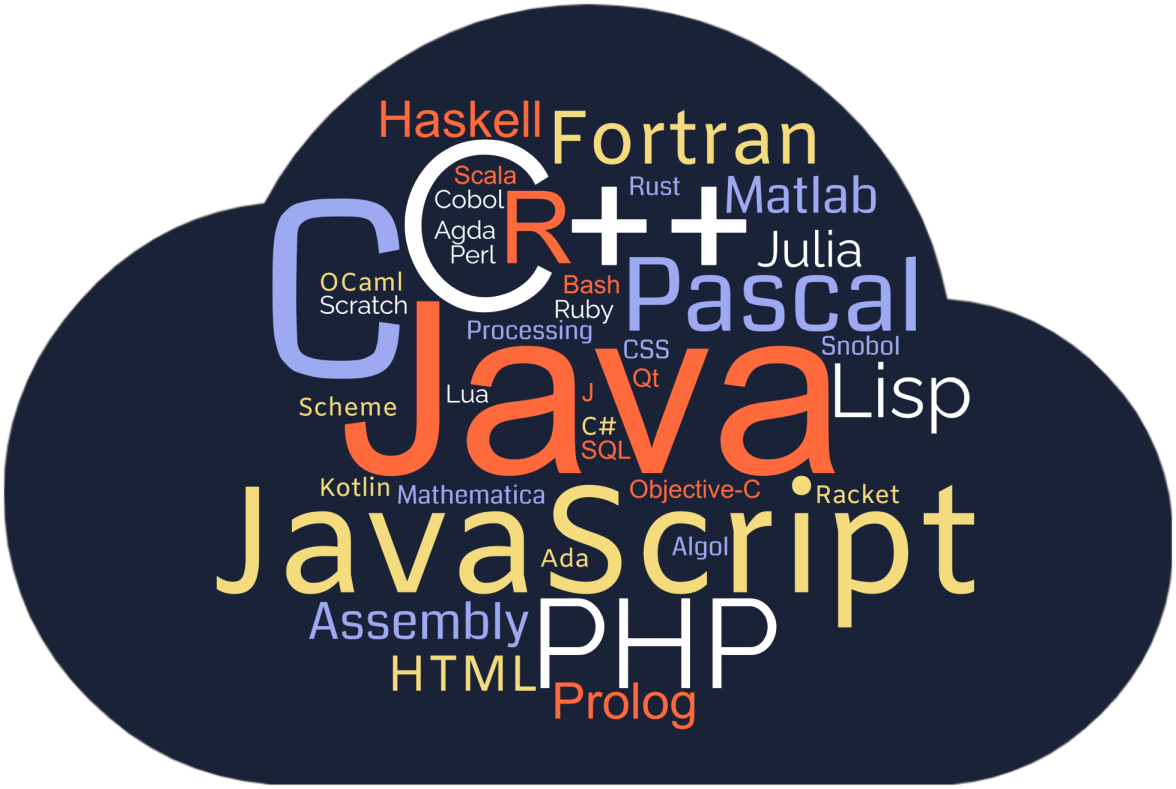


Figure 5: Other programming languages the respondents have familiarity.

3.2 MC³ Classification

The responses regarding the severity classification of the MC³ are described in this section. For each MC³ we present the original description, the Likert-item frequency distribution, and an analysis of the commentaries the respondents made. At the end of each MC³ category, an overall category classification is also presented, showing the cumulative percentage distribution of all the MC³ in said category.

There were answers to the Likert-item classification that were occasionally left in blank by respondents. However, we did not discard it since this situation did not happen for every answer of the respondent. Instead, we classified it as intentionally left blank. In the Likert-item frequency distribution, blanks are represented as "No Answer." In the cumulative percentage of the frequency distribution, present at the end of each MC³ category, blank answers were removed prior to calculation.

The analysis of the commentaries was conducted using content analysis [6, 10]. We classified the respondents' commentaries regarding 4 distinct topics about the MC³: severity, frequency, causes for occurrence, and mitigation approaches. It is important to notice that not all respondents who answered the Likert-item classification provided additional comments for the respective MC³. Therefore, we also inform the number of commentaries present in each question.

3.2.1 Category A: Variables, identifiers, and scope

This category presents MC³ that are related to variables, identifiers and their scope. There are 8 MC³ in total, cataloged from A1 to A8.

3.2.1.1 A1: Unused variable

Original Description: Indicates an occurrence in which a variable was declared somewhere in the code, but it wasn't used. This MC³ could lead to confusion in future reading and maintenance. In Algorithm 1, *var2* and *var3* were declared in lines 2 and 6, respectively, but were not used.

Algorithm 1: **A1**: Unused variable.

```
1 var1 = 3
2 var2 = 4
3 func1(var1)
4
5 def foo(arg1, arg2):
6     var3 = arg1 + arg2
7     return arg1 + arg2
8
9 func2(foo(5, 5))
```

Likert-item Classification: Figure 6 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 18 comments for this question. The respondents believe that although this MC³ is common, its severity is not of high concern. According to the instructors, some of the reasons regarding this low severity concern are the fact that unused variables are easily removed in future refinements, and they do not indicate a misunderstanding of concepts. Respondents also stated that the lack of time reserved for solving the exercise and lack of attention,

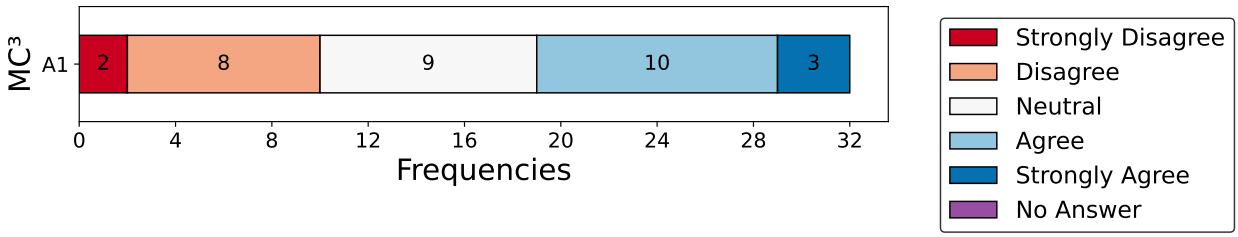


Figure 6: Frequency distribution obtained for the Likert-item severity classification of A1.

denoted by the student changing only some parts of the code and forgetting about the rest, are one of the main causes for this behavior. Some of the answers described there are ways of automatically detecting this problem, thus reducing the need of informing the students about it. Finally, one respondent said that even though unused variables do make a program hard to maintain, he would emphasize this problem in a future course other than CS1.

3.2.1.2 A2: Variable assigned to itself

Original Description: Indicates an occurrence in which a variable receives itself as a value. This MC³ could be related to a misunderstanding of the concepts of assignments of variables. In Algorithm 2, *var* is declared with itself as a value in line 2.

Algorithm 2: **A2**: Variable assigned to itself.

```
1 var = 3
2 var = var
```

Likert-item Classification: Figure 7 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

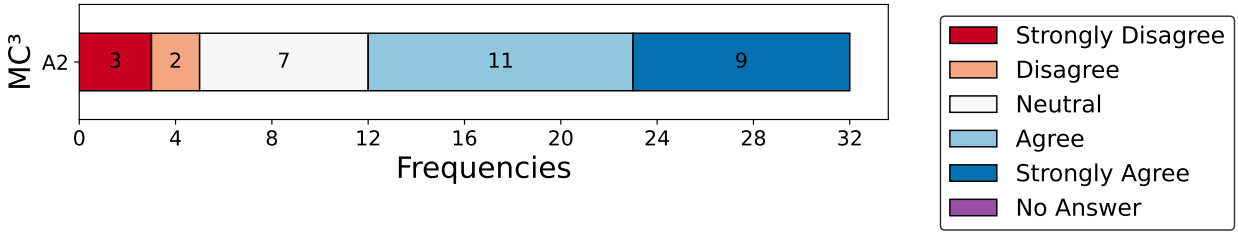


Figure 7: Frequency distribution obtained for the Likert-item severity classification of A2.

Commentaries Analysis: We received 15 comments for this question. Although none of the respondents said they have seen this MC³, most of them believe that it is considerably severe. Misunderstandings about variables and assignments were the most described reasons that classifies this behavior with a considerable severity. Some respondents also said this MC³ could be generated by weak variable naming patterns, typos, and copy-and-paste issues. Overall, the respondents agreed this behavior is worth giving feedback on.

3.2.1.3 A3: Variable unnecessarily initialized

Original Description: Indicates an occurrence in which a variable, before being used in the code, is initialized with a determined value, but this assignment is not necessary. One possible reason for the occurrence of this MC³ could be that some types of variables needs to be initialized with a value (e.g. accumulators) or a declared type (e.g. lists). So, based on this, students might think all variables share this necessity. In Algorithm 3, *var1* and *var2* were assigned an empty list and the *str* type, unnecessarily, in lines 1 and 2, respectively, before being assigned other values.

Algorithm 3: **A3**: Variable unnecessarily initialized.

```
1 var1 = ''
2 var2 = str
3 var1 = func1()
4 var2 = func2()
```

Likert-item Classification: Figure 8 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

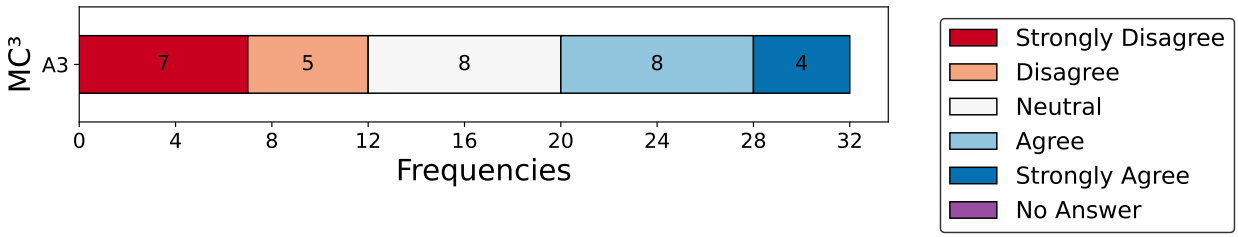


Figure 8: Frequency distribution obtained for the Likert-item severity classification of A3.

Commentaries Analysis: We received 16 comments for this question. In general, the respondents' thoughts about this MC³ were polarized. Some of them rated the severity concern as high because of examples they have seen in class that represent misunderstandings, such as the student routinely doing this, or the initialization trying to replicate a function call. Respondents that did not classify this behavior as severe indicated that even though this is generally not required in Python, students are denoting attention by initializing all variables in a desired state. The most frequent reason answered for the occurrence of this MC³ was previous education in strongly typed programming languages that students may have before studying Python.

3.2.1.4 A4: Redefinition of built-in

Original Description: Indicates an occurrence in which a preestablished Python command has its functionality overwritten with another purpose, generally being used as a variable name. Although Python allows the action, this MC³ could lead to future confusion if the student tries to use the original, intended built-in. In Algorithm 4, the variables *max*, *list*, and *bool* (which are Python built-ins) were declared and used in the code.

Algorithm 4: **A4**: Redefinition of *built-in*.

```
1 max = 23
2 list = [max]
3 bool = True
4 func(max, list, bool)
```

Likert-item Classification: Figure 9 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

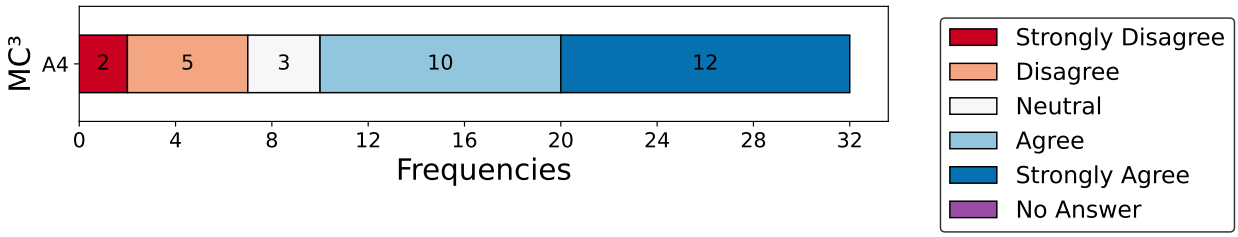


Figure 9: Frequency distribution obtained for the Likert-item severity classification of A4.

Commentaries Analysis: We received 16 comments for this question. According to the respondents, this MC³ was considered severe because it can cause unexpected program malfunctions that are difficult to detect. However, the answers also stated the occurrence of this MC³ is strongly related to Python and the chosen natural language the students use while programming (if they tend to use English, Portuguese, or Spanish, for example). As for those who did not classify this severity concern as high, respondents stated that students might not know all the built-ins at first, and the suggested programming environment should warn the programmer if this happens.

3.2.1.5 A5: Unused import

Original Description: Indicates an occurrence in which an import from a library is declared in the code, but none of its functionalities are used. This MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 5, the *math* module was imported, but none of its functionalities were used in the code.

Algorithm 5: **A5**: Unused *import*.

```
1 import math
2 base = 3
3 exp = 4
4 func(base**exp)
```

Likert-item Classification: Figure 10 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

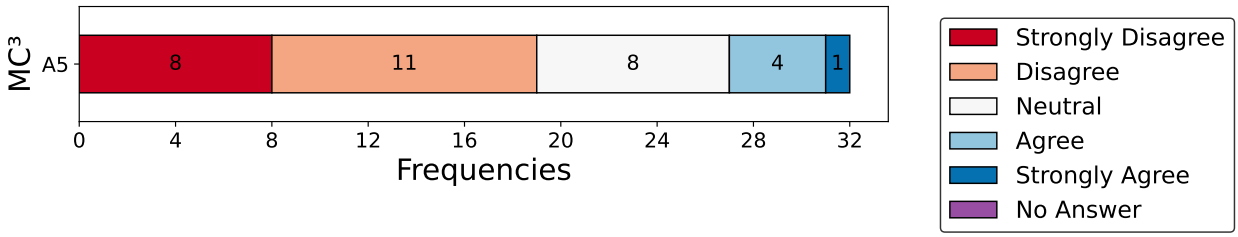


Figure 10: Frequency distribution obtained for the Likert-item severity classification of A5.

Commentaries Analysis: We received 14 comments for this question. In general, respondents, comparing this MC³ with A1, said its severity concern is not high. Again, most of the answers stated the unused import is due to a lack of attention and they can be easily removed from the code. Respondents also said that some environments can even automatically detect and inform this to the programmer. Another reason for the occurrence might be the student reusing code examples that needed the import. One statement, though, alerted this MC³ could potentially limit the portability of the code, since some machines might not have all required libraries installed.

3.2.1.6 A6: Variables with arbitrary values (Magic Numbers) used in operations

Original Description: Indicates an occurrence in which variables have arbitrary values used in the code. These values, also known as "magic numbers" are generally dependent on the problem instance. This MC³ could lead to confusion in future reading and maintenance of the code, due to the dependence on the problem instance. In Algorithm 6, *var1* and *var2* were declared with arbitrary numbers to be used in code operations.

Algorithm 6: **A6**: Variables with arbitrary values (Magic Numbers) used in operations.

```
1 var1 = 0
2 var2 = 10000000000000000000000000000000000000000000000000
3 var3 = int(input())
4 if var1 <= var3 < var2:
5     (...)
```

Likert-item Classification: Figure 11 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

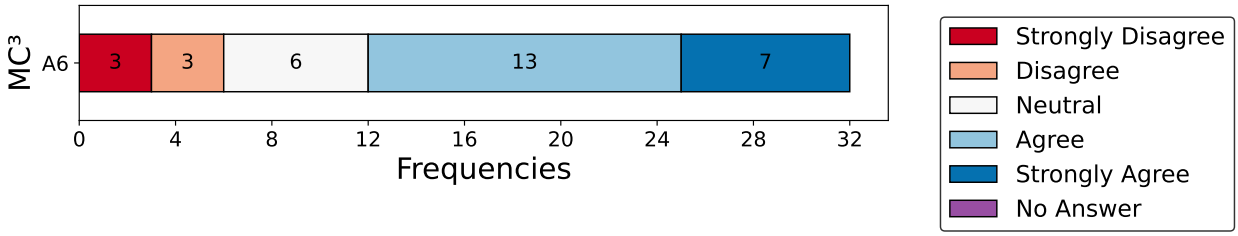


Figure 11: Frequency distribution obtained for the Likert-item severity classification of A6.

Commentaries Analysis: We received 15 comments for this question. Regarding this MC³, most of the respondents who said its concern was severe did not leave a commentary, thus making it difficult to understand the causes. Some answers stated that this MC³ can lead to future problems in maintenance or shows a sign of lack of confidence with the covered topics in the course so far. Those who did not believe the concern of this MC³ is severe stated that magic numbers are not a problem in CS1 but starts to be if this behavior lingers on future courses. In terms of frequency, respondents generally said this is a common MC³.

3.2.1.7 A7: Arbitrary manipulations to modify declared variables

Original Description: Indicates an occurrence in which variables have their values altered by arbitrary means, such as erasing its value by assigning an empty string to it. Because of those

arbitrary manipulations of variables, this MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 7, *var* has its own value erased by assigning an empty string to it in line 4 before being assigned another value in line 5.

Algorithm 7: **A7**: Arbitrary manipulations to modify declared variables.

```
1 var = int(input())
2 while var != -1:
3     func(var)
4     var = ''
5     var = int(input())
```

Likert-item Classification: Figure 12 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

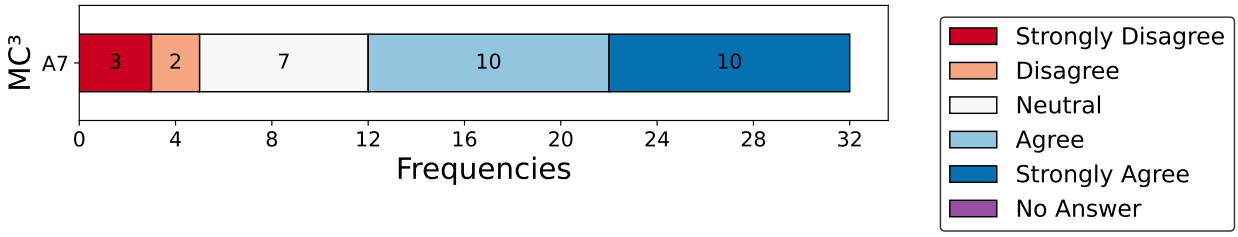


Figure 12: Frequency distribution obtained for the Likert-item severity classification of A7.

Commentaries Analysis: We received 14 comments for this question. Even though respondents stated this MC³ does not happen very often, most of them classified its severity concern as high. The reasons corroborating this classification regarded potential misunderstandings with the concepts of variables and assignments. This time, in contrast to A6, most of those who classified the concern of this MC³ as not severe did not leave a commentary. The ones who did, justified it with the infrequent nature of this behavior.

3.2.1.8 A8: Arbitrary treatment of the stopping point of reading values

Original Description: Indicates an occurrence in which the stopping condition for reading data is executed by arbitrary means, such as including the indicator in the set of valid data to be treated afterwards. Due to the presence of a set with valid and invalid data, this MC³ could lead to future malfunctions if the student forgets to remove the stopping condition from the set. In Algorithm 8, a *while* loop is declared to read a set of inputs which stopping condition is the value "-1". In the code, the stopping condition is read, assigned to variable *lst1* (the list with valid inputs), and is only removed in a check in line 9.

Algorithm 8: **A8**: Arbitrary treatment of the stopping point of reading values.

```
1 while True:
2     var1 = int(input())
3     lst1.append(var1)
4     if var1 == -1:
5         break
6
```

```
7 for iter in lst1:
8     if iter = -1:
9         lst1.remove(iter)
```

Likert-item Classification: Figure 13 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

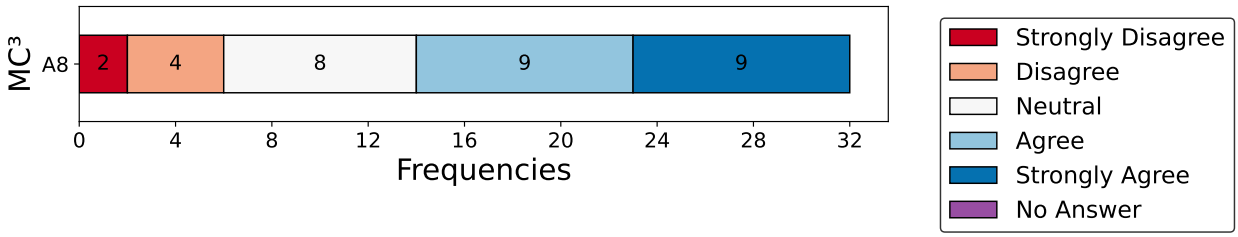


Figure 13: Frequency distribution obtained for the Likert-item severity classification of A8.

Commentaries Analysis: We received 13 comments for this question. The commentaries, in general, reveal that the severity concern of this MC³ is somewhat relevant due to lack of attention and thought organization of the student. There appears to have some discrepancies about the frequency of this behavior, since some respondents said it happens every semester and others stated they have never seen it. Some answers said that even though this MC³ generates some extra complexity in the code, it is acceptable by CS1 students. One comment attributed the cause for this behavior to be the teaching of the *break* statement.

3.2.1.9 Overall Category Classification

Figure 14 denotes the distribution of the severity concern of all the eight MC³ present in Category A.

3.2.2 Category B: Boolean expressions

This category presents MC³ that are related to boolean expressions and conditional statements. There are 12 MC³ in total, cataloged from B1 to B12.

3.2.2.1 B1: Redundant or simplifiable boolean comparison

Original Description: Indicates an occurrence in which a boolean comparison can be expressed in a simpler way. This MC³ could be related to a misunderstanding of boolean comparisons, possibly leading to code difficult to read and maintain in the future. In Algorithm 9, it is not necessary to verify if the boolean comparison declared in line 1 is *True*.

Algorithm 9: **B1**: Redundant or simplifiable boolean comparison.

```
1 if (var1 > var2) == True:
2     var3 = True
3 else:
4     var3 = False
```

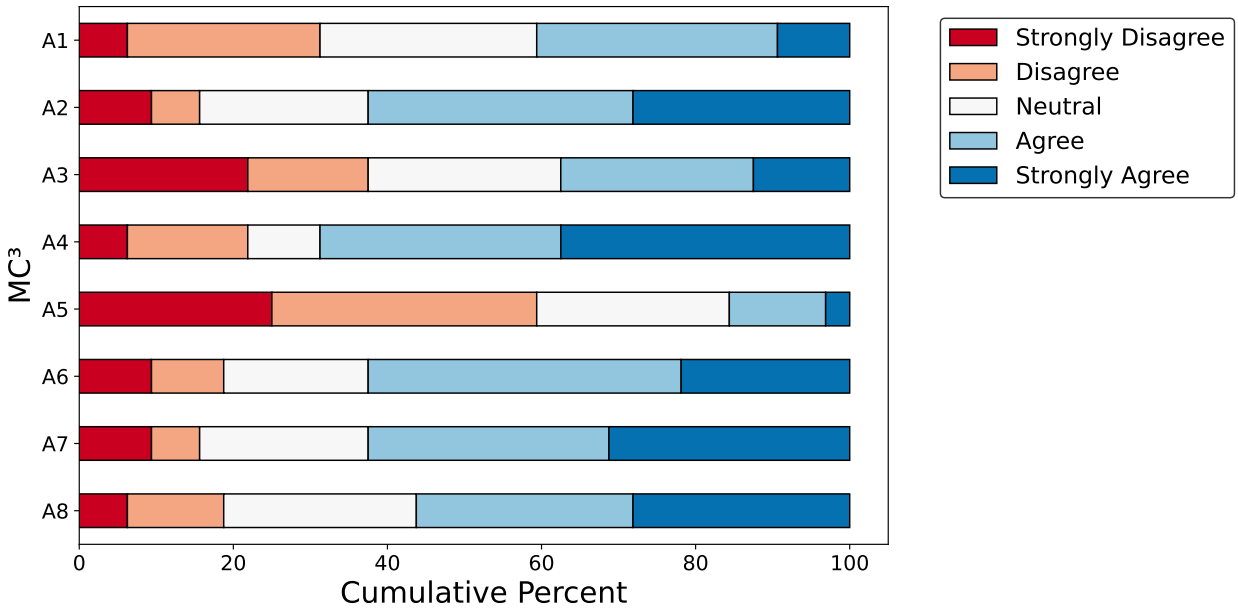


Figure 14: Cumulative percentage of the Likert-item classification for the MC³ in Category A.

Likert-item Classification: Figure 15 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

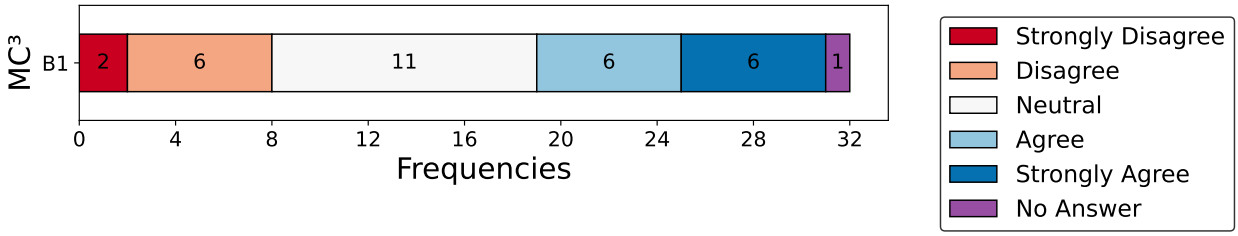


Figure 15: Frequency distribution obtained for the Likert-item severity classification of B1.

Commentaries Analysis: We received 14 comments for this question. In general, responses that stated about the severity concern of this MC³ were equally divided. A group said that students might be showing a lack of understanding about Booleans and by the end of the semester all students should know the simplified statement, thus classifying it as severe. Other group stated this MC³ is due to lack of programming experience in novice CS1 students and will naturally disappear in future courses. One answer also said that the redundant syntax is clearer for the student to understand what is being coded. Regarding the frequency, respondents agreed that this is a frequent behavior.

3.2.2.2 B2: Boolean comparison separated in intermediary variables

Original Description: Indicates an occurrence in which a boolean comparison, with more than two factors, is coded by assigning the preliminary results in auxiliary variables, instead of calculating the whole expression in the same operation. If the student does not keep track of those intermediary variables, this MC³ could lead to confusion in future reading and maintenance of the code. In

Algorithm 10, *condRes* is the result of *cond1* ***and*** *cond2* ***and*** *cond3* but instead of being calculated in one expression, *cond1* and *cond2* are calculated first in the *condAux* intermediary variable.

Algorithm 10: **B2**: Boolean comparison separated in intermediary variables.

```
1 condAux = cond1 and cond2
2 condRes = condAux and cond3
```

Likert-item Classification: Figure 16 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

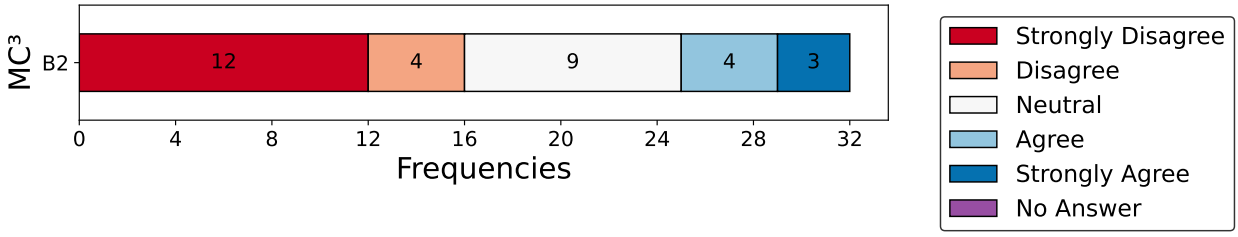


Figure 16: Frequency distribution obtained for the Likert-item severity classification of B2.

Commentaries Analysis: We received 14 comments for this question. The severity concern of this MC³ was not considered relevant. Most respondents stated they would prefer their students to divide and compute multiple, readable Boolean statements rather than having a giant expression. One respondent said that, in some cases, the auxiliary variable could even improve the code performance. One answer commented, though, that if the student is dividing all Boolean comparisons in this manner, it could indicate a misunderstanding of its concepts.

3.2.2.3 B3: Arithmetic expression instead of boolean

Original Description: Indicates an occurrence in which a boolean expression is coded via arithmetic expression, assigning numbers to certain simpler conditions and then checking a more complex case by operating these variables with numbers stored, like the sum of them all. This MC³ could be related to a misunderstanding of boolean comparisons, possibly leading to code with difficult reading and future maintenance. In Algorithm 11, the condition that needs to be checked in line 10 is made by an arithmetic expression check using integer numbers that were assigned to *cond1* and *cond2* in the previous if statements, instead of using boolean operators.

Algorithm 11: **B3**: Arithmetic expression instead of boolean.

```
1  if var1 > 0:
2      cond1 = 1
3  else:
4      cond1 = 0
5  if var2 > 0:
6      cond2 = 1
7  else:
8      cond2 = 0
9  if cond1 + cond2 == 2:
10     func()
```

Likert-item Classification: Figure 17 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

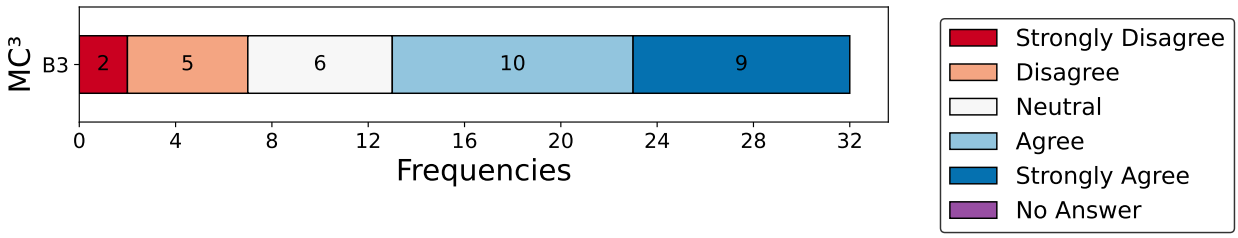


Figure 17: Frequency distribution obtained for the Likert-item severity classification of B3.

Commentaries Analysis: We received 13 comments for this question. This MC³ was considered of a high severity because of the additional, unnecessary complexity this logic adds to the code. Other answers also stated that either it is caused by a misunderstanding or the fact that the student is uncomfortable with using Boolean statements: in either case, it is convoluted and should be addressed in feedback. As for the frequency, even though most respondents said it its uncommon, it usually happens with students with some previous, uneducated programming experience. One answer said this MC³ is expected to be corrected during the whole CS course rather than just one discipline.

3.2.2.4 B4: Repeated commands inside *if-elif-else* blocks

Original Description: Indicates an occurrence in which the same commands appear in different blocks of *if-elif-else*. This MC³ could be associated with a misunderstanding of how those statements work, because the repeated code can be refactored outside the conditional statements' blocks. In Algorithm 12, the commands *func1()*, *func2()* and *func3()* are repeated in both *if* and *else* blocks declared in line 1 and 6, respectively.

Algorithm 12: **B4**: Repeated commands inside if-elif-else blocks.

```
1  if cond1:
2      func1()
3      func2()
4      func3()
5      func4()
6  else:
7      func1()
8      func2()
9      func3()
10     func5()
```

Likert-item Classification: Figure 18 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 13 comments for this question. Most of the answers classified this MC³ with a high severity concern. However, the lack of commentaries makes it

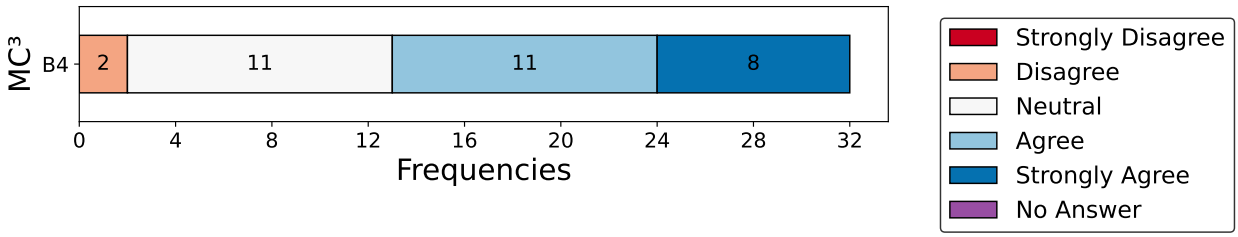


Figure 18: Frequency distribution obtained for the Likert-item severity classification of B4.

difficult to obtain more information. A group of respondents said this is a common source of bugs, even in small programs. In terms of frequency, the answers stated this MC³ seems to happen often, but one respondent said that after specifically addressing this issue in class, it tends to be rare.

3.2.2.5 B5: Nested *if* statements instead of boolean comparison

Original Description: Indicates an occurrence in which boolean comparisons are coded by nesting *if* statements instead of using logic operators. If there are few variables, this might not be an issue. However, as the number of variables grows, this MC³ could be associated with a misunderstanding of boolean comparisons, as each nesting adds complexity in the code, possibly leading to confusion in future reading and maintenance. In Algorithm 13, a nested *if* statement declared in line 1 was made to check if both *var1* and *var2* were greater than zero, instead of using a boolean comparison.

Algorithm 13: **B5**: Nested if statements instead of boolean comparison.

```
1 if var1 > 0:
2     if var2 > 0:
3         func()
```

Likert-item Classification: Figure 19 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

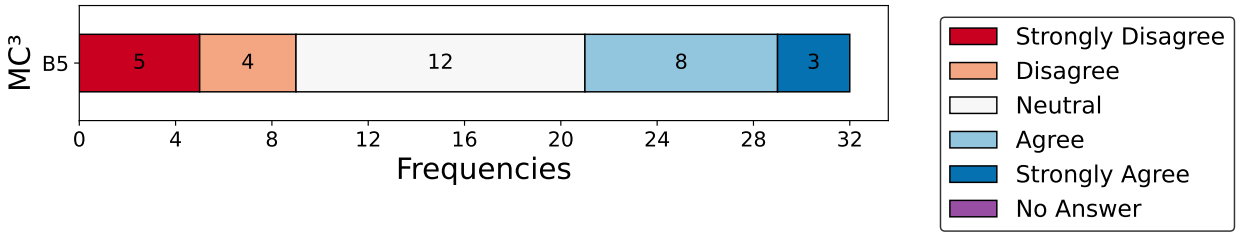


Figure 19: Frequency distribution obtained for the Likert-item severity classification of B5.

Commentaries Analysis: We received 9 comments for this question. It was difficult to understand the severity concern of this MC³ since there were few comments about it. Some respondents said nested *if* statements are used when the student is uncomfortable with using Boolean expressions, therefore indicating a degree of severity. Other instructors stated that it is a common behavior at first, but tends to vanish after being addressed in class. Those who believed this MC³ was not of high concern said that it depends on the complexity of the conditions that needs to be checked:

sometimes using nested *ifs* makes them clearer. One respondent stated that both constructions (nested and using Boolean comparisons) are acceptable.

3.2.2.6 B6: Boolean comparison attempted with *while* loop

Original Description: Indicates an occurrence in which a boolean comparison is coded using a *while* loop but there's no need for the commands inside its block to be looped, since its execution is stopped after one iteration. This MC³ could be associated with a misunderstanding of the usage of simple conditionals because there is no need to use a *while* loop that executes only once just to verify a condition. In Algorithm 14, the condition to execute *func1()*, *func2()* and *func3()* is checked using a *while* loop declared in line 1, but since there was no need for repeated iterations of those commands, a *break* statement was declared directly after them, in line 5.

Algorithm 14: **B6**: Boolean comparison attempted with while loop.

```
1 while var!=0:
2     func1()
3     func2()
4     func3()
5     break
```

Likert-item Classification: Figure 20 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

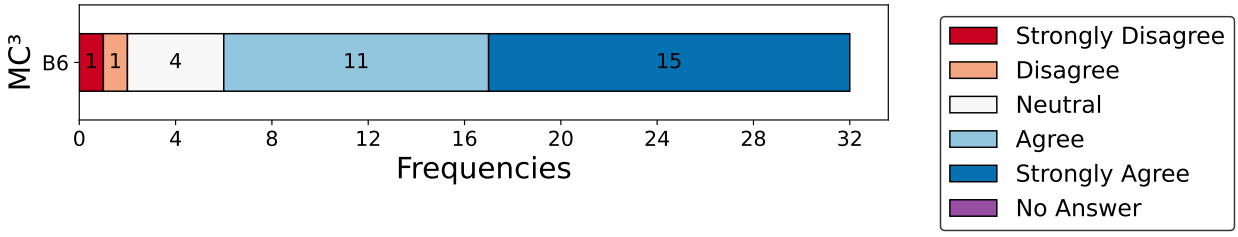


Figure 20: Frequency distribution obtained for the Likert-item severity classification of B6.

Commentaries Analysis: We received 10 comments for this question. Most respondents classified this MC³ with a high severity concern. The answers corroborating this classification stated that students might be having difficulties understanding general conditional statements from those used in loops. Others said this MC³ shows a lack of clarity in the student's reasoning. Those who did not believe this behavior has a high severity concern justified it with the infrequent nature it has, since only students with a previous, uneducated programming experience tends to replicate this MC³.

3.2.2.7 B7: Boolean validation variable instead of *elif/else*

Original Description: Indicates an occurrence in which a boolean variable (*validation flag*) is declared and assigned a condition that is tested further with consecutive *if* statements when an *elif/else* could be implemented without the need of this validation variable. This MC³ could be associated with misunderstandings of conditional statements, since the student might be trying to solve the problem using only *if* statements. In Algorithm 15, the *flag* variable is assigned in line 1 and used in lines 5 and 7 by using consecutive *if* statements instead of *elif* from the first declared *if* in line 2.

Algorithm 15: **B7**: Boolean validation variable instead of elif/else.

```
1 flag = False
2 if var1 > 0 and var2 > 0:
3     flag = True
4
5 if flag:
6     func1()
7 if cond1 and flag == False:
8     func2()
```

Likert-item Classification: Figure 21 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

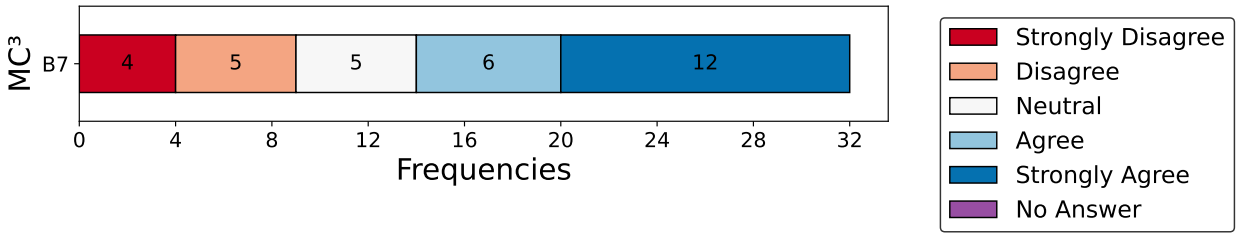


Figure 21: Frequency distribution obtained for the Likert-item severity classification of B7.

Commentaries Analysis: We received 13 comments for this question. The severity concern of this MC³ was classified as high. Those who stated it as such commented that it is a common behavior at the beginning of the course because students have only a limited understanding of *if* statements and how to structure their code. One answer said that flag variables should only be taught in Computer Architecture classes. On the other hand, respondents who did not agree with this high severity concern stated there are situations in which this structure is needed and can make the code more readable.

3.2.2.8 B8: Non utilization of *elif/else* statement

Original Description: Indicates an occurrence in which, by not using an *elif/else* statement, the code has all conditionals checked only with *if* statements and is prone to errors by specific executions that can enter multiple *if* statements when it is not supposed to. This MC³ could be associated with a non comprehension of the *elif/else* statements, and since the student is only using *if* statements, the code might be prone to undesirable errors. In Algorithm 16, because since lines 1 and 3 declared only *if* statements, both *func1()* and *func2()* are prone to be executed based on the value assigned to *var* and this behavior was not intended.

Algorithm 16: **B8**: Non utilization of elif/else statement.

```
1 if var <= 0:
2     func1()
3 if var % 2 == 0:
4     func2()
```

Likert-item Classification: Figure 22 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

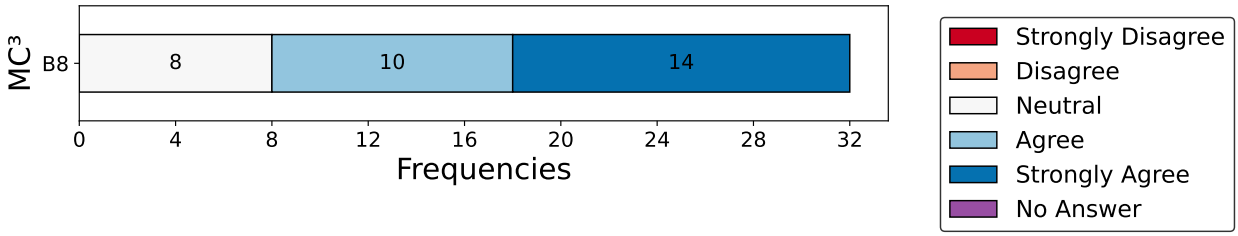


Figure 22: Frequency distribution obtained for the Likert-item severity classification of B8.

Commentaries Analysis: We received 11 comments for this question. This MC³ was classified with a high severity concern. Respondents stated this behavior indicates that students are misunderstanding fundamental concepts of *if* statements and a shows lack of clarity in student's reasoning. Commentaries also said this MC³ is frequent and can lead to bugs. One respondent commented that a good explanation addressing the number of comparisons with its impact in running time tends to correct this behavior.

3.2.2.9 B9: *elif/else* retesting already checked conditions

Original Description: Indicates an occurrence in which a condition that was already checked in an *if* statement is checked again in an *elif*, or inside an *else* block (with another *if* statement). This MC³ could be related with misunderstandings of how *elif/else* statements work. In Algorithm 17, the condition regarding *var1* in line 3 is unnecessary since the *elif* already does that.

Algorithm 17: **B9**: elif/else retesting already checked conditions.

```
1 if var1 <= 0:
2     func1()
3 elif var2 > 0 and var1 > 0:
4     func2()
```

Likert-item Classification: Figure 23 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

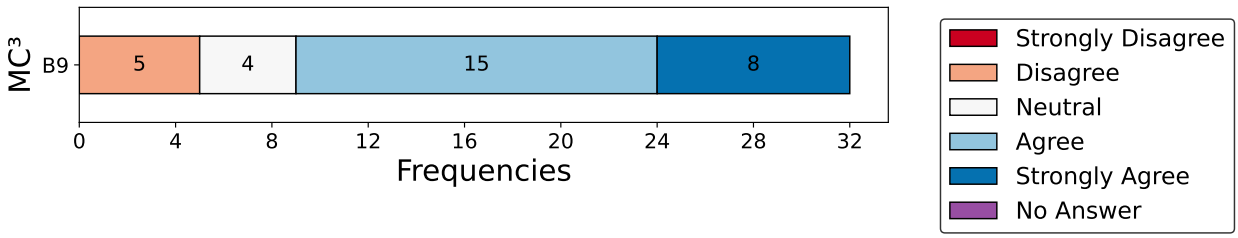


Figure 23: Frequency distribution obtained for the Likert-item severity classification of B9.

Commentaries Analysis: We received 12 comments for this question. Most of the respondents classified the severity concern of this MC³ as high. In general, the comments stated that such retesting shows a lack of clarity in the student's thoughts. It also suggests the student does not understand his own code. Some respondents said that even though this behavior is somewhat frequent, it does not bother them. One answer stated this MC³ can help novice coders to better organize their thoughts while programming, but if it is frequent in a language that does not require this retesting, it should be addressed in feedback.

3.2.2.10 B10: Unnecessary *elif/else*

Original Description: Indicates an occurrence in which *elif/else* statements are declared but their blocks contain code that has no effect (e.g. **pass**). This MC³ could be related to a misunderstanding in which the student believes that *elif/else* statements are mandatory. In Algorithm 18, even though all possible scenarios for the variable *var* were treated in *if/elif* statements in lines 1, 3 and 5, an *else* statement was declared to execute some command that will not be reached.

Algorithm 18: **B10**: Unnecessary elif/else.

```
1 if var < 0:
2     func1()
3 elif var == 0:
4     func2()
5 elif var > 0:
6     func3()
7 else:
8     (...)
```

Likert-item Classification: Figure 24 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

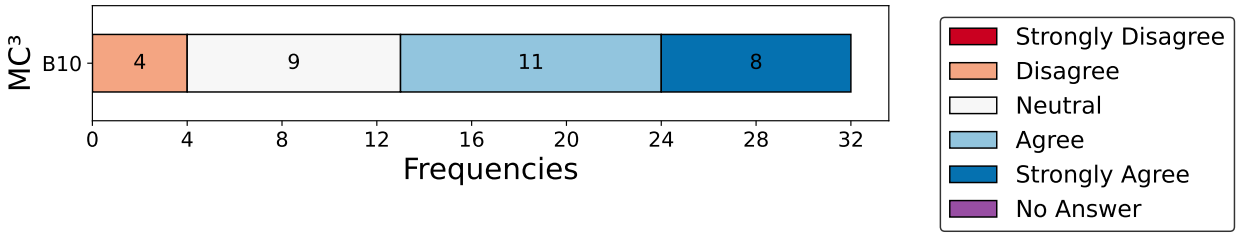


Figure 24: Frequency distribution obtained for the Likert-item severity classification of B10.

Commentaries Analysis: We received 10 comments for this question. Respondents who classified the severity concern of this MC³ as high stated that it can cause bugs and suggests students are having a limited understanding of the conditional structures. One answer also said that students should be able to determine whether code is unreachable or not. Respondents who did not classify the severity as high stated this behavior is a bad logic and takes a few weeks to disappear.

3.2.2.11 B11: Consecutive distinct *if* statements with the same operations in their blocks

Original Description: Indicates an occurrence in which consecutive *if* statements that check distinct conditions are declared, but there are repeated commands in their blocks. This MC³ could be related to a misunderstanding of *if* statements because the commands in the distinct *if*'s could be grouped in only one statement. In Algorithm 19, two distinct, consecutive *if* statements checking the conditions *cond1* and *cond2* are declared in order to execute the same *func1()* command.

Algorithm 19: **B11**: Consecutive distinct if statements with the same operations in their blocks.

```
1 if cond1:
2     func1()
3 if cond2:
4     func1()
```

Likert-item Classification: Figure 25 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

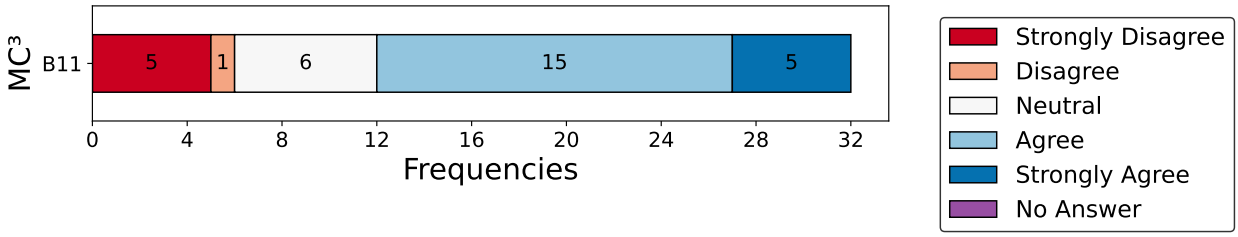


Figure 25: Frequency distribution obtained for the Likert-item severity classification of B11.

Commentaries Analysis: We received 13 comments for this question. Even though this MC³ was classified with a high severity concern, respondents stated there can be situations in which the same code needs to be executed more than once when given other conditions. However, if that is not the case, it indicates a flaw in code quality and a lack of clarity in the student's reasoning. Respondents also said this behavior in infrequent.

3.2.2.12 B12: Consecutive equal *if* statements with distinct operations in their blocks

Original Description: Indicates an occurrence in which consecutive *if* statements that check the same condition are declared, with distinct commands in their blocks. One possible reason for the occurrence of this MC³ is the student tries to emphasize two or more blocks regarding the same conditional check, but, nevertheless, the commands should be grouped in only one *if* statement. In Algorithm 20, two equal, consecutive *if* statements that check the same *cond1* condition are declared in order to execute distinct *func1()* and *func2()* commands.

Algorithm 20: **B12**: Consecutive equal if statements with distinct operations in their blocks.

```
1 if cond1:
2     func1()
3 if cond1:
4     func2()
```

Likert-item Classification: Figure 26 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

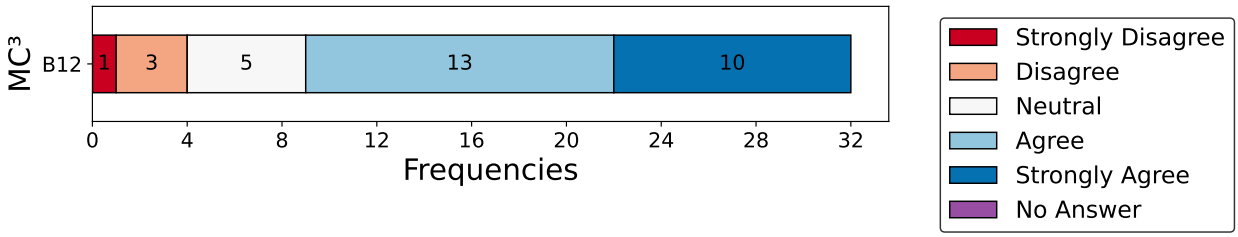


Figure 26: Frequency distribution obtained for the Likert-item severity classification of B12.

Commentaries Analysis: We received 9 comments for this question. Like B11, this MC³ was classified with a high severity concern. This time, though, the lack of commentaries polarized the stated opinions. A group of respondents said this is frequent and can make the code hard to read and understand. On the other hand, another group said it is infrequent. One respondent commented there are some situations in which this construction is needed because the code in the first *if* can alter the subsequent conditions.

3.2.2.13 Overall Category Classification

Figure 27 denotes the distribution of the severity concern of all the eight MC³ present in Category B.

3.2.3 Category C: Iteration

This category presents MC³ that are related to iteration structures and operations inside them. There are 8 MC³ in total, cataloged from C1 to C8.

3.2.3.1 C1: *while* condition tested again inside its block

Original Description: Indicates an occurrence in which a *while* loop has its condition tested again inside its block in order to verify if it's necessary to stop the looping process. This MC³ seems to happen when the loop control variable updated before the end of its block, thus creating redundant stopping conditions. The update could be done at the end of the *while* block. In Algorithm 21, the *while* condition needs to be checked again in line 4 because *varCond* is updated in line 3. If *varCond* was updated in the end, there would be no need for the check in line 4.

Algorithm 21: **C1**: While condition tested again inside its block.

```
1 varCond = (...)
2 while varCond != 0:
3     varCond = (...)
4     if varCond == 0:
5         break
```

Likert-item Classification: Figure 28 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

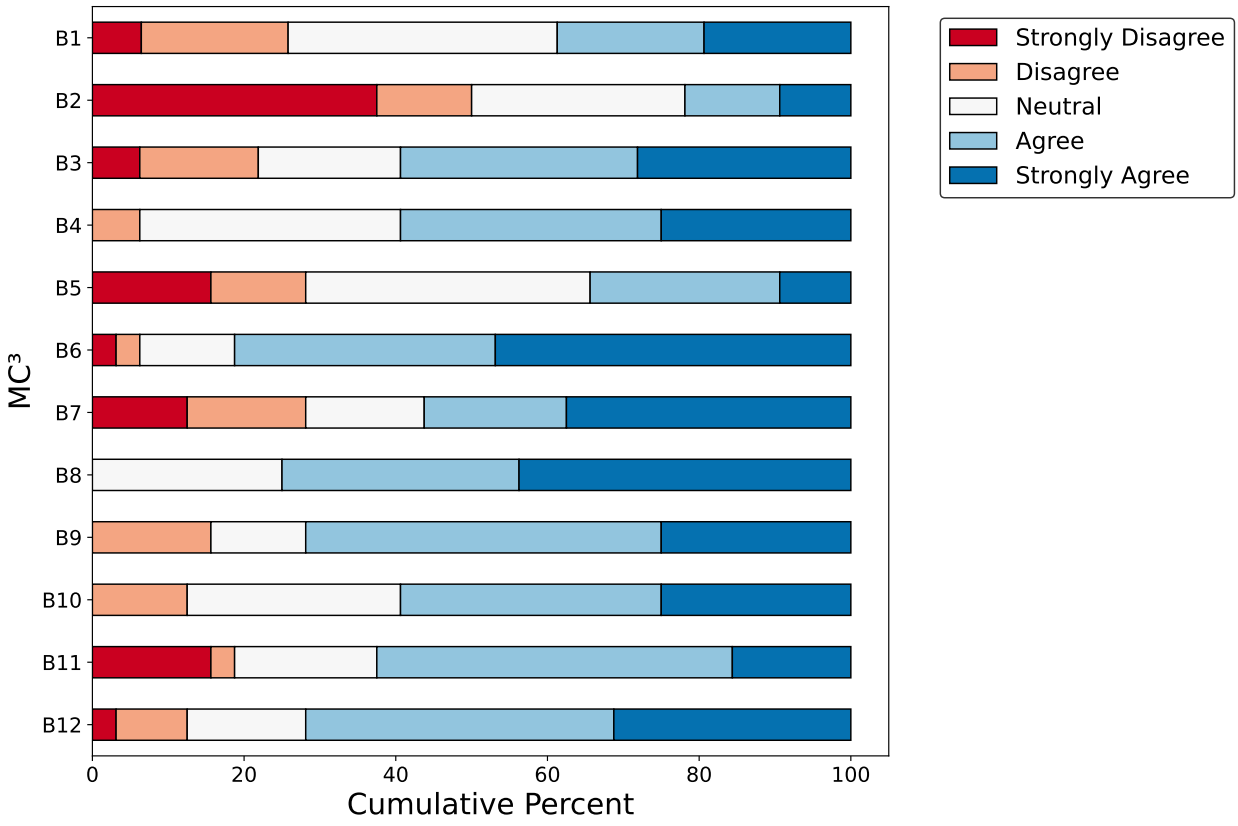


Figure 27: Cumulative percentage of the Likert-item classification for the MC³ in Category B.

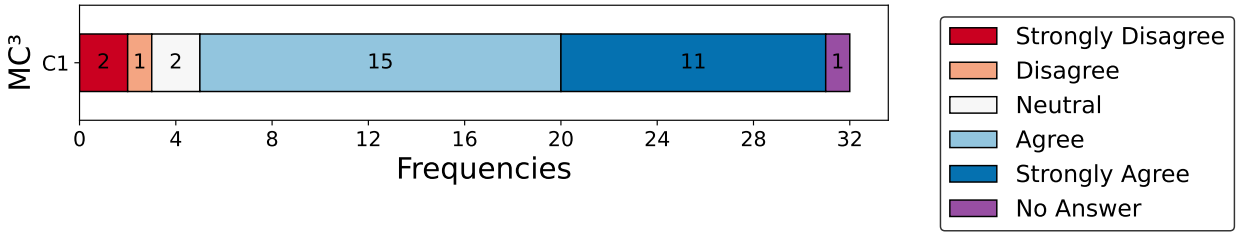


Figure 28: Frequency distribution obtained for the Likert-item severity classification of C1.

Commentaries Analysis: We received 11 comments for this question. Respondents classified the severity concern of this MC³ as high because it introduces redundancy in the code, suggesting a limited understanding of *while* loops. As for the frequency, the commentaries indicate this behavior usually happens at the beginning of the course, disappearing a few weeks later once students grasp the concept of loops. Once again, one respondent said the teaching of the *break* statement should be avoided because it can lead to structures such as this MC³.

3.2.3.2 C2: Redundant or unnecessary loop

Original Description: Indicates an occurrence in which although a loop is used, it is not needed since the whole iteration is executed only once. This MC³ could lead to confusion in future reading and maintenance of the code since it's a redundant operation. In Algorithm 22, an external *while*

loop encapsulates an internal *for* loop. The external loop executes only once since a *break* statement is declared in line 4. The internal *for* loop also executes only once, hence both loops are unnecessary.

Algorithm 22: **C2**: Redundant or unnecessary loop.

```
1 while True:
2     for i in range(1):
3         (...)
4     break
```

Likert-item Classification: Figure 29 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

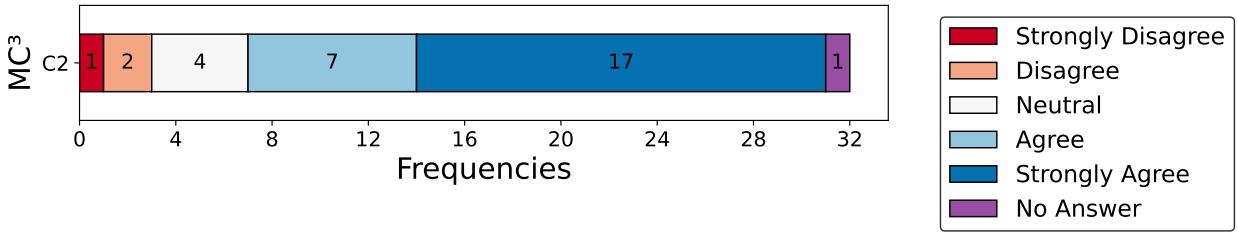


Figure 29: Frequency distribution obtained for the Likert-item severity classification of C2.

Commentaries Analysis: We received 8 comments for this question. Although most of the respondents classified this MC³ as severe, the comments stated this behavior is rare. It also indicates misunderstanding about the concept of loops, according to the responses. One respondent said to only have seen students doing it using *for* loops. The few classifications regarding a low severity concern justified it with the infrequent nature of this MC³.

3.2.3.3 C3: Redundant operations inside loop

Original Description: Indicates an occurrence in which operations are being unnecessarily calculated inside a loop, like the average of a list being calculated at each iteration. This MC³ could be related to misunderstandings of the concepts of code quality and efficiency. In Algorithm 23, the variable *var3* is attempting to calculate the average of the numbers stored in the list *lst*, but the operation is inside the loop, in line 4.

Algorithm 23: **C3**: Redundant operations inside loop.

```
1 for iter in lst:
2     var1 += 1
3     var2 += iter
4     var3 = var2/var1
```

Likert-item Classification: Figure 30 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 9 comments for this question. The severity concern of this MC³ was classified as high by most respondents. Their comments stated this behavior is due

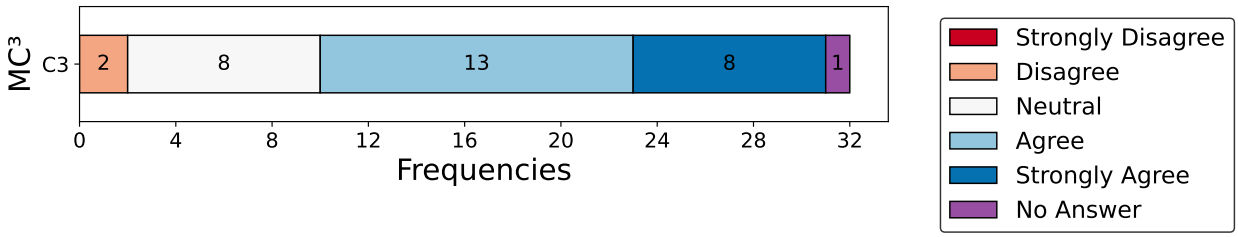


Figure 30: Frequency distribution obtained for the Likert-item severity classification of C3.

to students' lack of programming experience. It also indicates that students are having difficulties understanding loop concepts and probably do not understand math factorization. The group of respondents who did not classify the severity as high commented that even though this is a common behavior, it is related to code efficiency and should only be required in future CS courses.

3.2.3.4 C4: Arbitrary number of *for* loop execution instead of *while*

Original Description: Indicates an occurrence in which a *for* loop is declared with an arbitrary number of executions. This MC³ could be related to misunderstandings of the *while* loop, since it would be better to use it in this context. Aside from that, the student is assuming that the arbitrary number of iterations is *good enough* for all possible executions of the code, this leaves the program prone to errors. In Algorithm 24, the *for* loop was declared in line 1 to execute **65.535** times, an arbitrary number thought to be enough iterations needed before the condition *cond1* happens and executes the *break* statement out of the loop.

Algorithm 24: **C4**: Arbitrary number of for loop execution instead of while.

```
1 for i in range(65535):
2     if cond1:
3         break
```

Likert-item Classification: Figure 31 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

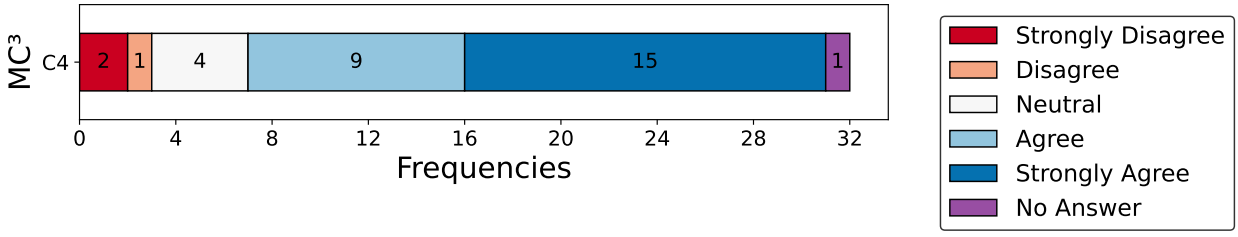


Figure 31: Frequency distribution obtained for the Likert-item severity classification of C4.

Commentaries Analysis: We received 11 comments for this question. Most of the respondents classified the severity concern of this MC³ as high. Students who do this are either misunderstanding or uncomfortable with *while* loops, according to the comments. One respondent stated that only students with previous, uneducated programming experience tends to do this MC³ in their code. This time, two responses said to avoid teaching the *break* statement as it can lead to this behavior.

3.2.3.5 C5: Use of intermediary variables to loop control

Original Description: Indicates an occurrence in which the variable that controls the loop is updated via other intermediary variables which are fuzzy or obfuscated. If the student does not keep track of those intermediary variables, this MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 25, a *while* loop is declared with *cond* as executing condition. Inside its block, *cond* is updated using the result of a function applied to an intermediary variable *var2*, which itself is a function applied to another variable, *var1*, obfuscating how *cond* is being updated.

Algorithm 25: **C5**: Use of intermediary variable to loop control.

```
1 while cond:
2     var2 = func1(var1)
3     cond = func2(var2)
```

Likert-item Classification: Figure 32 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

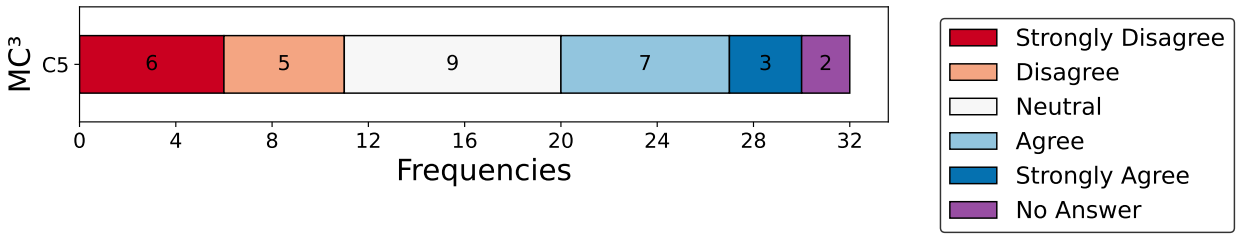


Figure 32: Frequency distribution obtained for the Likert-item severity classification of C5.

Commentaries Analysis: We received 10 comments for this question. Both the severity concern classification and responses to this question were polarized. A group stated this MC³ is common and shows a lack of clarity in student's reasoning. On the other hand, another group stated that it depends on the code: if the intermediary variables and functions are well named, this behavior does not represent a concern.

3.2.3.6 C6: Multiple distinct loops that operates over the same iterable

Original Description: Indicates an occurrence in which multiple consecutive loops that operate over the same iterable are declared with different operations being performed inside their respective blocks. This MC³ could be related to misunderstandings of the concepts of code quality and efficiency because, if the operations are over the same iterable, they could declared in the same loop. In Algorithm 26, three distinct *for* loops that iterate the same way over the same list *lst* were declared in lines 1, 4 and 7, in order to execute different commands.

Algorithm 26: **C6**: Multiple distinct loops that operates over the same iterable.

```
1 for iter in lst:
2     var1 = func1(iter)
3
4 for iter in lst:
5     var2 = func2(iter)
```

```
6
7 for iter in lst:
8     var3 = func3(iter)
```

Likert-item Classification: Figure 33 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

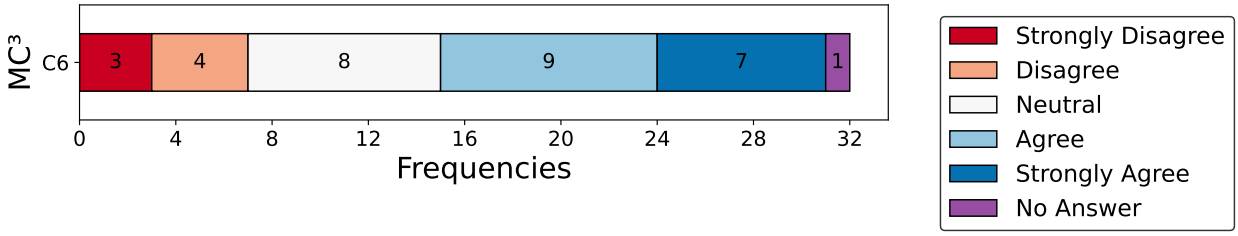


Figure 33: Frequency distribution obtained for the Likert-item severity classification of C6.

Commentaries Analysis: We received 10 comments for this question. Even though most of the respondents classified this MC³ with a high severity concern, the comments stated there can be multiple situations in which this behavior is acceptable. A group said that if all the functions over the same iterable are side-effect free, then they should be grouped into only one loop (respecting the order). However, situations in which they are dependent, such as the printing of variables, this structure is necessary, and thus is not a concern. Respondents also stated that situations in which this MC³ is non desirable tend to be rare.

3.2.3.7 C7: Arbitrary internal treatment of loop boundaries

Original Description: Indicates an occurrence in which there are specific conditionals treating the boundary values of a loop, inside it. Because these boundary checks are done in each execution of the loop, this MC³ could be related to misunderstandings of the concepts of code quality and efficiency. In Algorithm 27, specific conditions were declared for the boundary values the variable *iter* can be assigned in the execution of the *for* loop declared in line 1 (when *iter == 0* and *iter == numMaxIter -1*).

Algorithm 27: **C7**: Arbitrary internal treatment of loop boundaries.

```
1 for iter in range(numMaxIter):
2   if iter == 0:
3     func1()
4
5   if iter == numMaxIter - 1:
6     func2()
```

Likert-item Classification: Figure 34 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 11 comments for this question. This MC³ was classified with a high severity concern. A group of respondents stated that although this MC³ is common,

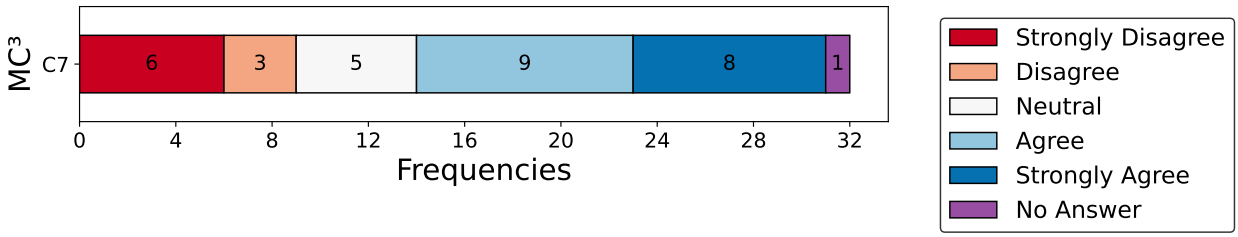


Figure 34: Frequency distribution obtained for the Likert-item severity classification of C7.

it should be avoided by declaring the boundary checks outside the loop. The group who did not classify the severity concern as high said that it is acceptable for novice programmers as it makes the code clearer to the students.

3.2.3.8 C8: *for* loop having its iteration variable overwritten

Original Description: Indicates an occurrence in which a *for* loop has its own iteration variable overwritten inside its block. This MC³ could be related to misunderstandings of the *for* loop, since the student might be thinking that this iteration variable needs to be manually incremented or it has to be used inside its block. In Algorithm 28, the iteration variable *iter* declared in the *for* loop in line 1 is overwritten in line 2.

Algorithm 28: **C8**: for loop having its iteration variable overwritten.

```
1 for iter in range(numMaxIter):
2     iter = (...)
```

Likert-item Classification: Figure 35 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

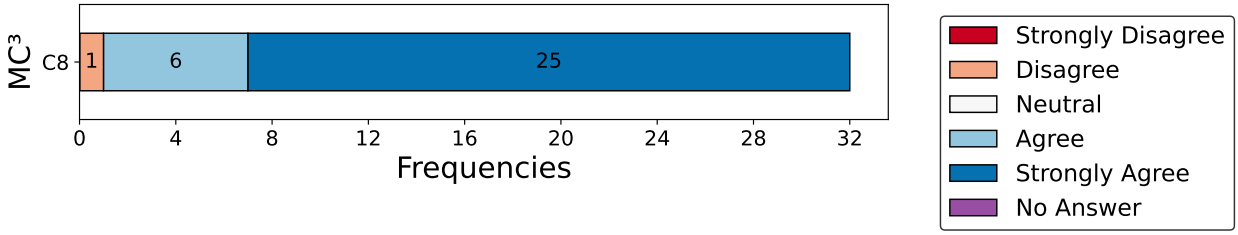


Figure 35: Frequency distribution obtained for the Likert-item severity classification of C8.

Commentaries Analysis: We received 11 comments for this question. Almost all respondents classified this MC³ with a high severity concern. Their comments stated this behavior can be disastrous in some programming languages, leading to a high number of malfunctions. Responses also said this MC³ shows a clear misunderstanding of how Python treats loops and iteration variables. One respondent commented that students also tend to do similar practices with *while* iterations, possibly leading to infinite or unexpected loop behavior.

3.2.3.9 Overall Category Classification

Figure 36 denotes the distribution of the severity concern of all the eight MC³ present in Category C.

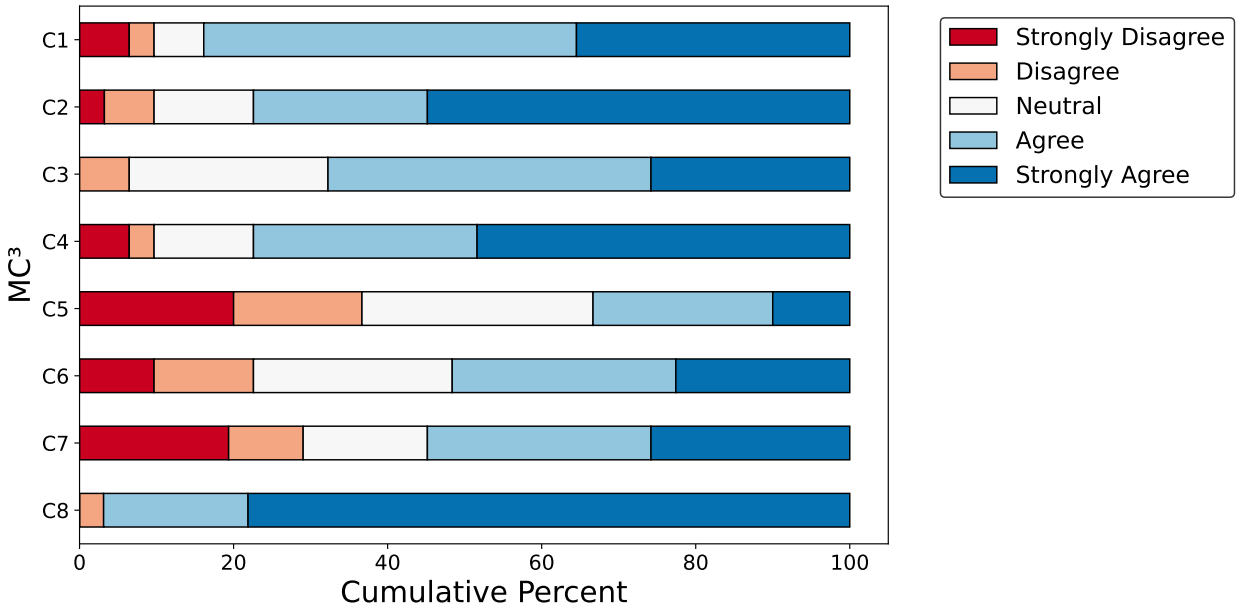


Figure 36: Cumulative percentage of the Likert-item classification for the MC³ in Category C.

3.2.4 Category D: Function parameter use and scope

This category presents MC³ that are related to the use of user defined functions, its parameters usage and function scope. There are 4 MC³ in total, cataloged from D1 to D4.

3.2.4.1 D1: Inconsistent *return* declaration

Original Description: Indicates an occurrence in which only some of the *return* declarations inside a function return an expression. If one *return* statement returns an expression, the others where nothing is returned should return *None*. This MC³ could be related to creation of code that is not formatted regarding code conventions such as the PEP8[3]. In Algorithm 29, according to PEP8, there would be a necessity of adding a *return* ***None*** in the case if the condition in line 2 was not met.

Algorithm 29: **D1**: Inconsistent return declaration.

```
1 def func1(var):
2     if var > 1:
3         return (...)
```

Likert-item Classification: Figure 37 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

[3] https://www.python.org/dev/peps/pep-0008/

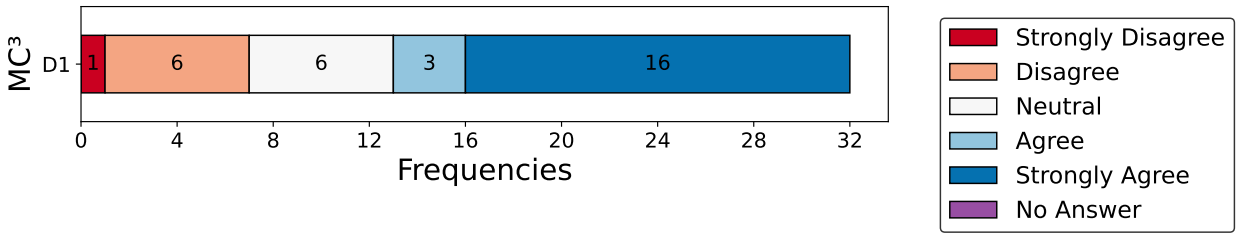


Figure 37: Frequency distribution obtained for the Likert-item severity classification of D1.

Commentaries Analysis: We received 8 comments for this question. Most respondents classified the severity concern of this MC³ as high. However, the lack of comments polarized the responses to this question. Those who did not classify it as severe said this behavior is a matter of style, and Python, by default, returns the *None* object, so the student might be using this knowledge to write a shorter code. On the other hand, a group of respondents stated that even though this is a problem with Python and is infrequent, it should be addressed because they consider it an important mistake.

3.2.4.2 D2: Too many *return* declarations inside a function

Original Description: Indicates an occurrence in which a function has a high number of *return* declarations inside its block. This MC³ could be related to misunderstandings of the concepts of code quality because the function is probably long and could be divided in other smaller functions. In Algorithm 30, a high number of *return* statements were declared inside the same function *func1()*.

Algorithm 30: **D2**: Too many return declarations inside a function.

```
1  def func1():
2      if cond1:
3          return (...)
4      elif cond2:
5          return (...)
6      elif cond3:
7          return (...)
8      (...)
9      elif condN:
10         return (...)
11     else:
12         return (...)
```

Likert-item Classification: Figure 38 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 9 comments for this question. Both the severity concern classification and the responses to this question were polarized. The group that classified it as severe stated this MC³ should be addressed in feedback because it reflects flaws in code quality, although it does not represent a misunderstanding of the *return* statement. Those who did not classify it that way said it depends on the code size and structure, sometimes being unavoidable. Respondents

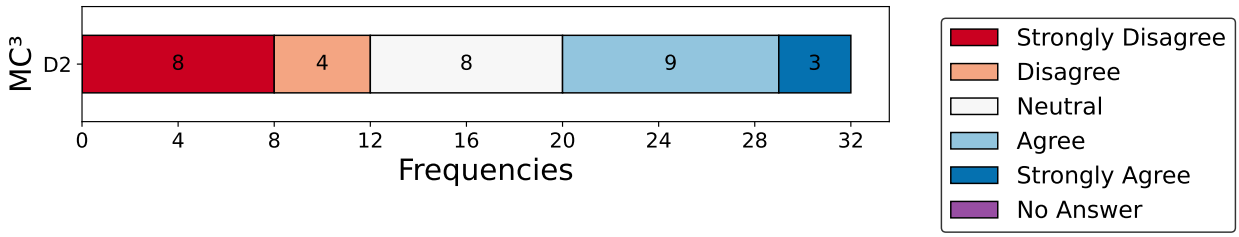


Figure 38: Frequency distribution obtained for the Likert-item severity classification of D2.

also commented they encourage students to write only one *return* per function to mitigate this occurrence.

3.2.4.3 D3: Redundant or unnecessary return declaration

Original Description: Indicates an occurrence in which a *return* declaration that has no impact over the execution of the program is present inside a function. This MC³ could be related to misunderstandings of functions, since the student might be thinking that the *return* statement is mandatory. In Algorithm 31, a *return* statement was declared inside the function *func1()* but since there was no need to return anything, it was unnecessary.

Algorithm 31: **D3**: Redundant or unnecessary return declaration.

```
1 def func1(var1):
2     print(var1)
3     return
```

Likert-item Classification: Figure 39 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

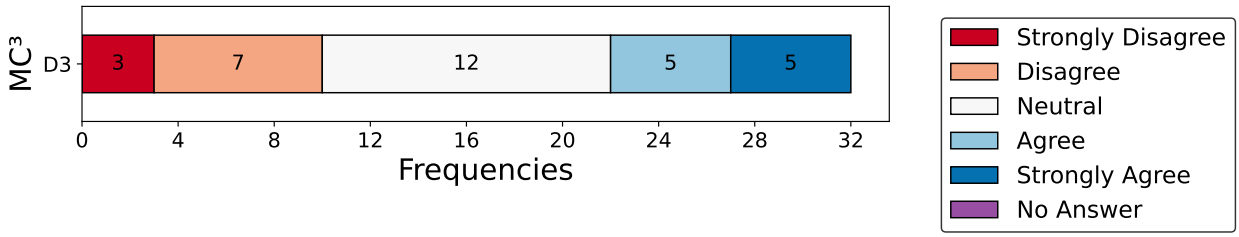


Figure 39: Frequency distribution obtained for the Likert-item severity classification of D3.

Commentaries Analysis: We received 8 comments for this question. Most respondents rated the severity concern of this MC³ as neutral but they did not leave a comment. Those who rated it having a high concern stated that it is a frequent behavior that can cause bugs, suggesting a limited understanding of the concepts of functions. On the other hand, respondents who rated it with a low concern said to not have seen this MC³ and it is a matter of style. One respondent also said this behavior can be misled by the instruction that the *return* statement is the way to explicitly end a function.

3.2.4.4 D4: Function accessing variables from outer scope

Original Description: Indicates an occurrence in which a function accesses variables from outside its scope, without passing them as a parameter. Without using the *global* declaration for the variable, Python lets some data types be accessed in read-only format (e.g. *int*, *float*) or read/write (e.g. lists and dictionaries). This MC³ could be related to misunderstandings of good practices and code quality because this occurrence could lead to confusion in future reading and maintenance. In Algorithm 32, the function *func1()* returns the sum of the variables *var1* and *var2*, but only *var1* is available in its scope.

Algorithm 32: **D4**: Function accessing variables from outer scope.

```
1 var2 = (...)
2 def func1(var1):
3     return var1 + var2
```

Likert-item Classification: Figure 40 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

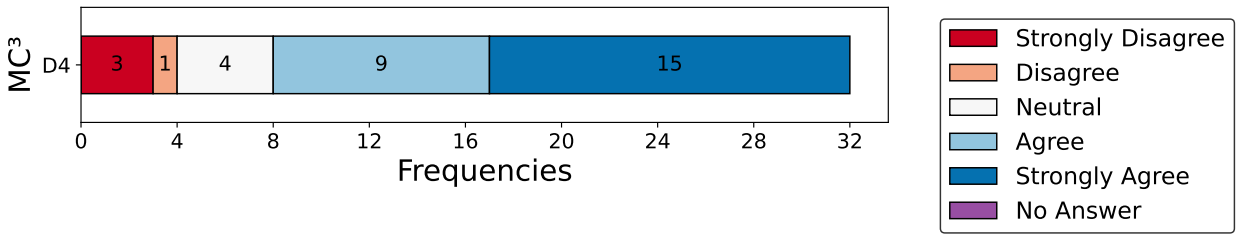


Figure 40: Frequency distribution obtained for the Likert-item severity classification of D4.

Commentaries Analysis: We received 11 comments for this question. This MC³ was classified with a high severity concern. Most respondents commented this is a Python issue that can lead to bad practices. Those who did not classify it as severe stated that global variables are acceptable in some cases, such as constants of the assignment, but abusing its usage should be avoided. Two respondents said this MC³ is frequent among students with previous, uneducated programming experience.

3.2.4.5 Overall Category Classification

Figure 41 denotes the distribution of the severity concern of all the eight MC³ present in Category D.

3.2.5 Category E: Reasoning

This category presents MC³ that are related to student reasoning while trying to solve problems in coding. There are 2 MC³ in total, cataloged as E1 and E2.

3.2.5.1 E1: Checking all possible combinations unnecessarily

Original Description: Indicates an occurrence in which all possible combinations of boolean variables and/or conditional expressions are unnecessarily declared in order to check all cases. The root cause for this MC³ could be related to misunderstandings of conditional statements. In Algorithm

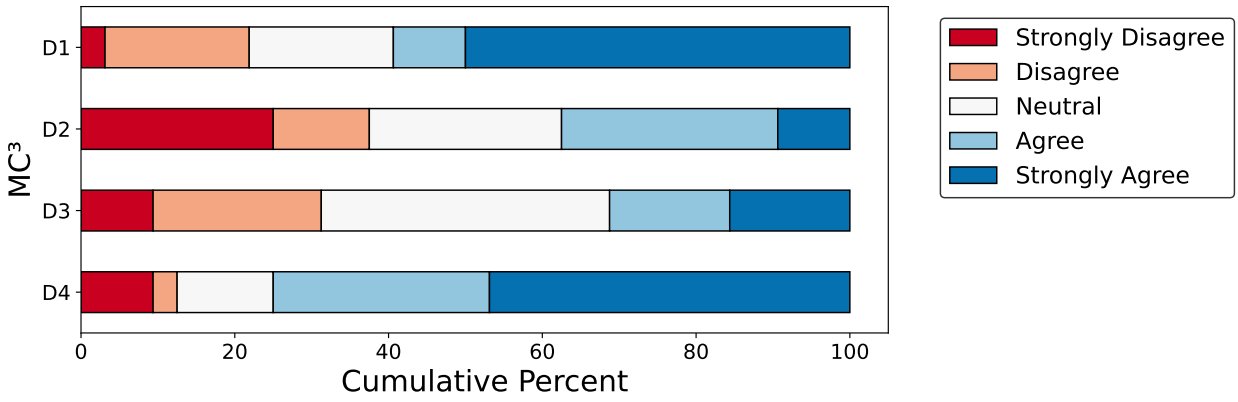


Figure 41: Cumulative percentage of the Likert-item classification for the MC³ in Category D.

33, the Truth Table is wholly declared unnecessarily in order to check all the possible combinations if *var1* and *var2* were *True* or *False*.

Algorithm 33: **E1**: Checking all possible combinations unnecessarily.

```
1 if var1 is False and var2 is False:
2     func1()
3 if var1 is False and var2 is True:
4     func1()
5 if var1 is True and var2 is False:
6     func1()
7 if var1 is True and var2 is True:
8     func2()
```

Likert-item Classification: Figure 42 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

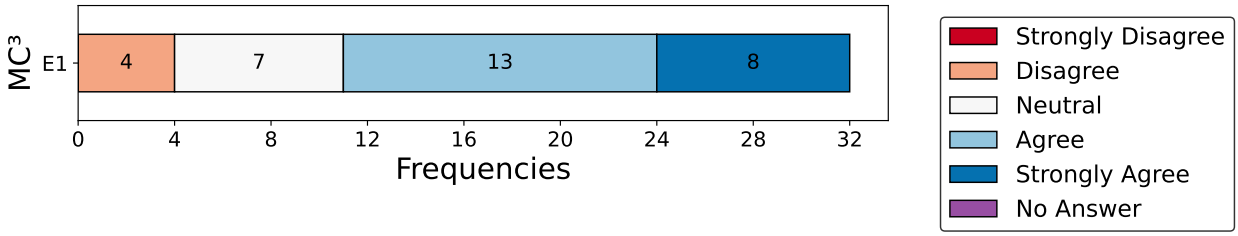


Figure 42: Frequency distribution obtained for the Likert-item severity classification of E1.

Commentaries Analysis: We received 9 comments for this question. Most respondents classified the severity concern of this MC³ as high. A group stated this behavior is frequent and shows a sign that students are failing to understand Boolean concepts. One respondent also said he runs examples in class to show how this structure is prone to bugs. However, those who did not classify the severity as high said this MC³ is acceptable within the scope of novice programming, as it helps students understand their logic. Nonetheless, those respondents agreed this should be discouraged

in more professional code.

3.2.5.2 E2: Redundant or unnecessary use of lists

Original Description: Indicates an occurrence in which lists are declared and used to perform operations that does not necessarily need the use of this data structure. This MC³ could be related to some dependence in using lists, since the student might not realize that some operations can be done while reading the values. In Algorithm 34, a *while* loop is declared in line 1 to read input data and store it in the list *lst1*. Then, a *for* loop is declared in line 6 to iterate over *lst1* and sum all of its numbers, storing it in the variable *var2*. The operation in this *for* loop in line 6 could have been already done while reading the input in the first *while* loop, without the need of storing it in a list.

Algorithm 34: **E2**: Redundant or unnecessary use of lists.

```
1 while expr1:
2   var1 = input()
3   lst1.append(var1)
4   expr1 = (...)
5
6 for iter in lst1:
7   var2 += iter
```

Likert-item Classification: Figure 43 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

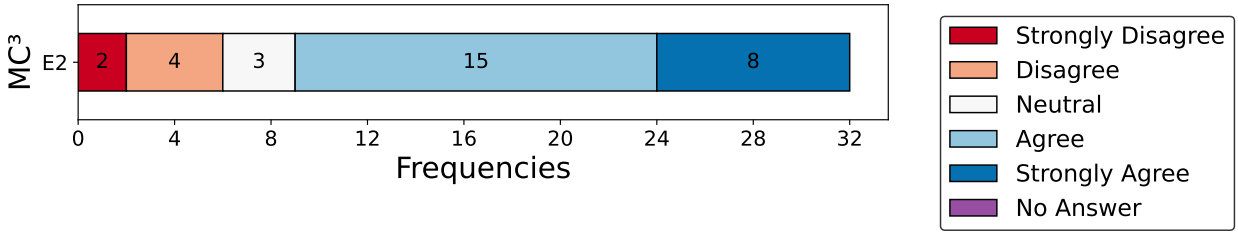


Figure 43: Frequency distribution obtained for the Likert-item severity classification of E2.

Commentaries Analysis: We received 11 comments for this question. The severity concern of this MC³ was classified as high by most respondents. However, the comments stated that the problem happens when the list is only being used once and then discarded, which tends to happen frequently. One respondent also said this behavior is hard to eradicate. On the other hand, those who did not classify this as problematic stated that it is a fair way of separating input from calculation of data.

3.2.5.3 Overall Category Classification

Figure 44 denotes the distribution of the severity concern of all the eight MC³ present in Category E.

3.2.6 Category F: Test cases

This category presents MC³ that are related to coding behavior that emerges from students knowing test cases which the code is subjected to in automatic correction. There are 2 MC³ in total, cataloged

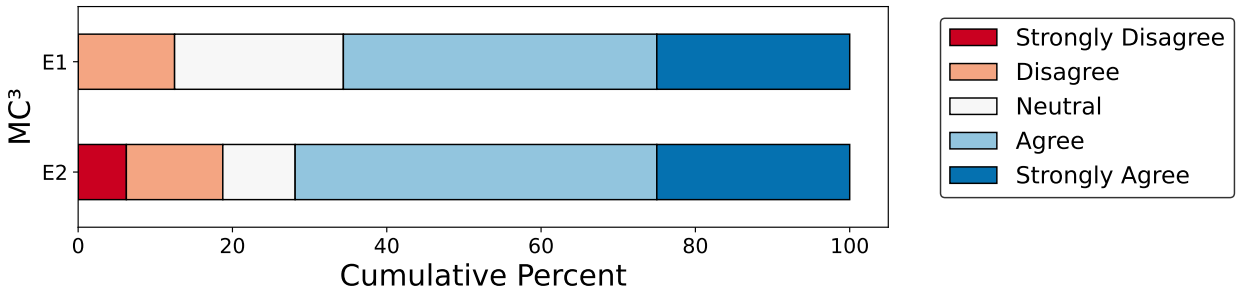


Figure 44: Cumulative percentage of the Likert-item classification for the MC³ in Category E.

as F1 and F2.

3.2.6.1 F1: Verification for non explicit conditions

Original Description: Indicates an occurrence in which the student over verifies conditions for the code, which were guaranteed by the assignment description they will not happen (e.g. it is stated that $0 < N < 100$ but the student creates a check that verifies N is within this range). The only possible issue with this MC³ would be adding conditions that leads to malfunctions in the code, which did not happen in the analyzed code. Nevertheless, this was an observed behavior and its causes might be interesting to investigate. In Algorithm 35, *expr2* and *expr3* are conditions that were not said in the exercise that they could happen in the problem description, but the student decided to verify them anyway.

Algorithm 35: **F1**: Verification for non explicit conditions.

```
1 if expr1 and expr2 and expr3:
2     (...)
3     #expr2 and expr3 were guaranteed by the assignment description
          that they will not happen in inputs
```

Likert-item Classification: Figure 45 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

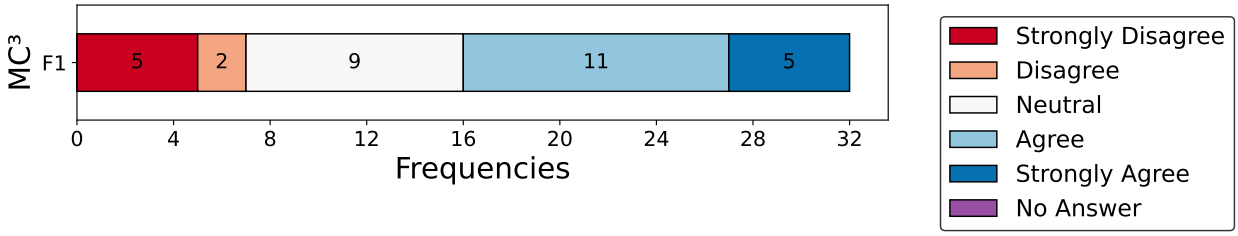


Figure 45: Frequency distribution obtained for the Likert-item severity classification of F1.

Commentaries Analysis: We received 11 comments for this question. Although most respondents classified the severity concern of this MC³ as high, the comments to this question were polarized. Those who rated it as severe stated this is a frequent behavior and suggests that students are not fully understanding how input is done when using automated tests. One respondent also said this

MC³ indicates students might be focusing on passing specific tests rather than generalizing their code. Another respondent said to encourage students to test problematic inputs only when they are not stated in the assignment description. However, the group who did not classify it as severe stated that it is better to program in a defensive manner, since code used in production by other people might be prone to errors. Finally, as one respondent stated he did not understand this MC³ description, we have updated it to be more precise.

3.2.6.2 F2: Specific verification for instances of open test cases

Original Description: Indicates an occurrence in which the student creates the code using conditions directly obtained by reading the test cases that will be used to assess the program. This MC³ could be related to misunderstandings of how the automatic grading systems work, since the student is adapting the code to work in specific conditions. In the worst case scenario, the student could be trying to cheat the system by providing direct answers to the expected results from the test cases. In Algorithm 36, the variable *var1* is compared with values *VAL1*, *VAL2* and *VAL3*, which are values from the test cases which student have access to, as denoted in Algorithm 37.

Algorithm 36: **F2**: Specific verification for instances of open test cases.

```
1  '''VAL1, VAL2, and VAL3 are
       inputs from the test case
       suite'''
2  if var1 == VAL1:
3      print("RES1")
4  elif var1 == VAL2:
5      (...)
6  elif var1 == VAL3:
7      (...)
8  else:
9      (...)
```

Algorithm 37: List of input and expected output for **F2**.

```
#input                  1
VAL1                    2
VAL2                    3
VAL3                    4
...                     5
                        6
#output                 7
RES1                    8
RES2                    9
RES3                    10
...                     11
```

Likert-item Classification: Figure 46 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

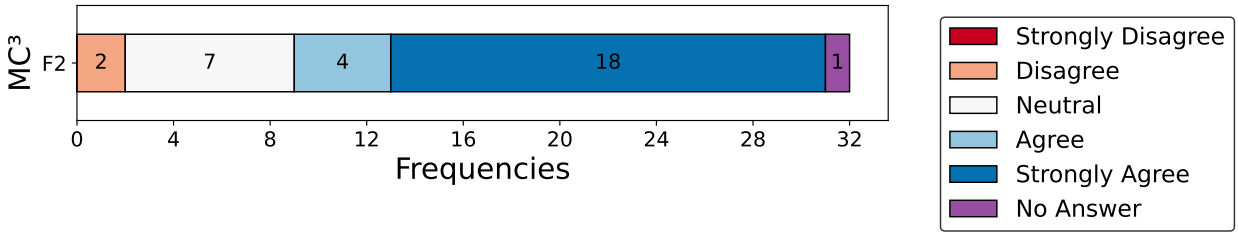


Figure 46: Frequency distribution obtained for the Likert-item severity classification of F2.

Commentaries Analysis: We received 11 comments for this question. The severity concern of this MC³ was rated as high. Most respondents stated this is a frequent behavior in which the students are either not understanding how the automated assessment works or they are trying to

cheat by providing the expected output. One respondent also said this MC³ indicates the student is in trouble because he has no real grasp of the course material. On a side note, as a considerable number of respondents said they did not understand this MC³, we have updated its description and added another algorithm that represents the input and expected output.

3.2.6.3 Overall Category Classification

Figure 47 denotes the distribution of the severity concern of all the eight MC³ present in Category F.

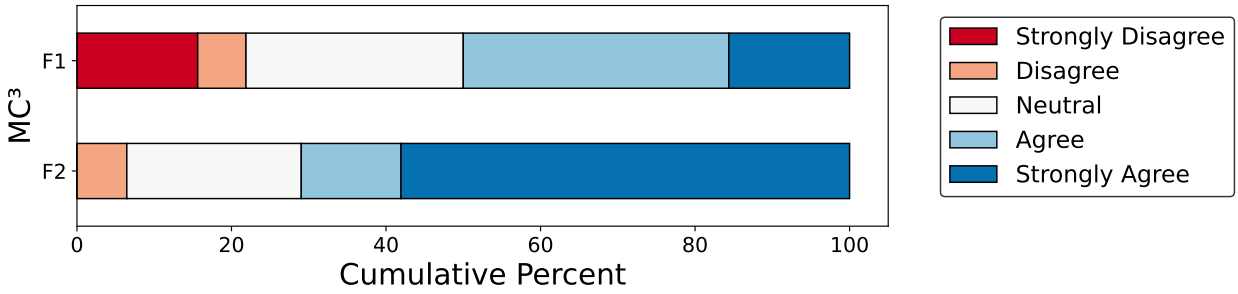


Figure 47: Cumulative percentage of the Likert-item classification for the MC³ in Category F.

3.2.7 Category G: Code organization

This category presents MC³ that are related to code organization, such coding behavior or order of general declarations. There are 6 MC³ in total, cataloged from G1 to G6.

3.2.7.1 G1: Long line commentary

Original Description: Indicates an occurrence in which long commentaries are written in the line format in Python, extending the file horizontally. This MC³ could be related to misunderstandings of the properties of the programming language, since a block commentary style would be better suited in this case. In Algorithm 38, a very long commentary was made using the Python line style commentary format instead of the block style one.

Algorithm 38: **G1**: Long line commentary.

```
1 #a long line commentary that should have used the block format...
```

Likert-item Classification: Figure 48 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 11 comments for this question. Most respondents classified the severity concern of this MC³ as low. In general, the answers stated this behavior is a matter of style and some editors even try to trim it if line commentaries pass a certain number of characters. A group of respondents also said this MC³ would be a problem if the CS1 class strictly adheres to a convention and the student is not doing so. However, another group of respondents said the real problem with comments are students thoroughly describing what a line or small chunk of code does instead of generalizing it in a small phrase.

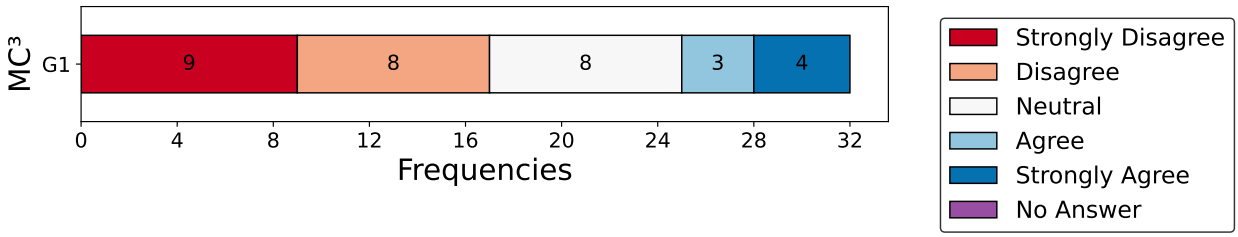


Figure 48: Frequency distribution obtained for the Likert-item severity classification of G1.

3.2.7.2 G2: Exaggerated use of variables to assign expressions

Original Description: Indicates an occurrence in which a high number of variables are declared to be assigned the result of simple expressions, being used further in the code. Although this practice can help to shorten operations that would be long, the exaggerated use can be seen as a MC³ because it could lead to confusion in future reading and maintenance. In Algorithm 39, variables from *var3* to *var7* were assigned the results of simple operations with other variables. Further in the code, those variables are used in conditional checks (lines 7 and 9), forcing the person reading the code to go back and forth to remember what those variables represent.

Algorithm 39: **G2**: Exaggerated use of variables to assign expressions.

```
1  var3 = var1 + var2
2  var4 = var1 - var2
3  var5 = func1(var4, var3)
4  var6 = var5 // var3
5  var7 = var1 and var5 or var3
6  (...)
7  if func2(var3):
8      (...)
9  elif func3(var7, var6)
10     (...)
```

Likert-item Classification: Figure 49 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

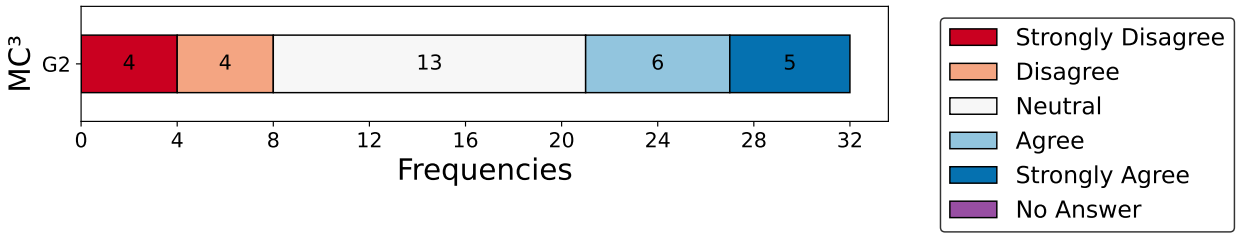


Figure 49: Frequency distribution obtained for the Likert-item severity classification of G2.

Commentaries Analysis: We received 10 comments for this question. Most respondents rated the severity concern of this MC³ as neutral. In general, respondents commented that if those variables are significant and well named, the code will become clear, therefore not being a concern.

Respondents also said this MC³ is infrequent, but students who do this are not likely to change this behavior.

3.2.7.3 G3: Too many declarations in a single line of code

Original Description: Indicates an occurrence in which many operations, distinct or not, are declared in the same line of code. Similar to G1, this MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 40, a *for* loop and a ternary *if* statement were declared in a single line of code. Although it makes code more compact, it will also make it more difficult to read.

Algorithm 40: **G3**: Too many declarations in a single line of code.

```
1 for iter in lst: func1(iter) if iter%2==0 else func2(iter,iter+1)
```

Likert-item Classification: Figure 50 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

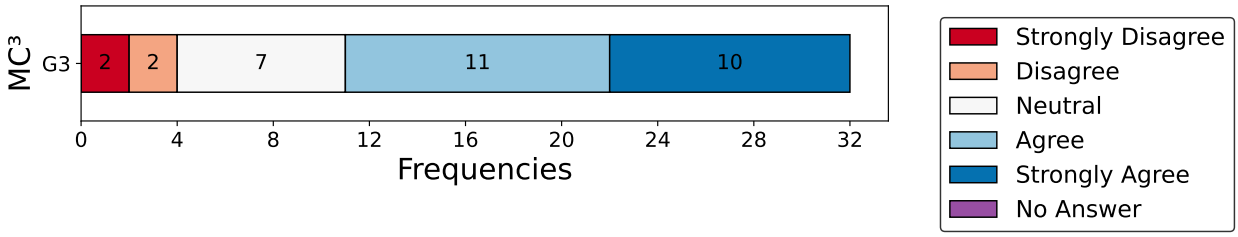


Figure 50: Frequency distribution obtained for the Likert-item severity classification of G3.

Commentaries Analysis: We received 8 comments for this question. The severity concern of this MC³ was classified as high. Respondents stated this kind of construction is prone to bugs and is not clear for others to read and understand it. Two respondents also said they encourage students to not do this even though it can be useful sometimes.

3.2.7.4 G4: Functions/variables with non significant name

Original Description: Indicates an occurrence in which the functions and/or variables are declared with arbitrary, non significant names that don't represent their purpose. This MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 41, variables a, b, x, y and z were assigned the results of functions. This type of issue generally happens with the declaration of subsequent alphabetical letters as variables.

Algorithm 41: **G4**: Functions/variables with non significant name.

```
1 a = func1(var1)
2 b = func1(var2)
3 x = func2(a, b)
4 y = func3(x)
5 z = func4(x, y)
```

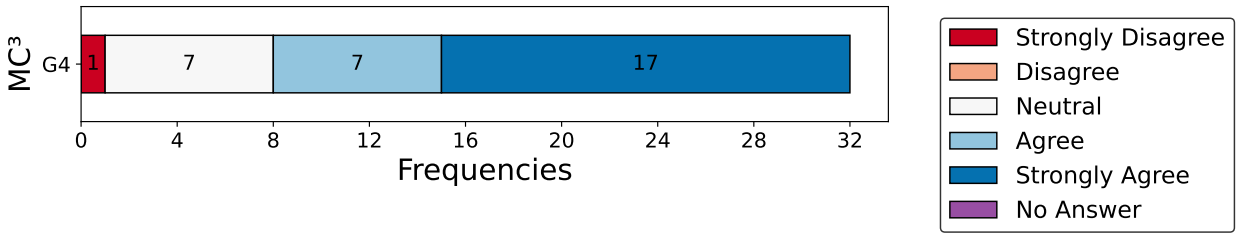


Figure 51: Frequency distribution obtained for the Likert-item severity classification of G4.

Likert-item Classification: Figure 51 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 12 comments for this question. Most respondents rated the severity concern of this MC³ as high. Respondents vehemently agreed this is an important concept that should be taught as soon as possible in CS1. The only exceptions for this behavior to be accepted, according to the answers, are when the student is drafting his code or when used in small code sections that are simple enough to understand.

3.2.7.5 G5: Arbitrary organization of declarations

Original Description: Indicates an occurrence in which the declarations of variables and functions are made arbitrarily throughout the code. The arbitrary order of declarations regarding this MC³ could lead to confusion in future reading and maintenance of the code. In Algorithm 42, variable *var1* is declared and assigned a value in line 1, followed by a function *func1()* declaration in line 3. After that, more variables are declared, followed again by another function *func2()*. Functions, input variables, and the rest of the code are declared in arbitrary order.

Algorithm 42: **G5**: Arbitrary organization of declarations.

```
1  var1 = input()
2
3  def func1():
4      (...)
5
6  var2 = input()
7  var3 = func1(var1, var2)
8
9  def func2():
10     (...)
11
12 var4= func2(var3, var1)
13 var5 = input()
```

Likert-item Classification: Figure 52 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 8 comments for this question. Most respondents classified this MC³ with a high severity concern. According to the comments, this MC³ indicates a lack of

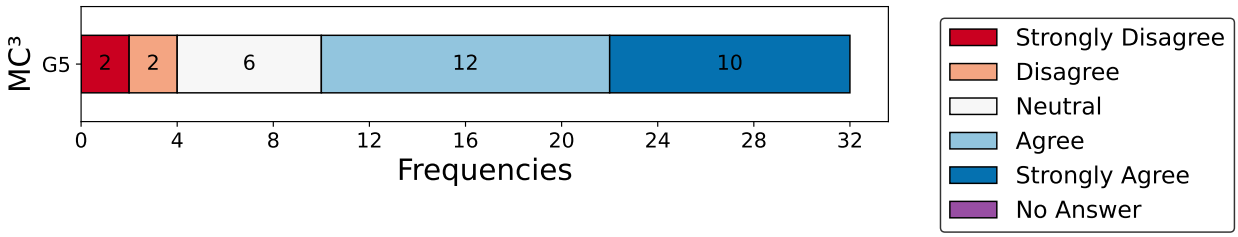


Figure 52: Frequency distribution obtained for the Likert-item severity classification of G5.

clarity in students' reasoning, which is expressed in their code. A group of respondents also said this is a frequent behavior, although it tends to disappear after a lecture in which good practices is commented upon. Those who did not classify this MC³ as severe stated they have either not seen it, or it is dependent on the programming language.

3.2.7.6 G6: Functions not documented in the Docstring format

Original Description: Indicates an occurrence in which the declared functions are not properly documented, or documented at all using the Python Docstring format. This MC³ could be related to the fact that students might not know that Docstring exists, and/or they are oblivious to the concepts of code quality. In Algorithm 43, the declared functions *func1()* and *func2()* have some code logic that is not very clear and lack documentation in any form. The function *func3()* has some commentaries about its logic but it is not documented in the Docstring format.

Algorithm 43: **G6**: Functions not documented in the Docstring format.

```
1  def func1():
2      (...) #code that is not clear
3
4  def func2():
5      (...) #more code that is not clear
6
7  def func3():
8      #some commentary about func3() but not in Docstring format
9      (...)
```

Likert-item Classification: Figure 53 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

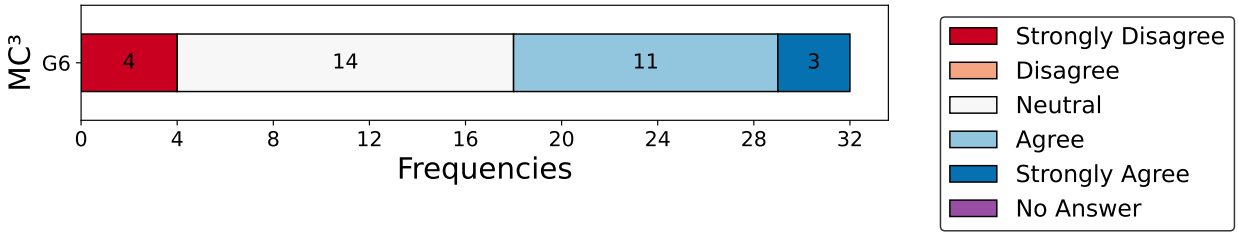


Figure 53: Frequency distribution obtained for the Likert-item severity classification of G6.

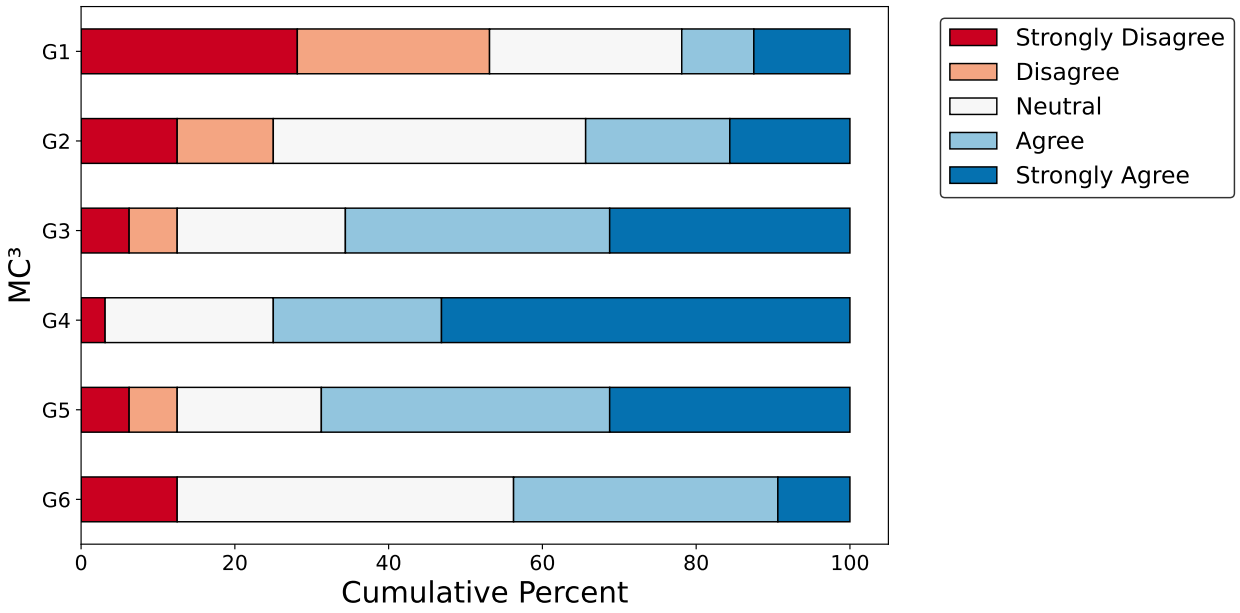


Figure 54: Cumulative percentage of the Likert-item classification for the MC³ in Category G.

Commentaries Analysis: We received 9 comments for this question. The severity concern classification of this MC³ ranged from neutral to high. Respondents stated that documentation should be properly addressed in a lecture, and they do require their students to do so, although not necessarily using the Docstring format. Once again, a group of answers stated that the problem is the quality of the comments (specifying what code that is not obvious does or making the comment concise and general) rather than their format.

3.2.7.7 Overall Category Classification

Figure 54 denotes the distribution of the severity concern of all the eight MC³ present in Category G.

3.2.8 Category H: Other

This category presents MC³ that are not fit to any other previous stated category. There are 3 MC³ in total, cataloged from H1 to H3.

3.2.8.1 H1: Statement with no effect

Original Description: Indicates an occurrence in which statements that doesn't affect the code execution are declared in the program, varying from literal numbers to not assigning the returning result of a function or method to a variable. Some cases, such as *loose numbers* in the code, could possibly be resulting from distractions. However, this MC³ could also be related to misunderstandings about functions or methods that does not update a parameter variable since its updated value result needs to be returned. In Algorithm 44, a number **4** was declared in line 2, having no effect in the code execution. Aside from that, the *round(var1)* declared in line 3 also has no effect, since the result is not assigned to any variable.

Algorithm 44: **H1**: Statement with no effect.

```
1 var1 = func1()
2 4
3 round(var1)
```

Likert-item Classification: Figure 55 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

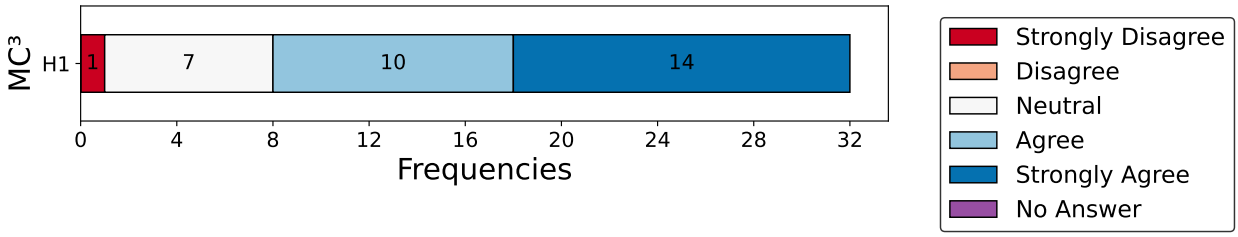


Figure 55: Frequency distribution obtained for the Likert-item severity classification of H1.

Commentaries Analysis: We received 11 comments for this question. Most respondents classified the severity concern of this MC³ as high. Responses stated this behavior indicates misunderstandings that students might be having, possibly leading to bugs. Respondents also said this is frequent in the first few weeks of the CS1 course and, if by the end of the term a student is still programming like this, it is a sign that he is close to failing the course. One respondent said that he usually sees only cases in which the updated value of the variable is not assigned.

3.2.8.2 H2: Redundant typecast

Original Description: Indicates an occurrence in which a typecast is applied to the result of a function or expression of the same type. This MC³ could be related to misunderstandings of basic properties of Python. In Algorithm 45, *input()* returns a string that is unnecessarily typecasted to the string format (line 1). The same goes for the result assigned to the variable *var2*, as the sum of two integers is an integer.

Algorithm 45: **H2**: Redundant typecast.

```
1 var1 = str(input())
2 var2 = int(6 + 4)
```

Likert-item Classification: Figure 56 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

Commentaries Analysis: We received 5 comments for this question. In general, the severity concern of this MC³ was classified as high. Respondents stated this MC³ is frequent and indicates some misunderstandings of the basic functionalities of Python. They also said that it is done by students with previous coding experience in other languages.

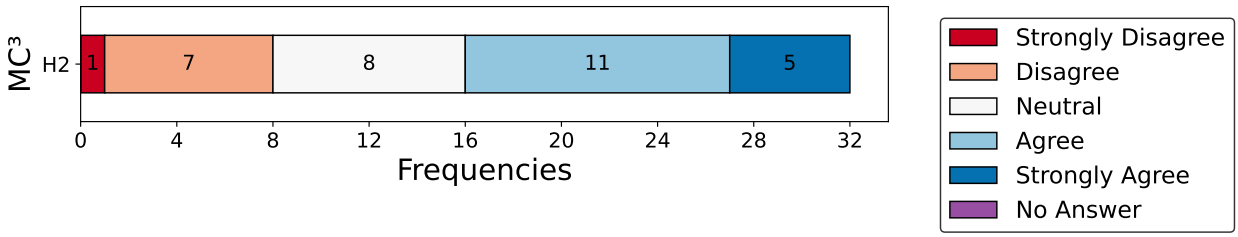


Figure 56: Frequency distribution obtained for the Likert-item severity classification of H2.

3.2.8.3 H3: Unnecessary or redundant semicolon

Original Description: Indicates an occurrence in which a semicolon is used unnecessarily, typically at the end of a statement or declaration in a line of code. This MC³ could be related to misunderstandings of the syntax of the programming language, probably because the student has previous knowledge of other languages. In Algorithm 46, a semicolon was typed at the end of the statement in line 1. Since it's a single declaration in Python, it was not needed.

Algorithm 46: **H3**: Unnecessary or redundant semicolon.

```
1 var3 = var2 - var1;
```

Likert-item Classification: Figure 57 shows the frequency distribution of the 32 responses to the statement "I believe this MC³ is of great concern".

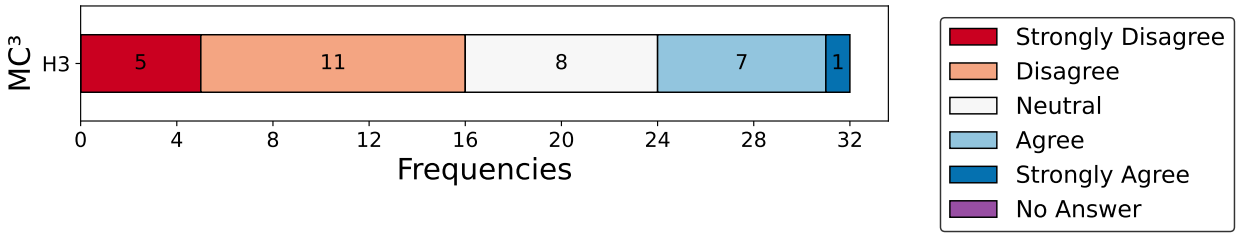


Figure 57: Frequency distribution obtained for the Likert-item severity classification of H3.

Commentaries Analysis: We received 7 comments for this question. Most respondents rated the severity concern of this MC³ as low. In general, responses agreed this behavior is consequence of the student having experience in other programming languages and will naturally disappear as the student gains more experience in Python.

3.2.8.4 Overall Category Classification

Figure 58 denotes the distribution of the severity concern of all the eight MC³ present in Category H.

4 Interviews Analysis

In this section we present the data collected with the semi structured interviews. As mentioned in Section 2, we managed to interview 9 out of 18 respondents who volunteered to participate in

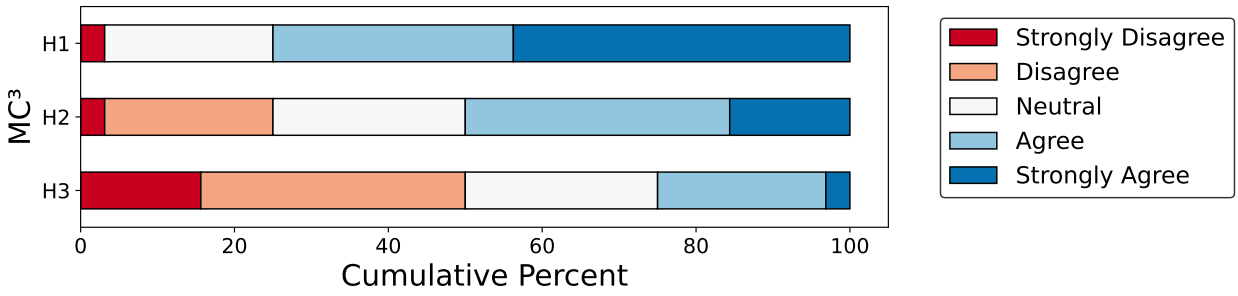


Figure 58: Cumulative percentage of the Likert-item classification for the MC³ in Category H.

this second phase of the study. 7 instructors were from Brazil and 2 from the USA. The average duration of the interviews was 42 minutes.

We conducted the interviews with Brazilian instructors in Portuguese. To provide our compilation of questions and answers, we translated the questions and answers into English in this report. To anonymize the respondents, we named them R1 to R9. We provide their answers in subsection 4.1, an overview of all the interviews in subsection 4.2, and the obtained insights from the interviews in subsection 4.3.

4.1 Questions and Answers

We conducted all the interviews in a semi structured manner. Even though we had prepared a set of questions, we slightly changed them as we did more interviews. Since we let the interviewees do most of the talk, sometimes while they were answering a question, they provided answer to a future question in advance. When this situation had happened, we still asked the already answered question to see if there was any other information the instructors could share. On a final note, as during the interviews the MC³ were still called Programming Issues, we decided to maintain this naming in the questions, but not in the compiled answers.

4.1.1 R1

Date: 25 March 2022.

1. **Introduction**

   Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

   R1 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

   R1 stated to use Python in his classes by his own initiative. He also said he does not intend to change the programming language, although he has interest in teaching Julia in the future. Finally, R1 said that Java was the dominant language in the institution he works, but its use has decreased.

4. **Structure of the CS1 courses you teach:**

   (a) **Is it split into theory and practice classes?**
   R1 said that his course is composed by 5 hours per week: 3 hours in the classroom and 2 hours in the laboratory. He administers all classes by himself since there are no teaching assistants. R1 also mentioned that the practical classes have approximately 30 students.

   (b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
   R1 said to divide the course in three parts: in the first he uses small examples, executing step-by-step in the blackboard, and talks about conceptual errors found in these examples. In the second he assigns tasks, which are solved by the students, and he corrects them manually. These assignments are summative. In the final part, he uses Online Judges such as *Euler* and *URIOnlineJudge* (now known as *beecrowd*). R1 assigns simple exercises from these sources and asks the students to show their code passes in the test cases.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R1 had already responded this in the previous question. He complemented that it is by his own will since the institution he works does not have any kind of system like this.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   R1 complemented his answer saying that he only requests some proof that the students' code passes the test cases. During the first and second part of the course, he walks around the laboratory and eventually makes commentaries about the student solutions he sees.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R1 said that autograders are helpful because they facilitate code assessment in classes with many students. He also believes that students should gain experience in using these systems. However, instructors cannot put their whole confidence in the autograders yet. R1 stated that sometimes, to pass the test cases, the programmer tends to do anything, even code with some "atrocities".

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   R1 said to read the solutions of students to find any big mistakes. However, he also said that he does not worry too much since the code has passed the online judges' tests.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R1 mentioned that he focuses on organizational and structural details of the code. He only penalizes students' solution regarding good practices and legibility at the end of the course. However, R1 stated that he does not penalize code in terms of efficiency, such as running time.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R1 stated that he does explain the difference between for and while, and nested ifs and Boolean expressions, but he does not explicitly ask any assignment to use a specific structure. However, in the case of recursion versus iteration, he considers the solution wrong if they do not construct it with the desired concept.

8. **Do you know any other programming behavior, similar to a Programming Issue [MC^s], which was not present in the questionnaire, and you would like to mention?**

   R1 commented about students using modules or built-in functions that solves part of the assignment. In these cases, he only allows their usage if the student can answer how these functions work (which does not tend to happen). He also commented on students using temporary variables as conditional for if structures.

9. **Consider an autograder system that is also able to detect the Programming Issues [MC^s] studied in this research:**

   (a) **Would you be interested in using it in your CS1 classes? How?**

   R1 said that he would try to adapt to the tool at first. If he felt comfortable, he would use the autograder in the third part of his course using the same methods already applied.

   (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**

   R1 commented that he does not see a problem to provide feedback to the students if it is also shown to the instructor. He believes that it should be configurable.

   (c) **Any other suggestions of the type of feedback the system should provide?**

   R1 commented about presenting links in which the student could learn more. However, he said that the students might only click on it once, at best. Therefore, they might be useful, but not too much.

10. **Active Learning methodologies:**

    (a) **Do you have familiarity with them?**

    Since R1 mentioned to not have familiarity with Active Learning, Eryck [Silva] explained about them and about the Peer Instruction technique.

    (b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^s], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^s] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts:**

the question aims at identifying code snippets that have Programming Issues [MC^3]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.

i. **Would you be interested in using this technique in your CS1 classes?**
R1 said that he would try this approach in his classes to see how the students would react to these questions. He also stated that he would intercalate questions regarding general concepts with those related to the MC^3.

ii. **Do you have any other suggestions for the use of this technique?**
R1 briefly compared the technique with gamification, but he did not delve into the topic.

11. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^3] in CS1 courses?**

R1 mentioned about machine learning techniques that could be used to provide feedback in autograders. He also said that, in classroom, examples of code with and without MC^3 should be shown to the students. Finally, he stated that instructors should also assign interesting exercises and good readings.

4.1.2 R2

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

R2 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

R2 commented that he only lectures Python theory classes. The laboratory hours are dedicated to the teaching assistants to help students in doing their assignments. R2 also stated that he solves exercises during his theory classes.

4. **Structure of the CS1 courses you teach:**

(a) **Is it split into theory and practice classes?**
R2 said that there are two types of assignments: fixation exercises and laboratory assignments. He elaborates and corrects the exercises while the teaching assistants do the same for the laboratory. Both types count towards the final grade. R2 also mentioned that after students received their evaluated assignments, they can submit them a second time to score a better grade.

(b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
R2 said that there are two types of assignments: fixation exercises and laboratory assignments. He elaborates and corrects the exercises while the teaching assistants does the

same for the laboratory. Both types count towards the final grade. R2 also mentioned that students can submit the assignments one more time to see if they can score a better grade.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R2 said to use a variety of autograders in his CS1 classes, such as: the one provided by the institution he works; run.codes; and codePost.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   R2 stated that even though these systems are used to correct the students' code, the teaching assistants check them manually after the initial evaluation.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R2 mentioned that autograders provide a scalable way of evaluating code from classes with many students. However, he said those systems have a binary assessment: the code is classified as right or wrong, and they also do not detect simple characteristics such as the ones Flake8[4] can detect.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   As answered earlier, R2 said that the teaching assistants check every code, even those that receive the maximum grade.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R2 said that he does penalize students' code regarding these questions specifically because they are allowed to submit their code again after the initial submission is graded. By doing this, he expects the students to evaluate if it is worth or not to redo the assignment to score a better grade.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R2 stated that he does not care about the usage of *for* and *while*, since they are dependent on the students' rationale. However, he thinks differently about Boolean expressions versus nested *if's*, and iteration versus recursion. If the assignment specifies the use of one structure, the students must elaborate their code with it because it is important to assess if they understood the concepts.

[4] `https://flake8.pycqa.org/en/latest/`

8. **Do you know any other programming behavior, similar to a Programming Issue [MC^s], which was not present in the questionnaire, and you would like to mention?**

   R2 said that students tend to misunderstand the concepts of functions and modularization when they create only one function that represents almost the whole code. He also said about behaviors related to inconsistencies in code style, such as spaces between characters and lines. R2 also mentioned that students tend to use lists even when other taught data structures would be better used, such as dictionaries. In terms of sorting algorithms, students tend to only remember Bubble Sort even though more efficient ones were taught.

9. **Consider an autograder system that is also able to detect the Programming Issues [MC^s] studied in this research:**

   (a) **Would you be interested in using it in your CS1 classes? How?**
   R2 said that he would be interested in using a system that has this characteristic. He did not mention any specific way in which he would use it, though.

   (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
   R2 stated that he does not see a problem in providing all feedback to the students. He suggested that there can be a priority: if the code does not compile, there would be no need to also inform about efficiency nor good practices. The best way would be the configuration of these settings by the instructor.

   (c) **Any other suggestions of the type of feedback the system should provide?**
   R2 suggested feedback that does the following checks: the presence of a requested function; the code complexity in terms of nesting levels (which could be configured by the instructor); suggestion to refactor repeated code sections; and code styling, such as linters and Flake8. R2 also commented about the ability for the instructor to provide feedback directly through the system, such as the commenting in Google Docs. He also said about the possibility of automatically generating test cases to ease the workload of the teaching assistants.

10. **Active Learning methodologies:**

    (a) **Do you have familiarity with them?**
    Since R2 mentioned to have familiarity with Active Learning and the Peer Instruction technique, Eryck [Silva] did not explain about it and moved to the next question.

    (b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^s], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^s] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^s]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

i. **Would you be interested in using this technique in your CS1 classes?**
R2 mentioned that he would be interested in using this technique in his classes, although there is some concern regarding the elaboration of the multiple-choice questions. He believes that since time is usually short to administer all the required concepts, code quality tends to be a topic that cannot have too much allocated time in class.

ii. **Do you have any other suggestions for the use of this technique?**
R2 said that it would be better to present this technique after the students gained experience with the taught concepts and not right after their explanation in class. He also said that Active Learning techniques would be better used in a single class dedicated to good practices in coding.

11. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^{8}] in CS1 courses?**

R2 mentioned about exposing a bad example of code in a class and then refactor it step-by-step with the students. He said to have some concerns if it would be useful to use bad examples from the students, since it might cause discomfort among them. Nevertheless, he stated that students should be exposed to good and bad examples of code to understand what they should or not be doing.

4.1.3 R3

Date: 28 March 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

R3 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

R3 stated that all CS1 courses from the institution he works are using only Python.

4. **Structure of the CS1 courses you teach:**

(a) **Is it split into theory and practice classes?**
R3 commented that there are both theory classes in the classroom and practice classes in the laboratory.

(b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
R3 stated that there are assignments, but the grading is dependant on the instructors. Some consider the assignments part of the final grade; others credit them as extra; and others do not credit any grade to them. R3 said that, in general, the instructors are responsible for the elaboration and evaluation of all the assignments. Some instructors use exercises from Online Judges. In this case, they decide whether to use or not their automated correction.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**

   R3 said that initially he used these systems only to select exercises by a specific topic such as loops. Then he used run.codes in a graduate course because the system can show the input and expected output from the test cases.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**

   R3 detailed about the experience using it in the graduate course. He noted that some students did elaborate their solutions to only print the expected output for each input (like the F2 MC^3). R3 stated that, because of this, he started to read each solution even if it passed the test cases.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**

   R3 mentioned that these systems do help the instructor in classes with many students. However, he also said that there are some cases in which the instructor must manually check the solutions to confirm if the students are doing what is expected.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**

   R3 commented that because he does check all solutions, he thinks it is a massive task, even if the students do the assignments in groups. R3 also commented that he believes other instructors from the institution he works do not check the submissions if they pass the test cases (if the instructors use this functionality from the autograders).

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R3 said that he does not penalize students' solution regarding these questions. However, he comments on them whenever he notes situations like these in the students' code. R3 also stated that this only happens if he can note these behaviors, and he recognizes that this identification is dependent on his opinion of what a bad behavior might be.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R3 stated that if the instructor is interested in assessing if the students have grasped the concepts of a specific structure, he must explicitly say it in the assignment. R3 also said that he believes that the instructor would have to check manually if the structures are present in the solutions.

8. **Do you know any other programming behavior, similar to a Programming Issue [MC^3], which was not present in the questionnaire, and you would like to mention?**

   R3 did not state any new programming behavior. He did comment on having seen some of the already listed: the Boolean comparison attempted with while loop (B6), and redundant or unnecessary loop (C2).

9. **Consider an autograder system that is also able to detect the Programming Issues [MC^3] studied in this research:**

   (a) **Would you be interested in using it in your CS1 classes? How?**
   R3 said that he would be interested in using the system because he believes it would considerably help the instructors. He commented that, sometimes, even if they manually check all the solutions, they are prone to not see everything.

   (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
   R3 said that, initially, the system should only show the feedback to the instructor, since it might be too much for the student. By doing this, the instructor will find the best way to help the student with each occurrence. As the course progresses, the system can show the feedback to the students.

   (c) **Any other suggestions of the type of feedback the system should provide?**
   R3 suggested that the system should show good examples of code whenever it detects a MC^3. However, he stated that if the system can pinpoint the location and the identification of the MC^3, it should already be enough to help both instructor and student.

10. **Active Learning methodologies:**

    (a) **Do you have familiarity with them?**
    Since R3 mentioned to have little knowledge of Active Learning methodologies, Eryck [Silva] explained about them and the Peer Instruction technique.

    (b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^3], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^3] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^3]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

       i. **Would you be interested in using this technique in your CS1 classes?**
       R3 commented that he would be interested in using this technique because he thinks students like to discuss. He mentioned that, during the remote classes, situations like this happened when he had asked students the reason why they answered a question in a certain way and they started discussing among themselves.

ii. **Do you have any other suggestions for the use of this technique?**
R3 suggested that the instructor should not tell that there are code with MC^3 in the multiple-choice questionnaire. Instead, he should let the students find out what bad behaviors might be present in the choices. R3 also stated that this approach could be used by dividing the students into groups during the laboratory classes.

11. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^3] in CS1 courses?**

R3 did not state any other methodology. He said that sometimes he wonders on how to balance the teaching of concepts with good practices and code efficiency in his CS1 classes. R3 commented that he thinks it might be better to let the students code pass the tests at first. By the end of the CS1 course they should be taught about good practices and let future courses continue working these concepts with them.

4.1.4 R4

Date: 31 March 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

R4 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

R4 said that the institution where he works began using Python in the last semester. Before that, they had used C and a small part of C++.

4. **Structure of the CS1 courses you teach:**

(a) **Is it split into theory and practice classes?**
R4 commented that there are both theory and practice classes. However, the practice classes are destined to help the students who have questions about the assignments.

(b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
R4 mentioned that there are assignments that counts toward the final grade. The students submit their solutions via an autograder, created by the institution. The same system does the automated grading based on test cases.

5. **Utilization of automatic grading systems:**

(a) **Do you use any kind of these systems in your CS1 course?**
R4 said that he uses the autograder developed by the institution where he works. R4 also stated that the previous versions of the CS1 course (in which C and C++ was taught) already used the same system.

(b) **Is the system used only to evaluate the output of the students' code by checking test cases?**

R4 stated that the system is only used to evaluate the output of the students' code. The grade is based solely on the number of passed test cases. In addition, he said that the system is also used for the application of exams.

(c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**

R4 mentioned that the students complain that the system does not explain why their code does not pass the test cases. He considers the immediate feedback as an advantage, but the lack of details when an error happens obfuscates this advantage. R4 also stated that he thinks the system could provide feedback from static analysis of the code, thus providing information on code quality and styling to the students.

(d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**

R4 had answered this question. He does not check the submissions after the system grades them.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R4 stated that he thinks these are important concepts that should be taught to the students. However, since the CS1 course follows the same guidelines for all instructors and because of the use of the autograder, he cannot penalize students' solutions regarding these concepts. While he does comment upon these concepts in his classes, R4 said if he could, he would credit extra points to code solutions that follow good practices, instead of penalizing those that does not.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R4 said that instructors might be interested in this scenario to assess if the students had learned a specific concept. However, in the context where he teaches it would not be feasible because of the use of the autograder.

8. **Do you know any other programming behavior, similar to a Programming Issue [MC^3], which was not present in the questionnaire, and you would like to mention?**

   R4 did not mention any behavior that resembled a MC^3.

9. **Consider an autograder system that is also able to detect the Programming Issues [MC^3] studied in this research:**

(a) **Would you be interested in using it in your CS1 classes? How?**
R4 stated that he would be interested in using the system, especially because he had been looking for similar approaches in the past. However, he also said that the adoption would be dependent on how well the system could be integrated to the one used in the institution he works.

(b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
R4 said that providing all feedback directly to the students can be massive and not useful. He suggested that the system should detect the most frequent and severe MC^3 and show only them to the students. The instructor should also comment about these MC^3 in the classroom.

(c) **Any other suggestions of the type of feedback the system should provide?**
R4 suggested that the system should also provide guidance to correct the MC^3 found in the students' code.

10. **Active Learning Methodologies:**

(a) **Do you have familiarity with them?**
Since R4 mentioned to have little knowledge of Active Learning methodologies, Eryck [Silva] explained about them and the Peer Instruction technique. During the explanation, R4 stated to use similar approaches to the Peer Instruction albeit not in CS1 classes.

(b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^3], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^3] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^3]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

i. **Would you be interested in using this technique in your CS1 classes?**
R4 stated that he would be interested in using this technique in his CS1 classes. However, he thinks it might not be enough for the students to understand about the MC^3 because it could become information overload. Regarding this, R4 said that the learning would be reinforced by the MC^3 feedback from the autograder.

ii. **Do you have any other suggestions for the use of this technique?**
R4 did not state any other suggestions for this technique.

11. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^3] in CS1 courses?**

R4 mentioned about game development by the students. He believes this is an interesting project because they would need to produce more robust code, and this kind of code would be interesting to assess in terms of MC^3.

4.1.5 R5

Date: 4 April 2022.

1. **Introduction**

   Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

   R5 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

   R5 said that Python is used in the institution he works. However, he stated that the institution is in an evaluation period, and it might be changed.

4. **Structure of the CS1 courses you teach:**

   (a) **Is it split into theory and practice classes?**
   R5 commented that theory and practice classes are mixed. As they happen in the laboratory, the instructor presents the concepts, and the students are asked to replicate and change the code examples.

   (b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
   R5 commented about final projects that the students present in a seminar format at the end of the course. The instructor is responsible for the elaboration and execution of the rubrics for these projects.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R5 said that the use is dependent on the instructor. In particular, he likes to use them although he thinks these systems are limited to output verification.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   R5 stated that he has been elaborating approaches to help him to analyze students' solutions albeit only to detect issues. However, he must parse the solutions himself while using these approaches. The grading is done manually by him.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R5 said that these systems still do not provide a thorough static and dynamic analysis of code. He mentioned that these systems do not assess programming styles nor variable names, for example.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   R5 had already answered that he uses systems (created by himself) to help detecting characteristics present in the code. However, he decides how the final grading will be.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R5 said that he thinks these questions should be taught since the beginning of the CS1 course. He particularly commented about students with previous programming experience that develop bad behaviors that should be addressed sooner than later. R5 stated that in the first half of the course he only comments when he sees code without good practices. However, since code quality is part of the objectives of the final project, he penalizes solutions that do not meet it.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R5 stated that this scenario depends on the learning objectives set by the instructor. He said that he considers the iteration versus recursion as a wrong solution if the student does use the specified one.

8. **Do you believe that those non desirable behaviors in CS1 students' code, such as the ones studied in this research, tend to disappear by themselves in the future or the students need proper feedback regarding them from the instructor in order to mitigate those behaviors?**

   R5 commented that students struggle with questions regarding code quality even when the instructor actively gives feedback about them.

9. **Do you know any other programming behavior, similar to a Programming Issue [MC^s], which was not present in the questionnaire, and you would like to mention?**

   R5 mentioned about functions and variables with names that are not significant, unnecessary comparisons, and unnecessary decision structures used in code (all have been mapped in this study). He also said that students cannot fail the CS1 course because of these behaviors, so they tend to repeat them in future courses.

10. **Consider an autograder system that is also able to detect the Programming Issues [MC^s] studied in this research:**

    (a) **Would you be interested in using it in your CS1 classes? How?**
    R5 said that he would be interested in using it, especially because he already tries to create systems to help with the evaluation of code.

    (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
    R5 stated that the instructor should be able to configure the number and type of feedback that the students receive. This configuration can help limiting students who are trying

to solve the exercise by trial and error. He also said that the number of submissions per student should also be limited.

(c) **Any other suggestions of the type of feedback the system should provide?**
R5 suggested that the system could also provide guidance to correct the MC^3 found in the students' code.

11. **Active Learning Methodologies:**

(a) **Do you have familiarity with them?**
Since R5 mentioned to have familiarity with Active Learning and the Peer Instruction technique, Eryck [Silva] did not explain about it and moved to the next question.

(b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^3], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^3] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^3]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

i. **Would you be interested in using this technique in your CS1 classes?**
R5 said that this is an interesting technique and he would use it. He stated to use a similar approach in class: he presents a solution to an exercise and asks how it could be improved. R5 also said that he brings students' solution to coding challenges to the classroom and does the same improvement analysis. In this case, the students discuss among themselves whether they accept or not the changes in the code.

ii. **Do you have any other suggestions for the use of this technique?**
R5 mentioned that the instructor should use this technique after the students are familiar with the concepts.

12. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^3] in CS1 courses?**

R5 commented that this final project is done in groups of students in which each one of them has a role. The students are trained for this presentation since the beginning of the course. The roles vary from coding, explaining the construction of the code, answering questions from the other students, among others. The roles are defined at the time of the presentation. R5 argued that programmers should not only know how to program, but also how their code interacts with real world problems and how to explain pieces of their code.

4.1.6 R6

Date: 11 April 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

   R6 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

   R6 said that the institution he works uses Python or Octave, depending on the undergraduate's course. Personally, he thinks Python is best for introductory courses, but he cannot use it in all classes he teaches.

4. **Structure of the CS1 courses you teach:**

   (a) **Is it split into theory and practice classes?**
   R6 said that each undergraduate course has a specific structure. One divides it in theory and practice classes in which the instructors are not the same. Another uses a mixed approach with theory and practice taught in the laboratory.

   (b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
   R6 said that he elaborates online questionnaires using Moodle. These questionnaires involve theoretical and coding exercises. Moodle and its CodeRunner module do the automated correction of the questions.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R6 stated that he uses the CodeRunner module for Moodle in his classes.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   R6 commented that for most assignments, he lets the system do the automatic grading. However, for complex assignments and for the exams, he checks every submission manually after they are graded by the system.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R6 said to like autograders, although he does not have much experience with them. He also mentioned that one advantage of these systems is the immediate feedback because students that might be having trouble with the course start to ask for help sooner. When he did not use autograders, these students tended to ask for help just before the exams. R6 commented that the disadvantage is the fact that these systems only check the code output.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   R6 had already answered this question before. He explained that he only sees submissions for exams and complex assignments to assess if the autograder was too strict, or the solution deviates too much from what was asked. In these situations, he tries to adjust the grading accordingly.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R6 stated that these concepts regarding code quality are taught in the classes. However, it depends on the undergraduate course's policy: some credit or penalize solutions that have (or not) flaws in code quality, while others do not.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R6 said that this scenario is possible, and he elaborates assignments that explicitly asks for the use of a specific structure. However, as he does not have a tool that does this verification, he does the check manually.

8. **Do you believe that those non desirable behaviors in CS1 students' code, such as the ones studied in this research, tend to disappear by themselves in the future or the students need proper feedback regarding them from the instructor in order to mitigate those behaviors?**

   R6 said that he believes these behaviors tend to be corrected in future courses because other instructors will address them. However, he said that feedback about it could be helpful in CS1 courses. R6 also stated that the focus on good programming practices depends on the undergraduate course. This is a factor that must be considered whether you want to request that students write good quality code besides learning how to program.

9. **Do you know any other programming behavior, similar to a Programming Issue [MC^s], which was not present in the questionnaire, and you would like to mention?**

   R6 mentioned students trying to cheat a specified way he wants them to use to solve the assignment. An example is using multiple if statements rather than iteration. To eliminate this, he explicitly says what is allowed or not in the assignment description.

10. **Consider an autograder system that is also able to detect the Programming Issues [MC^s] studied in this research:**

    (a) **Would you be interested in using it in your CS1 classes? How?**

    R6 said he would be interested in using. He compared the system to Tonny, which is a Python IDE that uses machine learning to help the programmer. Despite using Tonny, R6 said that he did not see any advantage in CS1 courses.

    (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**

R6 commented that it would be best if the instructor can configure the amount of feedback depending on the students' skill level. He argued that if the students are overwhelmed, they can give up on understanding the feedback.

(c) **Any other suggestions of the type of feedback the system should provide?**

R6 suggested techniques that are related to machine learning, such as comparing the solutions with good and bad examples of code to provide feedback about code quality. He also mentioned about lessening the strictness level of the expected output, like not counting the exact number of white spaces between the printed messages.

11. **Active Learning Methodologies:**

(a) **Do you have familiarity with them?**

Since R6 mentioned to not have familiarity with Active Learning, Eryck [Silva] explained about them and about the Peer Instruction technique.

(b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^s], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^s] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^s]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

i. **Would you be interested in using this technique in your CS1 classes?**
R6 stated that he would be interested in using, but he is afraid of the time it would require. The adoption would only be in classes that can afford the time.

ii. **Do you have any other suggestions for the use of this technique?**
R6 did not make any other suggestions.

12. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^s] in CS1 courses?**

Eryck [Silva] skipped this question because of the allotted time for the interview.

4.1.7 R7

Date: 27 April 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

R7 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

   R7 said to only use Python nowadays. He mentioned having used C before, but now he thinks that Python is best for the undergraduate courses in which he teaches.

4. **Structure of the CS1 courses you teach:**

   (a) **Is it split into theory and practice classes?**
   R7 said that the structure depends on the undergraduate course. Some are divided into theory and practice, other are mixed. R7 stated that even if he is assigned to a class that divides theory and practice, he ministers it in the laboratory, thus making it mixed. The reason for that is because R7 said that he does not see a division between theory and practice in the teaching of CS1 courses.

   (b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
   R7 commented that there are weekly assignments, but they count little toward the final grade. He said that he elaborates the assignments, and the students submit their solutions via Moodle. He checks manually a portion of them, but not all because there are too many. R7 stated that he provides most of the feedback in class by evaluating students' solutions that are implemented in the laboratory.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R7 said to never have used these systems.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   Eryck [Silva] skipped this question since R7 stated to have never used autograders.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R7 said that he still did not do a thorough research about autograders, but he has the impression that they help the students to evaluate their code better than in paper.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   Eryck [Silva] skipped this question since R7 stated to have never used autograders.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R7 said that he comments about these questions regarding code quality during the provided feedback in his classes. He said to alert students about the possibility of adding more significant names to variables or refactoring code in control structures. In this case, he sees some students changing their code while other do not, since the code is already working. R7 stated that in

exams he might penalize solutions regarding code quality, but it would be a minimal penalty. On a side note, R7 also commented that he often wonders how the level of strictness should be while teaching CS1. He thinks that he should not be too strict in the beginning to avoid demotivation and even evasion. However, by doing this, he also thinks students can develop bad programming behaviors that will be difficult to correct in the future.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R7 said that this scenario is possible, and students should use the required structure defined by the assignment. He also commented that a similar situation to this scenario is when students find out about a Python function that already does what an assignment asks. In this case, the students wonder why they should code what an already existing function does.

8. **Do you believe that those non desirable behaviors in CS1 students' code, such as the ones studied in this research, tend to disappear by themselves in the future or the students need proper feedback regarding them from the instructor in order to mitigate those behaviors?**

   R7 stated that if the student wants to become a better programmer, the student will start to policy his own code and avoid undesirable behaviors such as the MC^{3}. However, R7 said if the students receive systematic feedback from the beginning, this process of becoming a better programmer happens earlier.

9. **Do you know any other programming behavior, similar to a Programming Issue [MC^{3}], which was not present in the questionnaire, and you would like to mention?**

   R7 did not mention any other programming behavior. However, he commented about a scenario about students using a rationale that deviates too much from what he expects them to code in their solution. R7 cited an example in which a student used the indexes of a list as variables used in the code, instead of declaring these variables independently.

10. **Consider an autograder system that is also able to detect the Programming Issues [MC^{3}] studied in this research:**

    (a) **Would you be interested in using it in your CS1 classes? How?**
    R7 said that he would be interested in using and that was one of the reasons he decided to volunteer in this research. He thinks that an autograder like this would help both the students and the instructor, even though they would need some time to get used to it. However, R7 stated that the tool would complement the instructor's work, not substitute it.

    (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
    R7 commented that it would already help if the system showed the feedback only to the instructor, since he would be able to find the best way to help the students. To

show the feedback to the students, it would be necessary to know the right moment to avoid overwhelming the students. In this case, R7 said students first need to get used to compiler error messages, since they usually have trouble with it.

(c) **Any other suggestions of the type of feedback the system should provide?**
R7 suggested that the system could have a set of statistical data of what messages the system is sending to the students.

11. **Active Learning Methodologies:**

(a) **Do you have familiarity with them?**
Since R7 mentioned to have familiarity with Active Learning but not about Peer Instruction, Eryck [Silva] only explained about the Peer Instruction technique.

(b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^s], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^s] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^s]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

i. **Would you be interested in using this technique in your CS1 classes?**
R7 said that he would be interested in using this technique especially because it reminded him of gamification, and this factor might motivate the students. He also said that he would be willing to try it without guarantees that it helps the teaching and learning at the end. R7 also commented that the workload for preparing these questions would only happen in the first time, since they would only need minor adjustments in subsequent uses.

ii. **Do you have any other suggestions for the use of this technique?**
R7 did not make any other suggestions.

12. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^s] in CS1 courses?**

R7 did not mention any other methodology. However, he commented that formative feedback is essential for students nowadays. He believes that students often lack confidence in what they are doing, thus making feedback a powerful tool to motivate them and prevent evasion. However, by the end of the course, he also thinks feedback might not be that much helpful and may even make the students too dependent on it.

4.1.8 R8

Date: 1 June 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

   R8 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

   R8 said that he uses Python in a CS 0.5 course, which is one not necessarily targeted to majors in Computer Science. The CS1 course uses Java.

4. **Structure of the CS1 courses you teach:**

   (a) **Is it split into theory and practice classes?**
   R8 said that the course is not divided into theory and practice since it is taught in the laboratory. During the lecture, students try to reproduce the presented examples in their computers.

   (b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
   R8 commented that the instructors elaborate and grade the assignments manually. The reason for that is because of the small number of students in the classes. The assignments count toward the final grade which is also composed by exams and a final project.

5. **Utilization of automatic grading systems:**

   (a) **Do you use any kind of these systems in your CS1 course?**
   R8 stated that he does use Gradescope but not its automated grading function. He explained that since there are many simple assignments, the workload for configuring the system would be greater than manually grading the students' solutions. R8 did comment that he thinks this would not be feasible in larger classes. He also said that some of his colleagues use autograders (including the automated assessment function) when they decide to elaborate fewer but more complex assignments.

   (b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
   Eryck [Silva] skipped this question since R8 stated that he does not use the automated grading function.

   (c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
   R8 stated that these systems are helpful when there are many students in the class, thus making the manual assessment not feasible. He also said that these systems provide a pedagogical approach when they detail which test cases the students' code pass or not. R8 commented that this approach even motivates students to create their own test cases.

   (d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
   R8 had already answered that he does manually check all the students' solutions. He explained that since the assignments are simple enough, it is possible to assess them by just reading the code, without the need of a computer.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R8 stated that he thinks efficiency, in terms of running time, should not be something requested in CS 0.5 nor CS1 classes. He commented that students from these courses can only focus on a specific set of concepts at a time. However, he does emphasize about code simplicity: if students are elaborating too complex code, he suggests them to take a step back and try to think of a simpler solution. R8 said that he only penalizes code complexity if they are the cause for the wrong expected results of the code.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R8 stated that he does not ask for a specific structure to solve an assignment because one of the main messages of the course is that there is not a unique way to solve a problem. He also said that students tend to use the most recent structure they learn to solve an assignment. However, R8 also commented that some students might choose to not use a structure because they are not comfortable with it and decide to not learn it. Regarding the iteration versus recursion situation, R8 agreed that it can be specifically requested, since recursion is a learning objective that students must learn.

8. **Do you believe that those non desirable behaviors in CS1 students' code, such as the ones studied in this research, tend to disappear by themselves in the future or the students need proper feedback regarding them from the instructor in order to mitigate those behaviors?**

   R8 commented that he sees two scenarios regarding MC3. In the first, students are striving to get their code working in an incremental way. Once they accomplish this, they do not worry in cleaning their mid steps that might be left in the code. R8 believes that if the students want to become programmers, they will learn to do this cleaning. The other scenario contemplates situations that are misconceptions, such as an unnecessary else. Just by receiving feedback from a simple autograder, the student might think that the else is mandatory. In this case, only the instructor would be able to provide the correct feedback to the student in saying that it is not a mandatory structure.

9. **Do you know any other programming behavior, similar to a Programming Issue [MC^s], which was not present in the questionnaire, and you would like to mention?**

   R8 commented about two behaviors he usually sees. In the first, students think that the name they assign for a variable has a functional meaning. The other behavior is when students try to combine two different structures, such as *for* loop and if. Since R8 showed a minimal example of these two scenarios, we decided to report them here as Algorithm 47 and 48, respectively. In Algorithm 47, the student thinks that the ***even*** variable will already detect if a number is

even. In Algorithm 48, the student is aggregating a *for* loop and an *if* conditional together to detect even numbers in a list.

Algorithm 47: Example 1 given by R8.

```
1 List = [0, 1, 2, 3, 4, 5]
2 for even in List:
3     print even
```

Algorithm 48: Example 2 given by R8.

```
List = [0, 1, 2, 3, 4, 5]          1
for x in List % 2 == 0:            2
    (...)                          3
```

10. **Consider an autograder system that is also able to detect the Programming Issues [MC^s] studied in this research:**

    (a) **Would you be interested in using it in your CS1 classes? How?**
    R8 said that he would be interested in using it albeit not too much as an instructor that works with many students per class.

    (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
    R8 stated that the system could be configured to limit the amount of feedback to the students. An idea of filter could be messages regarding the MC^3 he most expects students will have in their code.

    (c) **Any other suggestions of the type of feedback the system should provide?**
    R8 suggested that the system could also assess how well the students' code are documented. The evaluation could be by analyzing the need of comments and their location in the code. He said that comment quality, on the other hand, would be difficult to assess automatically.

11. **Active Learning Methodologies:**

    (a) **Do you have familiarity with them?**
    Since R8 mentioned to have familiarity with Active Learning and the Peer Instruction technique, Eryck [Silva] did not explain about them.

    (b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^s], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^s] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^s]. Depending on the percentage of correct answers, the instructor can move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.**

        i. **Would you be interested in using this technique in your CS1 classes?**
        R8 stated that he would be interested in using this technique because he already does something similar. While elaborating a code solution in class, he asks questions (sometimes tricky) to the students. However, R8 said that he does not stimulate all

students since he waits for a volunteer to answer. In addition to that, R8 commented that this technique would help students to learn about MC^3 since they tend to ignore feedback they receive for their code. That way, using techniques like this in class would reinforce the learning.

ii. **Do you have any other suggestions for the use of this technique?**
R8 suggested that this technique should be used after the students get comfortable with the structures, specially about their syntax. He also stated to worry about the workload and the size of the classes to apply this technique, since he thinks it might not be so simple to use them in large classes.

12. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^3] in CS1 courses?**

R8 did not mention any other methodology. However, he said that the automatic detection of MC^3 is important to analyze which ones are most frequent. By doing this, the instructor would be able to create better interventions in his classes.

4.1.9 R9

Date: 14 June 2022.

1. **Introduction**

Here, Eryck [Silva] explains his position as a Ph.D. student and how this research connects with his thesis.

2. **May I record the interview?**

R9 authorized the recording.

3. **You have mentioned that you have familiarity with other programming languages besides Python. But, in terms of teaching CS1, which one do you use in the course?**

R9 stated that he is only using Python in the CS1 courses he teaches.

4. **Structure of the CS1 courses you teach:**

(a) **Is it split into theory and practice classes?**
R9 said that the classes happen in a 5 hour per week format: 3 hours with theory in classroom, and 2 hours of practice in the laboratory. He said that he tries to organize each week to teach about one specific concept, but that is not always possible.

(b) **Are there any kind of summative assignments? Who elaborates them? Who grades them?**
R9 commented that there are assignments in the laboratory, mini projects that students do in groups, and specific sets of exercises that students do individually. All of them count towards the final grade. R9 also said that there are partial exams, both in paper and in the computer, and a final exam. He elaborates and manually grades all these assignments.

5. **Utilization of automatic grading systems:**

(a) **Do you use any kind of these systems in your CS1 course?**
R9 said that he had never used these systems. He explained that even if he did use, he would still read the students' solutions because he thinks it is the best way to assess their knowledge.

(b) **Is the system used only to evaluate the output of the students' code by checking test cases?**
Eryck [Silva] skipped this question since R9 stated that he does not use autograders.

(c) **What is your opinion about autograders in general? Any advantages and disadvantages in specific?**
R9 commented that autograders could be useful in classes with many students because the automated grading would save some time for the instructor. However, he also said that these systems could create an environment in which the students do not receive quality instruction. R9 explained that if the instructor does not read the code, he would not see any good or bad practices that could be present in the students' solutions.

(d) **Do you usually check the students' submissions after they are graded by the autograder? Even the ones that received the maximum grade?**
R9 had already answered that he does not use autograders, and manually check all the students' solutions.

6. **What is your opinion on questions related to legibility, good practices, and code efficiency in CS1 courses? Do you consider them part of the learning objectives, penalizing the assignment grades in case they occur? Or you do not penalize, but comment on the issues with the students? Or do you not see them as a problem at all?**

   R9 commented that he teaches about concepts of code quality to his students. He specifically mentioned about the importance of good, significant variable and function names in which he always tries to use storytelling techniques to emphasize this importance. R9 stated that since he is thorough about these questions in his classes, he does penalize students' solutions in which code quality is not present.

7. **Consider the following scenario: the instructor asks the student to elaborate a solution to a determined problem with a determined programming structure, but the student uses another structure that generates the same desired output. Some examples can be in the use of for versus while; the use of Boolean expressions versus nested if statements; and the use of iteration versus recursion. What is your opinion about this scenario? Might the instructor be interested in this scenario in a learning context?**

   R9 said that he sees the importance of these scenarios and he deals with them by trying to direct his students to use the learning objectives for each assignment. In his classes, he elaborates assignments with a description that indirectly guides students to use a specified structure or code with good practices. He gave the example of requesting functions that have all variables passed as an argument and are returned at the end.

8. **Do you believe that those non desirable behaviors in CS1 students' code, such as the ones studied in this research, tend to disappear by themselves in the future or the students need proper feedback regarding them from the instructor in order to mitigate those behaviors?**

R9 commented that behaviors like the MC^3 need proper feedback because they provide situations in which the student would not care because their code is already correct. Another situation is that the students think that it is enough if they understand their own code, but in reality, it is not enough since programming is a cooperative work and people need to be able to understand code created by others.

9. **Do you know any other programming behavior, similar to a Programming Issue [MC^3], which was not present in the questionnaire, and you would like to mention?**

   R9 did not mention any other programming behavior. On a side note, he commented that he felt as if list of MC^3 was speaking directly to him, since he usually sees many of the mapped behaviors in his classes.

10. **Consider an autograder system that is also able to detect the Programming Issues [MC^3] studied in this research:**

    (a) **Would you be interested in using it in your CS1 classes? How?**
    R9 said that he would be interested in using this system. He explained that he will teach more CS1 classes and a system that could detect the lines or regions of code that has MC^3 would be helpful.

    (b) **About the generated feedback, do you believe that it should be readily shown to the student? Or would it be better to send them to the instructors beforehand so they can better make use of it?**
    R9 stated that this system would help more the students since they would be able to see the test cases they passed and find ways to improve their already correct code. R9 said that the instructor could be able to configure the amount of feedback that students receive to prevent overwhelming.

    (c) **Any other suggestions of the type of feedback the system should provide?**
    R9 suggested that there should be a way of assessing if the students are using the feedback they receive from the system. He said that the instructor can ask the students why they received the feedback messages or what these messages meant.

11. **Active Learning methodologies:**

    (a) **Do you have familiarity with them?**
    Since R9 mentioned to have familiarity with Active Learning and the Peer Instruction technique, Eryck [Silva] did not explain about them. R9 commented that he has experience with Active Learning in high school classes, but in higher education he does not see it working too well because of the amount of time for classes.

    (b) **Consider the following instantiation of Peer Instruction in a CS1 course: to teach about the Programming Issues [MC^3], the instructor first presents a concept about the course and then comments on possible Programming Issues [MC^3] related to that concept. For example, the instructor talks about what variables are and how we make attributions, after that, the instructor alerts about unused variables or a variable declared as itself. After that, a multiple-choice questionnaire is presented to assess the understanding of the concepts: the question aims at identifying code snippets that have Programming Issues [MC^3]. Depending on the percentage of correct answers, the instructor can**

move on to the next topic or stimulate an in-class discussion so the students can convince their peers about the right answer.

i. **Would you be interested in using this technique in your CS1 classes?**
R9 mentioned that he does not have good experiences with Peer Instruction and clickers because close classmates can whisper the answer to other students, or they can just guess the answer. He also commented that finding time to apply these techniques in higher education is challenging.

ii. **Do you have any other suggestions for the use of this technique?**
R9 mentioned that in the high school classes he visited students' workstations to assess their progress by looking directly at their work. He suggested that this technique could be done in a similar way by assigning small code exercises at the end of each class. By evaluating students' solutions to these exercises, the instructor can identify gaps in students' progress.

12. **Is there any other methodology that you believe could be useful to teach about Programming Issues [MC^{8}] in CS1 courses?**

R9 said that he likes storytelling. He believes that students have a better chance of understanding the course material if the instructor uses stories to emphasize details or importance of CS1 topics. He cited the film Inception (2010) and how it can relate to recursion as being "dreams within dreams". R9 suggested that finding a storytelling model to teach about MC^{3} could be helpful.

4.2 Overview

In total, 8 respondents stated that they use Python in their CS1 classes. Furthermore, 2 instructors mentioned that this was a recent change in their institutions, as C and C++ were substituted. One exception was R8 which said that Python is used in a CS 0.5 course, while Java is used in the CS1 course. The format of the classes varies between the institutions, as 2 CS1 courses mix theory and practice, while 5 split them, and 2 are dependent on the undergraduate course. Among those that split the course, there are 2 in which the practice class is dedicated to assist students who are struggling with the concepts. Lastly, 8 respondents stated that they elaborate assignments that count toward the final grade, and 1 said it depends on the instructor. These assignments vary from exercise lists, simple coding projects, and more complex final projects. In terms of assessment, 3 instructors grade these tasks manually while 6 use autograders for that purpose.

Respondents agreed unanimously that the main advantage of autograders is that these systems assist the instructor in classes with many students. This is a key factor as two instructors said that they do not use these systems because their classed do not have many students. These 2 instructors also justified that reading students' code is the best way to assess if the concepts are being understood. On the other hand, among the 6 instructors who use autograders, 3 stated to check students' solutions after the automatic grading, 1 only checks if the assignment is complex, and 2 do not check the solutions at all. Among the disadvantages of autograders were the strictness of their assessment (which is binary in terms of test cases) and the lack of static analysis evaluation of the code (e.g., complexity, style). One respondent stated that he tries to use tools on his own to help identify these assessments that autograders do not cover.

In general, all respondents agreed that code quality is important, but their opinions varied in terms of when they should be taught. While 8 of them do comment on these issues in class, 2 do not penalize students' solutions regarding them. The penalization is either prohibited by the course

organization or cannot happen due to the use of an autograder. However, respondents also agreed that students cannot fail the CS1 course because of code quality issues. 2 instructors stated that if the student is interested in becoming a programmer, he will learn by correct issues like the MC³ by himself along the way. On the other hand, another 2 said that, since some of these occurrences are hidden by the automatic grading process, the students need proper feedback because they will not worry since their code is already working. One respondent also said that teaching about code quality and its related concepts depends much on the undergraduate course.

Examples of other behaviors like the MC³ that we did not already map were: use of functions that are composed of almost all the code; code style issues e.g., inconsistency in spacing and line breaks; and use of lists when other data structures would be more appropriate. Some respondents commented about having already seen MC³ like redundant or simplifiable Boolean comparison (B1), redundant or unnecessary loop (C2), specified verification of instances of open test cases (F2), and functions/variables with non significant name (G4). Other respondents cited behaviors that would lead to incorrect code, thus not being part of the MC³.

All 9 respondents agreed that they would use an autograder that could detect coding behaviors such as the MC³. In general, the instructors stated that a system like this would ease the instructor's workload as well as guide the students in further refining their code. However, respondents were cautious regarding the automated feedback. 5 instructors explicitly stated that the system should be configurable by the instructor to not overwhelm students, especially in the first part of the CS1 course. The suggestions respondents commented involved applying machine learning techniques to detect and suggest code refactoring, detecting issues regarding code complexity and style, configuring the strictness level of the code output comparison (e.g., ignoring white spaces between the expected output), maintaining a set of statistical data regarding the provided feedback to the students, and a mechanism to check if the students are using the provided feedback (e.g., questions asking why they received these messages or what they meant).

In general, 8 respondents stated that they would be interested in using the Peer Instruction technique regarding the MC³ in their classes, although 4 of them does not have experience with it. Two respondents also believe that it would complement the automated feedback via the autograder that can detect MC³. However, the instructors also mentioned concern about the necessary time to implement the technique. One suggested that it should be only in one class dedicated to code quality, while another said that the multiple-choice questionnaire could only be used when classes have the time. In general, all interested respondents agreed that the Active Learning technique should not be used right after the students are exposed to a new CS1 concept, as it would be best for them to grasp the new material first. One instructor stated that he would not be interested in using the Peer Instruction because he does not have good experience with it, since students might cheat or just guess the answer for the multiple-choice questionnaire.

4.3 Obtained Insights

Considering the context of the CS1 classes, teaching interventions that cover MC³ may be useful since all respondents elaborate practical assignments and students' solutions to these assignments can have MC³. However, as there are multiple types of assignments, varying from fixation exercises to final projects, the use of automatic grading might not be feasible to all of them (e.g., a developed game). This factor indicates that a system that can detect MC³ should work independently from automatic grading.

The development of automated feedback to the students regarding MC³ should also be carefully elaborated since 5 instructors stated they worry about the amount of information received by novice programmers. The system should at least provide feedback to the instructors since they would be

able to understand it and find the best way to inform their students. However, as in larger classes this might not be feasible, a configurable option to send feedback directly from the system to the student could be useful. Further investigation is required to determine how the automated feedback should be elaborated.

On the other hand, there were respondents who are interested in Active Learning techniques but are not familiar with the involved methodologies. This factor indicates that the implementation of Peer Instruction in classes would need to prepare these instructors beforehand. The use of this technique in classes would help not only to complement an autograder or system that checks MC^3, but to identify other scenarios such as why students think a code with MC^3 can be constructed with these behaviors, or even new MC^3 not detailed in this research.

5 Rating the Severity of the MC^3

To rank each MC^3 severity, we decided to use the frequency of each category in the Likert-item answers (detailed in Section 3). We also computed the median for each Likert-item answer regarding the MC^3 and show it in Table 4. We created a new ranking based on the answers. First, we summed each similar category (Strongly Agree (SA) and Agree (A), Neutral (N) and Blanks (B), Strongly Disagree (SD) and Disagree (D)) and then we ranked decreasingly by the difference between those who agreed the MC^3 was of severe concern and those who did not. The Diff column in the table represents said difference i.e. the result of $(SA + A) - (N + B + SD + D)$.

Table 4: Ranking of the severity concern of the MC^3. Table sorted decreasingly by the Diff column.

MC^3	**Median**	**SA+A**	**N+B**	**SD+D**	**Diff**
C8	5	31	0	1	30
B6	4	26	4	2	20
C1	4	26	3	3	20
B8	4	24	8	0	16
C2	5	24	5	3	16
C4	4	24	5	3	16
D4	4	24	4	4	16
G4	5	24	7	1	16
H1	4	24	7	1	16
B12	4	23	5	4	14
B9	4	23	4	5	14
E2	4	23	3	6	14
A4	4	22	3	7	12
F2	5	22	8	2	12
G5	4	22	6	4	12
C3	4	21	9	2	10
E1	4	21	7	4	10
G3	4	21	7	4	10
A2	4	20	7	5	8
A6	4	20	6	6	8
A7	4	20	7	5	8

Table 4 (continued from previous page)

MC³	Median	SA+A	N+B	SD+D	Diff
B11	4	20	6	6	8
B10	4	19	9	4	6
B3	4	19	6	7	6
B4	4	19	11	2	6
D1	4.5	19	6	7	6
A8	4	18	8	6	4
B7	4	18	5	9	4
C7	4	17	6	9	2
C6	4	16	9	7	0
F1	3.5	16	9	7	0
H2	3.5	16	8	8	0
G6	3	14	14	4	-4
A1	3	13	9	10	-6
A3	3	12	8	12	-8
B1	3	12	12	8	-8
D2	3	12	8	12	-8
B5	3	11	12	9	-10
G2	3	11	13	8	-10
C5	3	10	11	11	-12
D3	3	10	12	10	-12
H3	2.5	8	8	16	-16
B2	2.5	7	9	16	-18
G1	2	7	8	17	-18
A5	2	5	8	19	-22

The Diff column in Table 4 shows how the disparity between the respondents' answers regarding the severity of the MC³ decreases. This difference reinforces how the C8 MC³ (*for* loop having its iteration variable overwritten (Table 2)) was classified as the most severe in the Likert-item questionnaire. The difference between C8 and B6, next one in ranking, is 10 points. This is much greater than any other gaps between consecutive MC³ in the table, which are composed of 4 points maximum (e.g., between C1 and B8). This indicates that although Diff has a fast decrease in the beginning, the difference becomes smaller over time.

There are 29 MC³ with Diff greater than 0, varying from 30 to 2 points. As it can be seen in Table 4, this group is composed of at least one MC³ from each category (from A to H, detailed in Table 2). Category E (Reasoning) is the only one that is wholly present in this group. On the other hand, there are 16 MC³ with Diff less or equal than 0, varying from 0 to -22 points. Aside from category E, all the other categories have at least one MC³ in this other group.

We highlighted the top 15 in the ranking as the MC³ with the most severity concern. The threshold at 15 was chosen because it was the last number in which the Diff was greater than 10. The highlighting is present in Table 4 by a horizontal line after the 15th ranked MC³ (G5). The same table also highlights the MC³ in which Diff is greater than 0, denoted by a double horizontal line. Table 5 represents the amount of MC³ from every category present in the listed groupings: the top 15, those with Diff greater than 0, and those with Diff less or equal than 0. It is important to remember that the top 15 is a subset of those with Diff greater than 0.

Table 5: Number of MC³ from each category present in the severity ranking (Table 4)

Category	Initial Listing	Severity Ranking		
		Top 15	Diff > 0	Diff ≤ 0
A	8	1	5	3
B	12	4	9	3
C	8	4	6	2
D	4	1	2	2
E	2	1	2	0
F	2	1	1	1
G	6	2	3	3
H	3	1	1	2
Total	45	15	29	16

Table 5 shows that there is at least one MC³ from each category in the top 15. About 53% of them are from categories B (Boolean expressions) and C (Iteration). This indicates that the majority of MC³ are not prone to occur in the first few classes of a typical CS1 course, albeit they could happen within the end of the first month. However, there are behaviors that can happen in the beginning, like A4 (*Redefinition of built-in*) and G4 (*Functions/variables with non significant name*). Moreover, the presence of MC³ from the G category in this group corroborates with CS1 instructors' opinions stated in the interviews (Section 4): variables and general code organization are early taught in CS1 and, according to the respondents, name significance and clear organization should be also taught as soon as possible. From this group of most severe MC³, D4 (*Function accessing variables from outer scope*) and E2 (*Redundant or unnecessary use of lists*) might be the ones that might not happen until later in the CS1 course, depending on its syllabus. D4 is also a MC³ related to Python since it does not occur in other programming languages such as C or Java. Lastly, F2 (*Specific verification for instances of open test cases*) is a MC³ that is dependent on the use of autograders by the CS1 course, independent of the programming language. Table 6 lists the name of the top 15 MC³ sorted by their given ID in the initial listing (Table 2).

Our decision to highlight the 15 most severe MC³ was to provide a feasible group that is prominent to further investigation. In the research presented in this report, we did not check the frequencies of these MC³ in the analyzed assignments (Table 1). However, we expect that this severity ranking will be a starting point to analyze not only the frequency, but also why students create their correct code with these misconceptions.

6 Conclusions

In this report, we described the process of identifying and assessing Misconceptions in Correct Code (MC³) of Python CS1 courses. MC³ are undesirable programming behaviors, with respect to the learning objectives, which are present in correct student code. We manually analyzed 2441 submissions from a Python CS1 course. All these submissions received maximum grade from an automatic grading system. In total, we identified 45 MC³ and divided them into 8 categories: Variables, identifiers, and scope; Boolean expressions; Iteration; Function parameter use and scope; Reasoning; Test cases; Code Organization; and Other.

To assess and identify a subset of MC³ that should be addressed with teaching interventions,

Table 6: List of the top 15 most severe MC³. Table sorted by ID.

Category	ID	Misconception Name
A: Variables, identifiers, and scope	A4	Redefinition of *built-in*
B: Boolean expressions	B6	Boolean comparison attempted with *while* loop
	B8	Non utilization of *elif/else* statement
	B9	*elif/else* retesting already checked conditions
	B12	Consecutive equal *if* statements with distinct operations in their blocks
C: Iteration	C1	*while* condition tested again inside its block
	C2	Redundant or unnecessary loop
	C4	Arbitrary number of *for* loop execution instead of *while*
	C8	*for* loop having its iteration variable overwritten
D: Function parameter use and scope	D4	Function accessing variables from outer scope
E: Reasoning	E2	Redundant or unnecessary use of lists
F: Test cases	F2	Specific verification for instances of open test cases
G: Code organization	G4	Functions/variables with non significant name
	G5	Arbitrary organization of declarations
H: Other	H1	Statement with no effect

we developed an online survey with CS1 instructors. This survey was composed of an online questionnaire and a semi structured interview. The questionnaire had the intention of classifying all 45 MC³ regarding its severity concern in a Likert-item format. The objective of the interview was to understand different contexts of CS1 teaching and to evaluate ideas of teaching interventions regarding the MC³. A total of 32 volunteers answered the online questionnaire, and 9 of them were interviewed. We ranked the severity of the MC³ according to the difference between respondents who agreed the misconception was severe and those who did not agree. A total of 15 MC³ were highlighted as the most severe MC³, prone to be further analyzed.

Further investigation of the most severe MC³ compose future work of this research. Evaluation of the frequency and understanding why students elaborate their code with the MC³ are the next steps we intend to take. The main goal is to elaborate teaching interventions that addresses that mitigate the occurrence of the MC³ and a system that can automatically detect these behaviors.

Acknowledgments

This research is supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under the process 142476/2020-0. Additional support was provided by the State

University of Campinas (UNICAMP).

We would like to thank all 32 respondents to the online questionnaire and the 9 participants in the semi structured interviews that were part of this research.

References

[1] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. Concept inventories in computer science for the topic discrete mathematics. *SIGCSE Bull.*, 38(4):132–145, jun 2006.

[2] C.C. Bonwell and J.A. Eison. *Active Learning: Creating Excitement in the Classroom.* J-B ASHE Higher Education Report Series (AEHE). Wiley, 1991.

[3] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 364–369, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Guilherme Gama, Ricardo Caceffo, Renan Souza, Raysa Bennati, Tales Aparecida, Islene Garcia, and Rodolfo Azevedo. An antipattern documentation about misconceptions related to an introductory programming course in python. *Institute of Computing, University of Campinas, Tech. Rep. IC-18-19*, 2018.

[5] Jack Hollingsworth. Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529, October 1960.

[6] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research methods in human-computer interaction.* Morgan Kaufmann, 2017.

[7] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas Price. The impact of adding textual explanations to next-step hints in a novice programming environment. In *Proceedings of the 2019 ACM conference on innovation and technology in computer science education*, pages 520–526, 2019.

[8] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. Metacognitive difficulties faced by novice programmers in automated assessment tools. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 41–50, 2018.

[9] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24, 2017.

[10] R.H. Sampieri, C.F. Collado, and P.B. Lucio. *Metodologia de pesquisa.* Penso, 2013.

[11] Juha Sorva. Notional machines and introductory programming education. *ACM Trans. Comput. Educ.*, 13(2), jul 2013.

A Questionnaire

This appendix presents the original questionnaire, created in Google Forms and sent to CS1 instructors.

Analysis of Programming Issues in Python Introductory Programming Courses

INFORMED CONSENT FORM

Research title: Analysis of Programming Issues in Python Introductory Programming Courses
Research team: Eryck Pedro da Silva (Head Researcher)
Rodolfo Jardim de Azevedo (Supervisor)
Ricardo Edgard Caceffo (Co-supervisor)
CAAE's number: 51444121.5.0000.5404

You are being invited to participate in a research project. This section, called Informed Consent Form, aims to establish your rights as a volunteer to the research. Please read this section carefully, so you can clear any questions you may have. If you have any questions before or even after accepting, you can ask them to the researcher. If you wish, please take this Form to your house, and talk to your family or other people before deciding to participate. There will be no type of penalization or loss if you decide not to participate or remove your authorization at any moment. We strongly suggest that you make a copy of this form (right click and choose "Save as") and keep it with you. We also ask that you provide an email address so that we can send you a copy of the answers you fill in this questionnaire.

Justification and objectives:

The objective of this research is to classify the severity concern of Programming Issues, which are occurrences that are present in student's code that are considered correct by an automatic grading system, but indicate non desirable factors that, if not corrected, can lead to future undesirable programming practices.

Procedure:

By participating in this research, you are being invited to take part in two main activities: responding to a questionnaire and the recording of an interview. Please note that although we primarily designed both activities in English, we will be accepting answers in English and in Brazilian Portuguese. The application method is composed of the following steps:

Invitation sent by email to professors of undergraduate introductory programming courses that have experience with the Python programming language, informing the justification, objectives and the link to the questionnaire. The invitations are done individually, with each email having only one sender and recipient, or via discussion lists, in which all recipients receive as hidden copies, so that no participant has access to the other email addresses.

Access to the questionnaire in which, firstly, will be presented the informed consent form. Since this entire research is done online, the consent shall be informed online, that is, by clicking the button "I agree with the terms in the Informed Consent Form", the volunteer expresses his sole desire to participate. The following questions will only be

accessed if the participant agrees with this form.

In the next session the volunteer will answer a few identification questions: name; undergraduate institution that they work; total time that the participant teaches introductory programming classes; if they are familiar with the Python programming language; and which other languages they are familiar. It's important to clarify that this data will be anonymized throughout the research but is needed to map the population that participated. After that, the questionnaire will follow to the next session.

In the next session the Programming Issues will be presented. They are composed of 45 questions that aim to classify their concern factor in a Likert scale, with the possibility of writing any additional comments that the volunteer deems worthy of informing.

At the end of the questionnaire, the volunteer will be invited to take part in another phase of this research: a semi structured interview that will be conducted at a later date, by virtual meeting. The semi structured interview will be composed by a small talk in order to get to know a bit about the volunteer, followed by a session asking if they have found any of the Programming Issues while teaching introductory programming courses and how they deal with them, and then followed by a session asking if they have any commentary or would like to inform of any new Programming Issue that was not listed in this research. The researcher will ask if the meeting can be recorded for future reference, but it's not mandatory.

If the volunteer agrees to take part in this next phase, they must mark the option "Yes, I volunteer to participate in the semi structured interview". The research team will contact them via the institutional email informed in the questionnaire at a later date.

With the conclusion of this research, the results will be sent to the volunteers via the email informed at the beginning of the questionnaire.

The estimated time of completing the questionnaire is from 40 to 55 minutes. For the semi structured interview, the estimated time is 40 minutes. During the research, the questionnaire's results and the recorded interviews will be kept within the computer of the head researcher, with backup copies stored in a cloud storage only accessible to the research team. When the research is concluded, the files will be stored in the State University of Campinas repository for the institution's recommended period.

Discomfort and risks:

In this research, the risk is assessed as minimum, that is, you can have a bad impression with the presented Programming Issues or feel emotionally uncomfortable while responding to the questionnaire and/or participating in the interview. Since the activities will be done online, you might also feel uncomfortable with the technology that is used. Also, by using those technologies, we cannot guarantee total anonymity nor that the data might be shared with partner enterprises associated. We assessed this risk as minimum because both questionnaire and interview will not involve moral or personal topics and the volunteers are people that are already familiar with teaching introductory programming courses, as well as the nature of the technology that is used throughout this research.

If you feel uncomfortable while participating in this research, you can interrupt it at any moment by closing the questionnaire's page or by leaving the virtual meeting of the interview. Even after your participation, should you deem necessary, you can contact the research staff via the contact information provided asking for your data to be removed

from this research. Your refusal will not affect your relationship with the researcher or the institution that is applying this research, nor the one that you're affiliated with.

Benefits:

The benefits involve eventually learning about Programming Issues that you may yet not know. The Programming Issues will be presented in a structured manner, including real examples of student code, so you can reflect and assess on them, possibly leading to further conclusions about those issues. The results of this research, that is, a list with all the Programming Issues that are considered more severe, will also be sent to you via the email informed in the questionnaire.

Medical follow-up and assistance:

Since during the elaboration and execution of this research there were no identifiable situations that need interventions (medical, pedagogical, nutritional, etc.), there will not be conducted any type of medical follow-up or assistance during, after or if this research is interrupted.

Privacy:

You are guaranteed that your identity will be kept confidential and no information about it will be shared with anyone outside the research team. When the results are published, your name will not be cited.

Your answers will be confidential and anonymized. The personal information will be used to map the population that participated in this research and will not, by any means, be based to make conclusions towards any specific group or institution that the volunteers are affiliated with.

Refunds and compensation:

There will not be any kind of refund or financial award since you will not have any expense with your participation in this research. You have the right of indemnification in occurrence to any eventual damages resulting from this research. The technologies used in this research are free to use, not creating any kind of expense to you.

Contact:

If you have any questions about this research, you can contact the research staff via the contact information below:

Eryck Pedro da Silva (head researcher)
Phone number: +55 19 3521-5857
E-mail: eryck.silva@ic.unicamp.br
Institute of Computing - UNICAMP
Address: Av. Albert Einstein, 1251 - Cidade Universitária / Campinas. Postal Code: 13083-852

Rodolfo Jardim de Azevedo (supervisor)
Phone number: +55 19 3521-5857
E-mail: rodolfo@ic.unicamp.br
Institute of Computing - UNICAMP
Address: Av. Albert Einstein, 1251 - Cidade Universitária / Campinas. Postal Code: 13083-852

Ricardo Edgard Caceffo (co-supervisor)
Phone number: +55 19 3521-5857
E-mail: rec@ic.unicamp.br

Institute of Computing - UNICAMP
Address: Av. Albert Einstein, 1251 - Cidade Universitária / Campinas. Postal Code: 13083-852

In case of complaints about your participation and about ethic issues of this research, you can contact with the Ethics Research Committee's secretariat of UNICAMP from 08:00hs to 11:30hs and from 13:00hs to 17:30hs (BRT) at street Tessália Vieira de Camargo, 126; Postal Code 13083-887 Campinas - São Paulo; phone numbers: +55 19 3521-8936 or +55 19 3521-7187; e-mail: cep@unicamp.br. If the communication must be accessible in the Brazilian Sign Language (LIBRAS) you can contact the TILS Central of UNICAMP in https://www.prg.unicamp.br/tils/.

The Ethics Research Committee (CEP in portuguese).

The purpose of the CEP is to evaluate the ethics aspects of every research that involves human subjects. The National Commission of Ethics Research (CONEP in portuguese) establishes the regulation of protection of human subjects involved in research, coordinates the various network of institutional Ethics Research Committees, and is the main consulting body in the field of ethics in research.

Informed consent:

The acceptance of this informed consent will be done virtually, you can download a copy of this file and keep it with you. As such, by clicking the button "I agree with the terms in the Informed Consent Form", you agree that:

"After reading and understanding the clarifications about the nature of this research, its objectives, methods, benefits, eventual risks and discomforts that it may cause, I agree to participate in the research: Analysis of Programming Issues in Python Introductory Programming Courses."

Researcher's responsibilities:

I have ensured to have accomplished the exigencies of CNS/MS 466/2012 resolution and its complements in the elaboration of protocol and the construction of this Informed Consent Form. I have also ensured to have given the volunteer one copy of this document. I inform that this research was approved by the Ethics Research Committee in which the project was subjected and by the National Commission of Ethics Research if it was needed. I declare to utilize the data gained by this research exclusively for the purposes described in this document or the ones agreed by the volunteer of this research.

* Required

1. Do you agree with the terms explicited in the Informed Consent Form? *

Mark only one oval.

- ◯ I agree with the terms in the Informed Consent Form *Skip to question 2*
- ◯ I don't agree with the terms in the Informed Consent Form

Personal information

We would like to know a few questions about yourself. Please remember that, as said in the Informed Consent Form, the answers will be anonymized throughout the research.

2. Name:

_____

3. University you work:

_____

4. Institutional e-mail:

_____

5. How many years of experience do you have teaching introductory programming courses?

Mark only one oval.

- ◯ Less than 1 year
- ◯ 1-2 years
- ◯ 3-5 years
- ◯ 6-10 years
- ◯ More than 10 years

6. Do you have experience teaching the Python programming language?

Mark only one oval.

- ◯ Yes
- ◯ No

7. Please state any other programming languages you are familliar with:

Category A - Variables, identifiers and scope

In this section the Programming Issues that are related to variables, identifiers and their scope are presented. There are 8 Programming Issues in total, cataloged from A1 to A8. For each one we show the name, an example, and a description of the example. There are two questions to each Issue: the first is a Likert scale classification of the severity concern of the Programming Issue and the second is a text field in which you can provide any additional comment you deem necessary to express your opinion about the Issue or your answer. This structure will be similar throughout the other Categories in this questionnaire. Furthermore, please remember we will be accepting answers in English and in Brazilian Portuguese.

A1: Unused variable

Indicates an occurrence in which a variable was declared somewhere in the code, but wasn't used furtherly.

8. A1: I believe this issue is of great concern:

```
1   '''A1: Unused variable'''
2   var1 = 3
3   var2 = 4
4   func1(var1)
5
6   def foo(arg1, arg2):
7       var3 = arg1 + arg2
8       return arg1 + arg2
9
10  func2(foo(5, 5))
```

In this generic example of A1, var2 and var3, declared in lines 3 and 7, respectively, were not used.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

9. A1: Use this space to add further comments about this issue:

A2: Variable assigned to itself

Indicates an occurrence in which a variable receives itself as a value.

10. A2: I believe this issue is of great concern:

```
1   '''A2: Variable assigned to itself'''
2   var = 3
3   var = var
```

In this generic example of A2, the variable "var" is assigned to itself in line 3.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

11. A2: Use this space to add further comments about this issue:

A3: Variable unnecessarily initialized

Indicates an occurrence in which a variable, before being used in the code, is initialized with a determined value, but this assignment is not necessary.

12. A3: I believe this issue is of great concern:

```
1   '''A3: Variable unnecessarily initialized'''
2   var1 = ''
3   var2 = str
4   var1 = func1()
5   var2 = func2()
```

In this generic example of A3, the variables "var1" and "var2" were unnecessarily initialized in lines 2 and 3, respectively, before being assigned other values.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

13. A3: Use this space to add further comments about this issue:

A4: Redefinition of built-in

Indicates an occurrence in which a preestablished Python command has its functionality overwritten with another purpose, generally being used as a variable name.

14. A4: I believe this issue is of great concern:

```
'''A4: Redefinition of built-in'''
max = 23
list = [max]
bool = True
func(max, list, bool)
```

In this generic example of A4, variables with names "max", "list" and "bool" were declared and used in the code, but they are already built-in with specific definitions already present in Python.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

15. A4: Use this space to add further comments about this issue:

A5: Unused import

Indicates an occurrence in which an import from a library is declared in the code, but none of its functionalities is used.

16. A5: I believe this issue is of great concern:

```
'''A5: Unused import'''
import math
base = 3
exp = 4
func(base**exp)
```

In this generic example of A5, the math module was imported but none of its functions or methods were not used in the code.

Mark only one oval.

	Strongly disagree					Strongly agree
1	◯					
2		◯				
3			◯			
4				◯		
5					◯	

17. A5: Use this space to add further comments about this issue:

A6: Variables with arbitrary values (Magic Numbers) used in operations

Indicates an occurrence in which variables have arbitrary values used in the code. These values, also known as "magic numbers" are generally dependent on the problem instance.

18. A6: I believe this issue is of great concern:

```
'''A6: Variables with arbitrary values (Magic Numbers) used in operations'''
var1 = 0
var2 = 10000000000000000000000000000000000000000000000
var3 = int(input())
if var3 < var2:
    varRes1 = var3
if var3 > var1:
    varRes2 = var1

func(varRes1, varRes2)
```

In this generic example of A6, the variables "var1" and "var2" were declared and assigned arbitrary values to be in further comparisons in the code.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

19. A6: Use this space to add further comments about this issue:

A7: Arbitrary manipulations to modify declared variables

Indicates an occurrence in which variables have their values altered by arbitrary means, such as erasing its value by assigning an empty string to it.

20. A7: I believe this issue is of great concern:

```
1  '''A7: Arbitrary manipulations to modify declared variables'''
2  var = int(input())
3  while var != -1:
4      func(var)
5      var = ''
6      var = int(input())
```

In this generic example of A7, the variable "var" had its previous value erased by assigning an empty string to it in line 5, before being assigned another value in line 6.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

21. A7: Use this space to add further comments about this issue:

A8: Arbitrary treatment of the stopping point of reading values

Indicates an occurrence in which the stopping condition for reading data is executed by arbitrary means, such as including the indicator in the set of valid data to be treated furtherly.

22. A8: I believe this issue is of great concern:

```
1  '''A8: Arbitrary treatment of the stopping point of reading values'''
2  while True:
3      var1 = input()
4      lst1.append(var1)
5      if var1 == -1:
6          break
7
8  for iter in lst1:
9      if iter = -1:
10         lst1.remove(iter)
```

In this generic example of A8, a while loop for reading input data was declared and the stopping point was when the value of "-1" was read. In the code, this stopping value is read, assigned to "lst1", which is a list of valid inputs and then excluded from "lst1" in a further for loop in line 10.

Mark only one oval.

	Strongly disagree				Strongly agree
	1	2	3	4	5

23. A8: Use this space to add further comments about this issue:

Category B - Boolean expressions	In this section the Programming Issues that are related to boolean expressions and conditional statements are presented. There are 12 Programming Issues in total, cataloged from B1 to B12.

B1: Redundant or simplifiable boolean comparison

Indicates an occurrence in which a boolean comparison can be expressed in a simpler way.

24. B1: I believe this issue is of great concern:

```
1  '''B1: Redundant or simplifiable boolean comparison'''
2  if (var1 > var2) == True:
3      var3 = True
4  else:
5      var3 = False
```

In this generic example of B1, there was no need to check if the comparison "var1 > var2" was "True" in line 2.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

25. B1: Use this space to add further comments about this issue:

B2: Boolean comparison separated in intermediary variables

Indicates an occurrence in which a boolean comparison, with more than two factors, is coded by assigning the preliminary results in auxiliary variables, instead of calculating the whole expression in the same operation.

26. B2: I believe this issue is of great concern:

```
'''B2: Boolean comparison separated in intermediary variables'''
condAux = cond1 and cond2
condRes = condAux and cond3
```

In this generic example of B2, "condRes" is the result of "cond1 and cond2 and cond3" but instead of being calculated in one expression, "cond1" and "cond2" are calculated first in the "condAux" intermediary variable.

Mark only one oval.

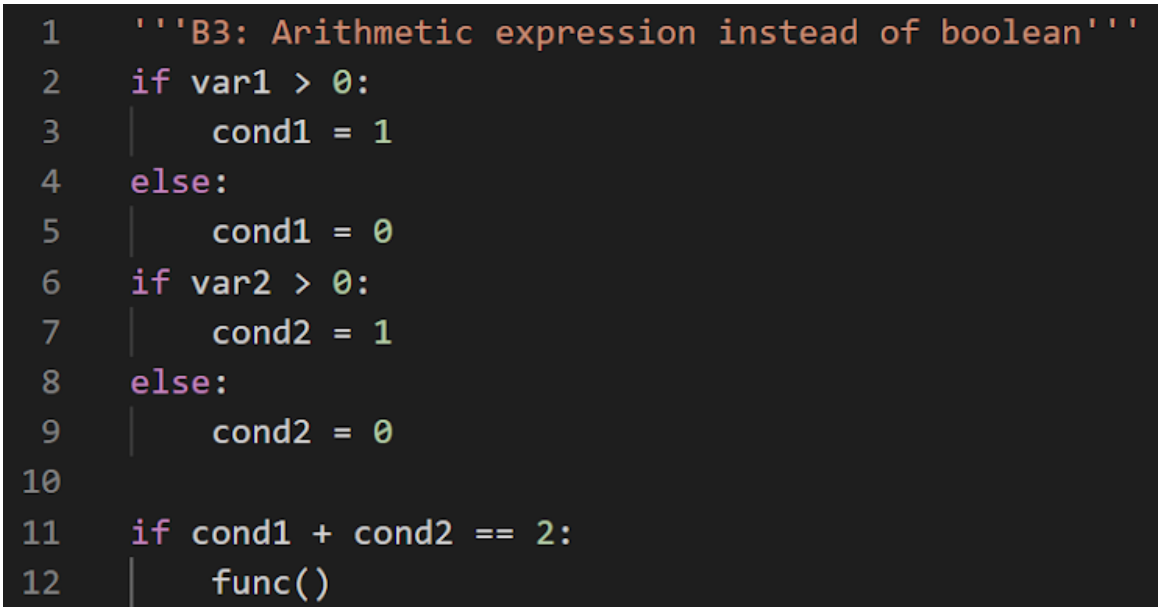
	Strongly disagree
1	
2	
3	
4	
5	
	Strongly agree

27. B2: Use this space to add further comments about this issue:

B3: Arithmetic expression instead of boolean

Indicates an occurrence in which a boolean expression is coded via arithmetic expression, assigning numbers to certain simpler conditions and then checking a more complex case by operating these variables with numbers stored, like the sum of them all.

28. B3: I believe this issue is of great concern:



```
1   '''B3: Arithmetic expression instead of boolean'''
2   if var1 > 0:
3       cond1 = 1
4   else:
5       cond1 = 0
6   if var2 > 0:
7       cond2 = 1
8   else:
9       cond2 = 0
10
11  if cond1 + cond2 == 2:
12      func()
```

In this generic example of B3, the condition that needs to be checked in line 11 is made by an arithmetic expression check using integer numbers that were assigned to "cond1" and "cond2" in the previous if statements, instead of using boolean operators.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

29. B3: Use this space to add further comments about this issue:

B4: Repeated commands inside if-elif-else blocks

Indicates an occurrence in which the same commands appear in different blocks of if-elif-else.

30. B4: I believe this issue is of great concern:

```
1   '''B4: Repeated commands inside if-elif-else blocks'''
2   if cond1:
3       func1()
4       func2()
5       func3()
6       func4()
7   else:
8       func1()
9       func2()
10      func3()
11      func5()
```

In this generic example of B4, the commands "func1()", "func2()" and "func3()" are repeated in both if and else blocks declared in line 2 and 7, respectively.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

31. B4: Use this space to add further comments about this issue:

B5: Nested if statements instead of boolean comparison

Indicates an occurrence in which boolean comparisons are coded by nesting if statements instead of using logic operators.

32. B5: I believe this issue is of great concern:

```
1  '''B5: Nested if statements instead of boolean comparison'''
2  if var1 > 0:
3      if var2 > 0:
4          func()
```

In this generic example of B5, a nested if statement declared in line 2 was made to check if both "var1" and "var2" were greater than zero, instead of using a boolean comparison.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

33. B5: Use this space to add further comments about this issue:

B6: Boolean comparison attempted with while loop

Indicates an occurrence in which a boolean comparison is coded using a while loop but there's no need for the commands inside its block to be looped, since it's execution is stopped after one iteration.

34. B6: I believe this issue is of great concern:

```
1  '''B6: Boolean comparison attempted with while loop'''
2  while var!=0:
3      func1()
4      func2()
5      func3()
6      break
```

In this generic example of B6, the condition to execute "func1()", "func2()" and "func3()" is checked using a while loop declared in line 2, but since there was no need for repeated iterations of those commands, a break statement was declared directly after them, in line 6.

Mark only one oval.

	Strongly disagree				Strongly agree
	1	2	3	4	5

35. B6: Use this space to add further comments about this issue:

B7: Boolean validation variable instead of elif/else

Indicates an occurrence in which a boolean variable is declared and assigned a condition that is tested further with consecutive if statements when an elif/else could be implemented without the need of this validation variable.

36. B7: I believe this issue is of great concern:

```
1  '''B7: Boolean validation variable instead of elif/else'''
2  flag = False
3  if var1 > 0 and var2 > 0:
4      flag = True
5
6  if flag == True:
7      func1()
8  if cond1 and flag == False:
9      func2()
```

In this generic example of B7, the "flag" variable is assigned in line 2 and furtherly used in lines 6 and 8 by using consecutive if statements instead of elifs from the first declared if in line 3.

Mark only one oval.

	Strongly disagree				Strongly agree
	1	2	3	4	5

37. B7: Use this space to add further comments about this issue:

B8: Non utilisation of elif/else statement

Indicates an occurrence in which, by not using an elif/else statement, the code has all conditionals checked only with if statements and is prone to errors by specific executions that can enter multiple if statements when it's not supposed to.

38. B8: I believe this issue is of great concern:

```
1  '''B8: Non utilisation of elif/else statement'''
2  if var <= 0:
3      func1()
4  if var % 2 == 0:
5      func2()
```

In this generic example of B8, because in lines 2 and 4 were declared only if statements, both "func1()" and "func2()" are prone to be executed depending on the value assigned to "var" and this behavior was not intended.

Mark only one oval.

	Strongly disagree
1	
2	
3	
4	
5	
	Strongly agree

39. B8: Use this space to add further comments about this issue:

B9: elif/else retesting already checked conditions

Indicates an occurrence in which a condition that was already checked in an if statement is checked again in an elif, or inside an else block (with another if statement).

40. B9: I believe this issue is of great concern:

```
1  '''B9: elif/else retesting already checked conditions'''
2  if var1 <= 0:
3      func1()
4  elif var2 > 0 and not var1:
5      func2()
```

In this generic example of B9, the elif statement in line 4 is checking the already verified condition related to the "var1" variable in line 2.

Mark only one oval.

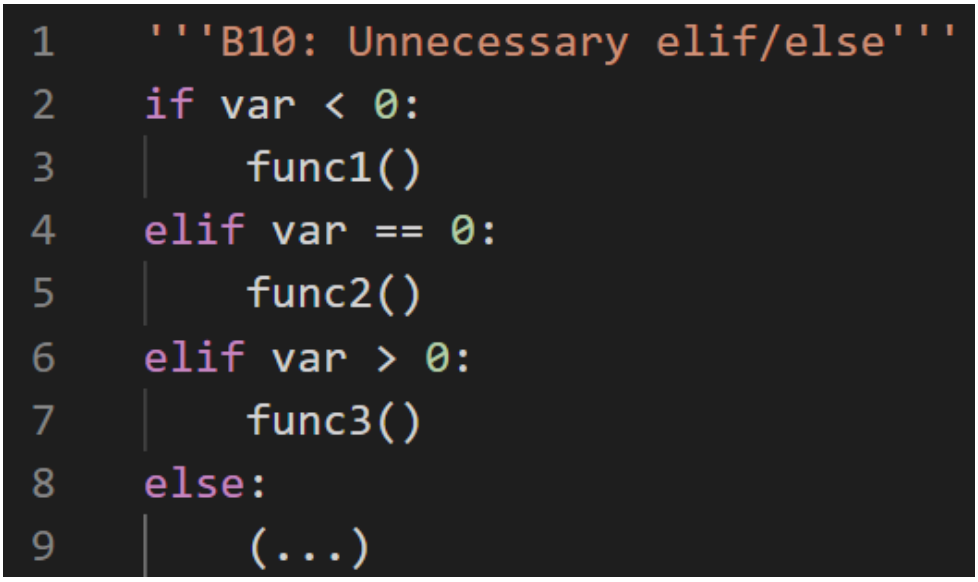
	Strongly disagree				Strongly agree
	1	2	3	4	5

41. B9: Use this space to add further comments about this issue:

B10: Unnecessary elif/else

Indicates an occurrence in which elif/else statements are declared but their blocks contain code that has no effect (e.g. else: pass).

42. B10: I believe this issue is of great concern:



```
1  '''B10: Unnecessary elif/else'''
2  if var < 0:
3      func1()
4  elif var == 0:
5      func2()
6  elif var > 0:
7      func3()
8  else:
9      (...)
```

In this generic example of B10, even though all possible scenarios for the variable "var" were treated in if/elif statements in lines 2, 4 and 6, an else statement was declared to execute some command that will not be reached.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

43. B10: Use this space to add further comments about this issue:

B11: Consecutive distinct if statements with the same operations in their blocks
Indicates an occurrence in which consecutive if statements that check distinct conditions are declared, but there are repeated commands in their blocks.

44. B11: I believe this issue is of great concern:

```
'''B11: Consecutive distinct if statements with the same operations in their blocks'''
if cond1:
    func1()
if cond2:
    func1()
```

In this generic example of B11, two distinct, consecutive if statements checking the "cond1" and "cond2" are declared in order to execute the same "func1()" command.

Mark only one oval.

Strongly disagree

1
2
3
4
5

Strongly agree

45. B11: Use this space to add further comments about this issue:

B12: Consecutive equal if statements with distinct operations in their blocks

Indicates an occurrence in which consecutive if statements that check the same condition are declared, with distinct commands in their blocks.

46. B12: I believe this issue is of great concern:

```
'''B12: Consecutive equal if statements with distinct operations in their blocks'''
if cond1:
    func1()
if cond1:
    func2()
```

In this generic example of B12, two equal, consecutive if statements that check the same "cond1" condition are declared in order to execute distinct "func1()" and "func2()" commands.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

47. B12: Use this space to add further comments about this issue:

Category C - Iteration

In this section the Programming Issues that are related to iteration structures and operations inside them are presented. There are 8 Programming Issues in total, cataloged from C1 to C8.

C1: While condition tested again inside its block

Indicates an occurrence in which a while loop has its condition tested again inside its block in order to verify if it's necessary to stop the looping process.

48. C1: I believe this issue is of great concern:

```
1  '''C1: While condition tested again inside its block'''
2  varCond = (...)
3  while varCond != 0:
4      varCond = (...)
5      if varCond != 0:
6          break
```

In this generic example of C1, the while condition needs to be checked again in line 5 because "varCond" is updated in line 4. If "varCond" was updated in the end, there would be no need for the check in line 5.

Mark only one oval.

	Strongly disagree					Strongly agree
	1	2	3	4	5	

49. C1: Use this space to add further comments about this issue:

C2: Redundant or unnecessary loop

Indicates an occurrence in which although a loop is used, its necessity is not needed since the whole iteration is executed only once.

50. C2: I believe this issue is of great concern:

```
1   '''C2: Redundant or unnecessary loop'''
2   while True:
3       for i in range(1):
4           (...)
5       break
```

In this generic example of C2, an external while loop encapsulates an internal for loop. The external loop executes only once since a break statement is declared in line 5. The internal for loop also executes only once, hence both loops are unnecessary.

Mark only one oval.

	Strongly disagree				Strongly agree
	1	2	3	4	5

51. C2: Use this space to add further comments about this issue:

C3: Redundant operations inside loop

Indicates an occurrence in which operations are being unnecessarily calculated inside a loop, like the average of a list being calculated at each iteration.

52. C3: I believe this issue is of great concern:

```
1   '''C3: Redundant operations inside loop'''
2   for iter in lst:
3       var1 += 1
4       var2 += iter
5       var3 = var2/var1
```

In this generic example of C3, the variable "var3" is attempting to calculate the average of the numbers stored in the list "lst", but the operation is inside the loop, in line 5.

Mark only one oval.

	Strongly disagree					Strongly agree
1	◯					
2		◯				
3			◯			
4				◯		
5					◯	

53. C3: Use this space to add further comments about this issue:

C4: Arbitrary number of for loop execution instead of while

Indicates an occurrence in which a for loop is declared with an arbitrary number of executions, believing that it will be enough to cover all the probable executions, instead of using a while loop, more suited for this case.

54. C4: I believe this issue is of great concern:

```
1  '''C4: Arbitrary number of for loop execution instead of while'''
2  for i in range(65535):
3      if cond1:
4          break
```

In this generic example of C4, the for loop was declared in line 2 to execute "65535" times, an arbitrary number thought to be enough iterations needed before the condition "cond1" happens and executes the break statement out of the loop.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

55. C4: Use this space to add further comments about this issue:

C5: Use of intermediary variables to loop control

Indicates an occurrence in which the variable that controls the loop is updated via other intermediary variables which are fuzzy or obfuscated.

56. C5: I believe this issue is of great concern:

```
'''C5: Use of intermediary variables to loop control'''
while cond:
    var2 = func1(var1)
    cond = func2(var2)
```

In this generic example of C5, a while loop is declared with "cond" as executing condition. Inside its block, "cond" is updated using the result of a function applied to an intermediary variable "var2", which itself is a function applied to another variable, "var1", obfuscating how "cond" is being updated.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

57. C5: Use this space to add further comments about this issue:

C6: Multiple distinct loops that operates over the same iterable

Indicates an occurrence in which multiple consecutive loops that operate over the same iterable are declared with different operations being performed inside their respective blocks.

58. C6: I believe this issue is of great concern:

```
1  '''C6: Multiple distinct loops that operates over the same iterable'''
2  for iter in lst:
3      var1 = func1(iter)
4
5  for iter in lst:
6      var2 = func2(iter)
7
8  for iter in lst:
9      var3 = func3(iter)
```

In this generic example of C6, three distinct for loops that iterate the same way over the same list "lst" were declared in lines 2, 5 and 8, in order to execute different commands.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

59. C6: Use this space to add further comments about this issue:

C7: Arbitrary internal treatment of loop boundaries

Indicates an occurrence in which inside a loop are declared specific conditionals treating the boundary values of the iterable, that is, the first and/or the last element.

60. C7: I believe this issue is of great concern:

```
1  '''C7: Arbitrary internal treatment of loop boundaries'''
2  for iter in range(numMaxIter):
3      if iter == 0:
4          func1()
5
6      if iter == numMaxIter - 1:
7          func2()
```

In this generic example of C7, specific conditions were declared for the boundary values the variable "iter" can be assigned in the execution of the for loop declared in line 2 (when "iter == 0" and "iter == numMaxIter -1").

Mark only one oval.

	Strongly disagree					Strongly agree
	1	2	3	4	5	

61. C7: Use this space to add further comments about this issue:

C8: for loop having its iteration variable overwritten

Indicates an occurrence in which a for loop has its own iteration variable's value overwritten inside its block.

62. C8: I believe this issue is of great concern:

```
1  '''C8: for loop having its iteration variable overwritten'''
2  for iter in range(numMaxIter):
3      iter = (...)
```

In this generic example of C8, the iteration variable "iter" declared in the for loop in line 2 is overwritten in line 3.

Mark only one oval.

Strongly disagree	
1	◯
2	◯
3	◯
4	◯
5	◯
Strongly agree	

63. C8: Use this space to add further comments about this issue:

Category D - Function parameter use and scope

In this section the Programming Issues that are related to the use of user defined functions, its parameters usage and function scope are presented. There are 4 Programming Issues in total, cataloged from D1 to D4.

D1: Inconsistent return declaration

Indicates an occurrence in which only some of the return declarations inside a function return an expression. If one return statement returns an expression, the others where nothing is returned should return None.

64. D1: I believe this issue is of great concern:

```
1  '''D1: Inconsistent return declaration'''
2  def func1(var):
3      if var > 1:
4          return (...)
```

In this generic example of D1, according to PEP8, there would be a necessity of adding a "return None" in the case if the condition in line 3 was not met.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

65. D1: Use this space to add further comments about this issue:

D2: Too many return declarations inside a function

Indicates an occurrence in which a function has a high number of return declarations inside its block.

66. D2: I believe this issue is of great concern:

```
'''D2: Too many return declarations inside a function'''
def func1():
    if cond1:
        return (...)
    elif cond2:
        return (...)
    elif cond3:
        return (...)
    (...)
    elif condN:
        return (...)
    else:
        return (...)
```

In this generic case of D2, a high number of return statements were declared inside the same function "func1()".

Mark only one oval.

	Strongly disagree				Strongly agree
	1	2	3	4	5

67. D2: Use this space to add further comments about this issue:

D3: Redundant or unnecessary return declaration

Indicates an occurrence in which a return declaration that has no impact over the execution of the program is present inside a function.

68. D3: I believe this issue is of great concern:

```
'''D3: Redundant or unnecessary return declaration'''
def func1(var1):
    print(var1)
    return
```

In this generic case of D3, a return statement was declared inside the function "func1()" but since there was no need to return anything, it was unnecessary.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

69. D3: Use this space to add further comments about this issue:

D4: Function accessing variables not present in its scope

Indicates an occurrence in which a function accesses variables from outside its scope, without passing them as a parameter. Without using the global declaration for the variable, Python lets some data types be accessed in read-only format (e.g. int, float) or read/write (e.g. lists and dictionaries).

70. D4: I believe this issue is of great concern:

```
'''D4: Function accessing variables not present in its scope'''
var2 = (...)
def func1(var1):
    return var1 + var2
```

In this generic case of D4, the function "func1()" returns the sum of the variables "var1" and "var2", but only "var1" is present in its scope.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

71. D4: Use this space to add further comments about this issue:

Category E - Reasoning

In this section the Programming Issues that are related to student reasoning while trying to solve problems in coding are presented. There are 2 Programming Issues in total, cataloged from E1 to E2.

E1: Checking all possible combinations unnecessarily

Indicates an occurrence in which all possible combinations of boolean variables and/or conditional expressions are unnecessarily declared in order to check all cases.

72. E1: I believe this issue is of great concern:

```
'''E1: Checking all possible combinations unnecessarily'''
if var1 is False and var2 is False:
    func1()
if var1 is False and var2 is True:
    func1()
if var1 is True and var2 is False:
    func1()
if var1 is False and var2 is False:
    func2()
```

In this generic example of E1, the Truth Table is wholly declared unnecessarily in order to check all the possible combinations if "var1" and "var2" were "True" or "False".

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

73. E1: Use this space to add further comments about this issue:

E2: Redundant or unnecessary use of lists

Indicates an occurrence in which lists are declared and used to perform operations that does not necessarily need the use of this data structure.

74. E2: I believe this issue is of great concern:

```
1  '''E2: Redundant or unnecessary use of lists'''
2  while expr1:
3      var1 = input()
4      lst1.append(var1)
5      expr1 = (...)
6
7  for iter in lst1:
8      var2 += iter
```

In this generic example of E2, a while loop is declared in line 2 to read input data and store it in the list "lst1", then, a for loop is declared in line 7 to iterate over "lst1" and sum all of its numbers, storing it in the variable "var2". The operation in this for loop in line 7 could have been already done while reading the input in the first while loop, without the need of storing it in a list.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

75. E2: Use this space to add further comments about this issue:

Category F - Test cases	In this section the Programming Issues that are related to coding behavior that emerges from students knowing test cases which the code is subjected to in automatic correction are presented. There are 2 Programming Issues in total, cataloged from F1 to F2.

F1: Verification for non explicit conditions

Indicates an occurrence in which verifications and procedences for situations, which were not explicitly said they could happen, are declared in the code.

76. F1: I believe this issue is of great concern:

```
'''F1: Verification for non explicit conditions'''
if expr1 and expr2 and expr3:
    (...)
    #expr2 and expr3 were not said they could happen
```

In this generic example of F1, "expr2" and "expr3" are conditions that were not said in the exercise that they could happen in the test cases, but the student decided to verify them anyway.

Mark only one oval.

	Strongly disagree
1	◯
2	◯
3	◯
4	◯
5	◯
	Strongly agree

77. F1: Use this space to add further comments about this issue:

F2: Specified verification for instances of open test cases

Indicates an occurrence in which verifications and procedures for specific information present in the open test cases are declared in the code.

78. F2: I believe this issue is of great concern:

```
'''F2: Specified verification for instances of open test cases'''
#VAL1, VAL2 and VAL3 are explicit examples in the open test cases
if var1 == VAL1:
    (...)
elif var1 == VAL2:
    (...)
elif var1 == VAL3:
    (...)
else:
    (...)
```

In this generic example of F2, the variable "var1" is compared with values "VAL1", "VAL2" and "VAL3", which are values that are explicit in the open test cases (that is, the test cases that the student can see the input) that will run to verify the program.

Mark only one oval.

	Strongly disagree
1	◯
2	◯
3	◯
4	◯
5	◯
	Strongly agree

79. F2: Use this space to add further comments about this issue:

Category G - Code organization

In this section the Programming Issues that are related to code organization, such coding behavior or order of general declarations, are presented. There are 6 Programming Issues in total, cataloged from G1 to G6.

G1: Long line commentary

Indicates an occurrence in which long commentaries are written in the line format in Python, extending the file horizontally.

80. G1: I believe this issue is of great concern:

```
'''G1: Long line commentary'''
#this is a very long commentary that should have used the block format instead of the single type one
```

In this generic example of G1, a very long commentary was made using the Python line style commentary format instead of the block style one.

Mark only one oval.

	Strongly disagree					Strongly agree
	1	2	3	4	5	

81. G1: Use this space to add further comments about this issue:

G2: Exaggerated use of variables to assign expressions

Indicates an occurrence in which a high number of variables are declared to be assigned the result of simple expressions, being used further in the code.

82. G2: I believe this issue is of great concern:

```
1  '''G2: Exaggerated use of variables to assign expressions'''
2  var3 = var1 + var2
3  var4 = var1 - var2
4  var5 = func1(var4, var3)
5  var6 = var5 // var3
6  var7 = var1 and var5 or var3
7  (...)
8  if func2(var3):
9      (...)
10 elif func3(var7, var6)
11     (...)
```

In this generic example of G2, variables from "var3" to "var7" were assigned the results of simple operations with other variables. Further in the code, those variables are used in conditional checks (lines 8 and 10), forcing the person reading the code to go back and forth to remember what those variables represent.

Mark only one oval.

	Strongly disagree					Strongly agree
	1	2	3	4	5	

83. G2: Use this space to add further comments about this issue:

G3: Too many declarations in a single line of code

Indicates an occurrence in which declarations, distinct or not, are declared in the same line of code.

84. G3: I believe this issue is of great concern:

```
'''G3: Too many declarations in a single line of code'''
for iter in lst: func1(iter) if iter % 2 == 0 else func3(func1(iter),func2(iter))
```

In this generic example of G3, a for loop and a ternary if statement were declared in a single line of code. Although it makes the code more compact, sometimes it can make it difficult to read.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

85. G3: Use this space to add further comments about this issue:

G4: Functions/variables with non significant name

Indicates an occurrence in which the functions and/or variables are declared with arbitrary, non significant names that don't represent their purpose.

86. G4: I believe this issue is of great concern:

```
'''G4: Functions/variables with non significant name'''
a = func1(var1)
b = func1(var2)
x = func2(a, b)
y = func3(x)
z = func4(x, y)
```

In this generic example of G4, variables "a", "b", "x", "y" and "z" were assigned the results of functions. This type of issue generally happens with the declaration of subsequent alphabetical letters as variables.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

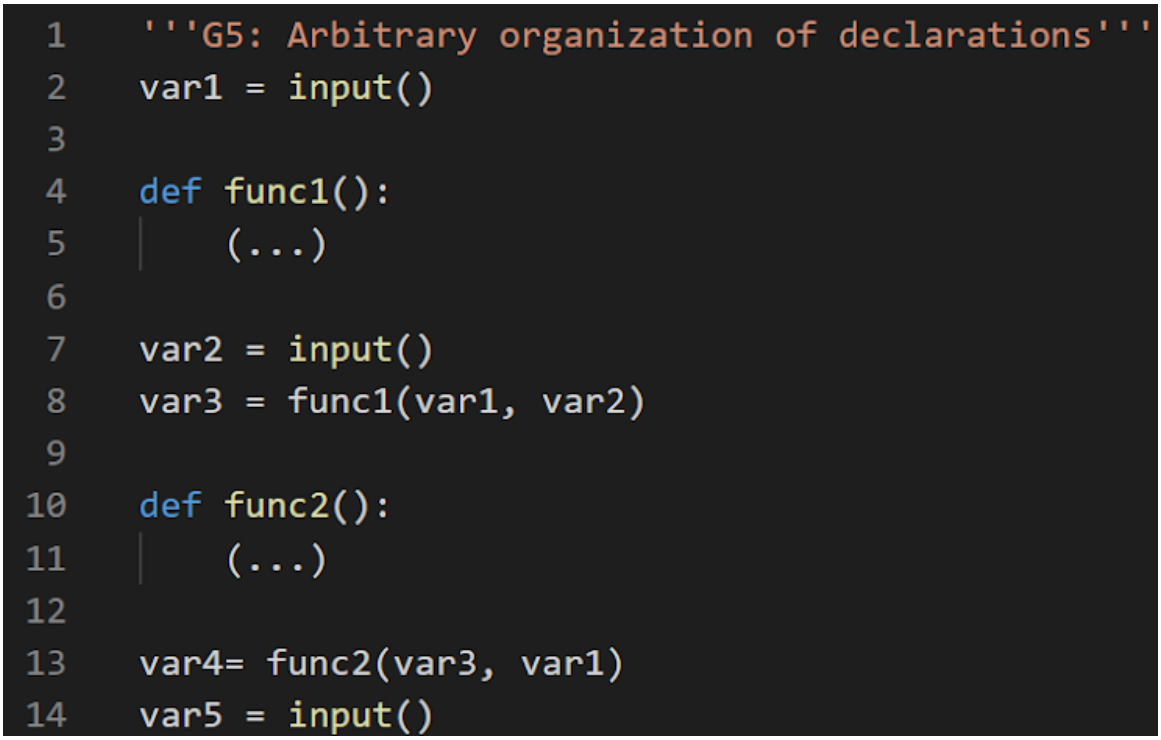
Strongly agree

87. G4: Use this space to add further comments about this issue:

G5: Arbitrary organization of declarations

Indicates an occurrence in which the declarations of variables and functions are made arbitrarily throughout the code.

88. G5: I believe this issue is of great concern:



```
1   '''G5: Arbitrary organization of declarations'''
2   var1 = input()
3
4   def func1():
5       (...)
6
7   var2 = input()
8   var3 = func1(var1, var2)
9
10  def func2():
11      (...)
12
13  var4= func2(var3, var1)
14  var5 = input()
```

In this generic example of G5, variable "var1" is declared and assigned a value in line 2, followed by a function "func1()" declaration in line 4. After that, more variables are declared, followed again by another function "func2()". The declaration of functions, input variables and the rest of the code is done arbitrarily.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

89. G5: Use this space to add further comments about this issue:

G6: Functions not documented in the Docstring format

Indicates an occurrence in which the declared functions are not properly documented, or documented at all using the Python Docstring format.

90. G6: I believe this issue is of great concern:

```
'''G6: Functions not documented in the Docstring format'''
def func1():
    (...) #some code that is not very clear

def func2():
    (...) #more code that is not very clear

def func3():
    #Some commentary about the function but not in Docstring format
    (...)
```

In this generic example of G6, the declared functions "func1()" and "func2()" have some code logic that is not very clear and lack documentation in any form. The function "func3()" has some commentaries about its logic but it's not documented in the Docstring format.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

91. G6: Use this space to add further comments about this issue:

Category H - Other

In this section the Programming Issues that are not fit to any other previous stated category are presented. There are 3 Programming Issues in total, cataloged from H1 to H3.

H1: Statement with no effect

Indicates an occurrence in which statements that doesn't affect the code execution are declared in the program, varying from literal numbers to not assigning the returning result of a function of method to a variable.

92. H1: I believe this issue is of great concern:

```
1  '''H1: Statement with no effect'''
2  var1 = func1()
3  4
4  round(var1)
```

In this generic example of H1, a number "4" was declared in line 3, having no effect in the code execution. Aside from that, the "round(var1)" declared in line 4 also has no effect, since the result is not assigned to any variable.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

93. H1: Use this space to add further comments about this issue:

H2: Redundant typecast

Indicates an occurrence in which a typecast is applied to the result of a function or expression, but there is no need to do so because the result of the function or expression is already converted to the desired type.

94. H2: I believe this issue is of great concern:

```
1   '''H2: Redundant typecast'''
2   var1 = str(input())
3   var2 = int(6 + 4)
```

In this generic example of H2, the result assigned to the variable "var1" is converted to a string, but the "input()" function already does that. The same goes for the result assigned to the variable "var2", since in Python, the sum of two integers returns an integer.

Mark only one oval.

Strongly disagree

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Strongly agree

95. H2: Use this space to add further comments about this issue:

H3: Unnecessary or redundant semicolon

Indicates an occurrence in which a semicolon is used unnecessarily, typically at the end of a statement or declaration in a line of code.

96. H3: I believe this issue is of great concern:

```
1   '''H3: Unnecessary or redundant semicolon'''
2   var3 = var2 - var1;
```

In this generic example of H3, a semicolon was typed at the end of the statement in line 2. Since it's a single declaration in Python, it was not needed.

Mark only one oval.

Strongly disagree

1

2

3

4

5

Strongly agree

97. H3: Use this space to add further comments about this issue:

Participation in a semi structured interview

As mentioned in the Informed Consent Form, this research will have a second phase composed of a semi structured interview. This interview will be composed by a small talk in order to get to know a bit about the volunteer, followed by a session asking if they have found any of the Programming Issues while teaching introductory programming courses and how they deal with them, and then followed by a session asking if they have any commentary or would like to inform of any new Programming Issue that was not listed in this research. The researcher will ask if the meeting can be recorded for future reference, but it's not mandatory.

If you agree to take part in this second phase, please answer the question below and the research staff will contact you at a later date via the email you informed.

98. Would you like to participate in the semi structured interview described above? *

Mark only one oval.

- ◯ Yes, I volunteer to participate in the semi structured interview
- ◯ No, I don't I volunteer to participate in the semi structured interview

Thank you for participating in this research!

When this research is concluded, we will send you the results via the email informed in this questionnaire.