A Questionnaire of Perceptual Learning Modules (PLM) for Introductory Programming (CS1) Courses in Python

Ricardo Caceffo *Guilherme Gama* *Jacques Wainer*
Islene Garcia *Rodolfo Azevedo*

Technical Report - IC-18-11 - Relatório Técnico
September - 2018 - Setembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

A Questionnaire of Perceptual Learning Modules (PLM) for Introductory Programming (CS1) Courses in Python

Ricardo Caceffo, Guilherme Gama, Jacques Wainer, Islene Garcia, Rodolfo Azevedo

Instituto de Computação Universidade Estadual de Campinas (UNICAMP), Caixa Postal 6176
13083-970 Campinas-SP, Brasil

caceffo@ic.unicamp.br; guilhermegama@gmail.com; wainer@ic.unicamp.br;
islene@ic.unicamp.br; rodolfo@ic.unicamp.br

Abstract. The educational environment usually emphasizes learning as a declarative and procedural process. However, there are other crucial components that can be considered, such as the Perceptual Learning (PL), related to the ways the information is perceived and extracted. A way to model the PL approach is through Perceptual Learning Modules (PLMs), a set of multiple-choice questions in which each available choice is mapped to a learning pattern. Currently, there are initiatives related to PLM practices in mathematics, flight learning, medicine and language educational areas. This Technical Report is a reference document, not peer-reviewed, that aims to document a set of 64 PLM multiple-choice questions designed to Introductory Programming (CS1) courses in Python language, therefore allowing our work to be replicated by other researchers. The actual use of the PLM questions and the analysis and discussion of its impact on a CS1 course will be addressed in further works.

Keywords: Perceptual Learning, PL, Perceptual Learning Model, PLM, CS1, Learning Pattern, Python, Programming

1. Introduction

Kellman *et al.* [1] explain that usually the educational environment emphasizes learning as a declarative and procedural process. However, the authors also point out that there are other crucial components to be considered, such as the Perceptual Learning (PL), related to the ways the information is perceived and extracted. The authors applied these ideas in the form of **Perceptual Learning Modules (PLMs)** – a set of multiple-choice questions in which each available choice is mapped to a learning pattern – in the mathematics learning.

Quoting Kellman *et al.* [1] "*unlike conventional practice in solving problems, learners in PLMs typically discriminate patterns, compare structures, make classifications, or map structure across representations".*

Besides the Kellman *et al.* [1] study in the mathematics field, there are successful initiatives related to PLM practices in the flight learning [3], medicine [4] and language [5], educational areas. Our motivation and also research hypothesis is that it is possible to design PLMs to Introductory Programming Courses (CS1) [2], also having a positive impact in this learning environment.

This Technical Report is a **reference document,** not peer-reviewed, that aims to document a set of 64 PLM multiple-choice questions designed to CS1 courses in Python language, **therefore allowing our research to be replicated by other researchers.** The actual use of the PLM questions and the analysis and discussion of its impact on a CS1 course will be addressed in further works.

This Technical Report is organized as follows: Section 3 describes how the PLM questions were designed; Section 4 presents all 64 PLM questions and; in Section 5 we make the acknowledgments. Appendix 1 presents the details related to the constraints used to organize the patterns mapped to the question's wrong choices.

2. Materials and Methods

We designed the PLM questions through an iterative process, used to identify the main topics and patterns related to CS1 content in the Python language. Initially, **3 topics** were defined: T1, related to arrays (lists); T2, related to loops; and T3 related to conditionals. We then identified **9 patterns** related to these topics, as described in Table 1:

Topic	**Pattern**	
	ID	**Description**
T1 **Arrays (Lists)**	*P1-T1*	Iterate over the array until the first element to satisfy a certain property is located.
	P2-T1	Append an element to the array.
	P3-T1	Copy the contents of an array to another array.
T2 **Loops**	*P1-T2*	Carry out a preset number of iterations.
	P2-T2	Carry out iterations while a certain property is true.
	P3-T2	Carry out iterations until a certain situation is identified.
T3 **Conditionals**	*P1-T3*	Check whether a number is in a certain interval
	P2-T3	Choose an alternative from a set of options

Table 1. Patterns related to the topics (T1, T2, and T3). Each pattern received a unique ID, mapping it to its related topic. For example, the second pattern on Topic T1 was identified as P2-T1.

We then determined the Python structure syntaxes that would be mapped to the questions: loops (*for in range*, *while* and *for each*); conditionals (*if)*; and arrays (lists) manipulation. The next step was to create the PLM questions, following these rules:

- If a pattern is associated with the loop syntax: to create 4 questions for each loop structure (*i.e.* 4 questions with *for in range,* 4 questions with *while* and 4 questions with *for each).*
- If a pattern is not associated with the loop syntax: to create 4 questions according to the mapped syntax.

Through this process, 64 PLM questions were designed, as described in Table 2. Each question received an ID with the format: *Question Number–Pattern Number–Topic Number*. For example, the question Q1-P2-T3 is the first question of the second pattern in Topic 3.

Topic	Pattern	Python Syntax	Question ID
T1 **Arrays (Lists)**	**P1** Iterate over the array until the first element to satisfy a certain property is located.	for in range	Q1-P1-T1
			Q2-P1-T1
			Q3-P1-T1
			Q4-P1-T1
		for each[1]	Q5-P1-T1
			Q6-P1-T1
			Q7-P1-T1
			Q8-P1-T1
		while	Q9-P1-T1
			Q10-P1-T1
			Q11-P1-T1
			Q12-P1-T1
	P2 Append an element to the array.	*Array manipulation*	Q1-P2-T1
			Q2-P2-T1
			Q3-P2-T1
			Q4-P2-T1
	P3 Copy the contents of an array to another array.	for in range	Q1-P3-T1
			Q2-P3-T1
			Q3-P3-T1
			Q4-P3-T1
		for each	Q5-P3-T1
			Q6-P3-T1

[1] We define `for each` as the `for` loop over a list of elements, in which **each** element is accessed in **each** iteration. For example:

```
my_list = [10, 20, 30]
for element in list:
    print(element)
```

			Q7-P3-T1
			Q8-P3-T1
		while	Q9-P3-T1
			Q10-P3-T1
			Q11-P3-T1
			Q12-P3-T1
T2 Loops	**P1** Carry out a preset number of iterations.	for in range	Q1-P1-T2
			Q2-P1-T2
			Q3-P1-T2
			Q4-P1-T2
		for each	Q5-P1-T2
			Q6-P1-T2
			Q7-P1-T2
			Q8-P1-T2
		while	Q9-P1-T2
			Q10-P1-T2
			Q11-P1-T2
			Q12-P1-T2
	P2 Carry out iterations while a certain property is true.	while	Q1-P2-T2
			Q2-P2-T2
			Q3-P2-T2
			Q4-P2-T2
			Q5-P2-T2
			Q6-P2-T2
			Q7-P2-T2
			Q8-P2-T2
	P3 Carry out iterations until	for in range	Q1-P3-T2
			Q2-P3-T2

	a certain situation is identified.		Q3-P3-T2
			Q4-P3-T2
			Q5-P3-T2
			Q6-P3-T2
			Q7-P3-T2
			Q8-P3-T2
T3 Conditionals	**P1** Check whether a number is in a certain interval	*if*	Q1-P1-T3
			Q2-P1-T3
			Q3-P1-T3
			Q4-P1-T3
	P2 Choose an alternative from a set of options	*if*	Q1-P2-T3
			Q2-P2-T3
			Q3-P2-T3
			Q4-P2-T3

Table 2. How each one of the 64 PLM questions was designed, accordingly to its topic, pattern and syntax in python language.

3. PLM Questions

The PLM questions were designed to match the template illustrated in Table 3:

Question (*Question_ID***):** *Question_Description*

Consider the code excerpt below:

```
Question_Code
```

What does the above code do?

a) *Right_Pattern*
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 3. PLM questions template

The template fields are:

- *Question_ID:* the question identifier, as described in Table 2. This is a control field, not supposed to be showed to students.
- *Question_Description (optional):* Describes, when appropriate, what the programming code does (e.g. "*Check whether the array contains only positive numbers*"). This is a control field, not supposed to be showed to students.
- *Question_Code:* The programming code created to match the question pattern.
- *Right_Pattern:* The description of the pattern mapped to the question, as described in Tables 1 and 2.
- *b) to e) other patterns:* The wrong choices (wrong patterns) available in the PLM question. These choices can be random or follow any logical organization (*e.g.* in the total each pattern would be addressed as a wrong choice the same number of times). **However,** some patterns have **constraints**, so they do not allow determined patterns to be mapped as wrong choices, as the students could consider these patterns as the right answer. This situation is addressed in **Appendix 1**.

This section is organized as follows: subsection 3.1 presents the PLM questions related to Topic T1 (lists); subsection 3.2 presents the questions related to Topic T2 (loops) and; subsection T3 the questions related to the Topic T3 (conditionals).

3.1 Topic T1: Arrays (Lists)

3.1.1) Pattern P1-T1: Iterate over the array until the first element to satisfy a certain property is located.

Question (Q1-P1-T1): Search for an element in the array

Consider the code excerpt below:

```
for i in range(n):
  if array[i] > 3:
       break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 4. Question Q1-P1-T1

Question (Q2-P1-T1): Check whether the array is ordered

Consider the code excerpt below:

```
is_ordered = True
for i in range(n - 1):
    if array[i] > array[i + 1]:
        is_ordered = False
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 5. Question Q2-P1-T1

Question (Q3-P1-T1): Check whether the array contains only positive numbers.

Consider the code excerpt below:

```
only_positive = True
for i in range(n):
    if array[i] <= 0:
        only_positive = False
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 6. Question Q3-P1-T1

Question (Q4-P1-T1): Find a specific element in the array.

Consider the code excerpt below:

```
found_element = False
for i in range(n):
    if array[i] == 10:
        found_element = True
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 7. Question Q4-P1-T1

Question (Q5-P1-T1): Find an element in the array.

Consider the code excerpt below:

```
for a in array:
  if a > 3:
       break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 8. Question Q5-P1-T1

Question (Q6-P1-T1): Check whether the array only contains 0s and 1s**.**

Consider the code excerpt below:

```
binary = True
for v in array:
    if v != 0 and v != 1:
        binary = False
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 9. Question Q6-P1-T1

Question (Q7-P1-T1): Check whether the array only contains positive numbers

Consider the code excerpt below:

```
is_positive = True
for a in array:
    if a <= 0:
        is_positive = False
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 10. Question Q7-P1-T1

Question (Q8-P1-T1): Check whether an array only contains odd numbers.

Consider the code excerpt below:

```
is_odd = True
for a in array:
    if a%2 == 0:
        is_odd = False
        break
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 11. Question Q8-P1-T1

Question (Q9-P1-T1): Find an element in an array

Consider the code excerpt below:

```
found = False
i = 0
while i < n and not found:
   if array[i] == 2:
       found = True
   i += 1
```

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 12. Question Q9-P1-T1

Question (Q10-P1-T1): Check whether an array is ordered

Consider the code excerpt below:

```
is_ordered = True
i = 0
while i < n - 1 and is_ordered:
    if array[i] > array[i + 1]:
       is_ordered = False
    i += 1
```

What does the above code do?

a) Iterate over the array until the first element to satisfy a certain property is located.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 13. Question Q10-P1-T1

Question (Q11-P1-T1): Check whether an array only contains positive numbers

Consider the code excerpt below:

```
is_positive = True
i = 0
while i < n and is_positive:
    if array[i] <= 0:
        is_positive = False
    i += 1
```

a) Iterate over the array until the first element to satisfy a certain property is located.

b) to e) other patterns following the constraints tables (see Appendix 1)

Table 14. Question Q11-P1-T1

Question (Q12-P1-T1): Look for the first element X in an array.

Consider the code excerpt below:

```
found_element_x = False
i = 0
while i < n  and not found_element_x:
    if array[i] == x:
        found_element_x = True
    i += 1
```

a) Iterate over the array until the first element to satisfy a certain property is located.

b) to e) other patterns following the constraints tables (see Appendix 1)

Table 15. Question Q12-P1-T1

3.1.2) Pattern P2-T1: Append an element to the array.

Question (Q1-P2-T1): Add the number 4 to the array.

Consider the code excerpt below:

```
my_list = [1,2,3]
my_list.append(4)
```

What does the above code do?

a) Append an element to the array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 16. Question Q1-P2-T1

Question (Q2-P2-T1): Append the string "Beijing" to the array.

Consider the code excerpt below:

```
world_capitals = ["Addis Abeba", "Canberra", "Ottawa"]
world_capitals = world_capitals + ["Beijing"]
```

What does the above code do?

a) Append an element to the array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 17. Question Q2-P2-T1

Question (Q3-P2-T1): Append the number 1.55 to the array.

Consider the code excerpt below:

```
my_list = [2.0, 5.1, 3.6, -7.76]
my_list.append(1.55)
```

What does the above code do?

a) Append an element to the array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 18. Question Q3-P2-T1

Question (Q4-P2-T1): Append an `int` object to the array.

Consider the code excerpt below:

```
my_list = [5, "Carla", 3.66, -8e5]
my_list = my_list + [10]
```

What does the above code do?

a) Append an element to the array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 19. Question Q4-P2-T1

3.1.3) Pattern P3-T1: Copy the contents of an array to another array.

Question (Q1-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
for i in range(len(my_list)):
    new_list.append(my_list[i])
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 20. Question Q1-P3-T1

Question (Q2-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
for i in range(len(my_list)):
    new_list = new_list + [my_list[i]]
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 21. Question Q2-P3-T1

Question (Q3-P3-T1):

Consider the code excerpt below:

```
array_b = list()
for i in range(len(array_a)):
    array_a.append(array_a[i])
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 22. Question Q3-P3-T1

Question (Q4-P3-T1):

Consider the code excerpt below:

```
B = []
for i in range(len(A)):
    B = B + [A[i]]
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 23. Question Q4-P3-T1

Question (Q5-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
for value in my_list:
    new_list.append(value)
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 24. Question Q5-P3-T1

Question (Q6-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
for value in my_list:
    new_list = new_list + [value]
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 25. Question Q6-P3-T1

Question (Q7-P3-T1):

Consider the code excerpt below:

```
array_b = [0,1,2,3]
for v in array_a:
    array_b.append(v)
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 26. Question Q7-P3-T1

Question (Q8-P3-T1):

Consider the code excerpt below:

```
array_b = []
for v in array_a:
    array_b = array_b + [v]
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 27. Question Q8-P3-T1

Question (Q9-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
i = 0
while i < len(my_list):
    new_list.append(my_list[i])
    i += 1
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 28. Question Q9-P3-T1

Question (Q10-P3-T1):

Consider the code excerpt below:

```
new_list  =  []
i = 0
while i < len(my_list):
    new_list = new_list + [my_list[i]]
    i += 1
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 29. Question Q10-P3-T1

Question (Q11-P3-T1):

Consider the code excerpt below:

```
B = list()
i = 0
while i < len(A):
    B.append(A[i])
    i += 1
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 30. Question Q11-P3-T1

Question (Q12-P3-T1):

Consider the code excerpt below:

```
array_b = []
i = 0
while i < len(array_a):
    array_b = array_b + [array_a[i]]
    i += 1
```

What does the above code do?

a) Copy the contents of an array to another array.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 31. Question Q12-P3-T1

3.2) Topic T2: Loops

3.2.1) Pattern P1-T2: Carry out a preset number of iterations.

Question (Q1-P1-T2):

Consider the code excerpt below:

```
for counter in range(11):
    sum_value += counter
```

What does the above code do?

a) Carry out a preset number of iterations.

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 32. Question Q1-P1-T2

Question (Q2-P1-T2):

Consider the code excerpt below:

```
for i in range(lo, hi):
    print(A[i])
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 33. Question Q2-P1-T2

Question (Q3-P1-T2):

Consider the code excerpt below:

```
n = int(input())
for k in range(n):
    array.append(input())
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 34. Question Q3-P1-T2

Question (Q4-P1-T2):

Consider the code excerpt below:

```
x = 1
for i in range(1,n):
    x *= i
```

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 35. Question Q4-P1-T2

Question (Q5-P1-T2):

Consider the code excerpt below:

```
for value in array_a:
    sum_value += value
```

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 36. Question Q5-P1-T2

Question (Q6-P1-T2):

Consider the code excerpt below:

```
for v in array_a[lo:hi]:
    print(v)
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 37. Question Q6-P1-T2

Question (Q7-P1-T2):

Consider the code excerpt below:

```
for x in array:
    print(x, x**2+5*x+1)
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 38. Question Q7-P1-T2

Question (Q8-P1-T2):

Consider the code excerpt below:

```
for line in matrix:
    for element in line:
        print(element)
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 39. Question Q8-P1-T2

Question (Q9-P1-T2):

Consider the code excerpt below:

```
counter = 0
while counter <= 10:
    sum_value += counter
    counter += 1
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 40. Question Q9-P1-T2

Question (Q10-P1-T2):

Consider the code excerpt below:

```
i = lo
while i < hi:
    print(array_a[i])
    i += 1
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 41. Question Q10-P1-T2

Question (Q11-P1-T2):

Consider the code excerpt below:

```
n = int(input())
i = 0
while i < n:
    array.append(input())
    i += 1
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 42. Question Q11-P1-T2

Question (Q12-P1-T2):

Consider the code excerpt below:

```
x = 1
i = 1
while i < n:
     x *= i
     i += 1
```

What does the above code do?

a) Carry out a preset number of iterations.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 43. Question Q12-P1-T2

3.2.2) Pattern P2-T2: Carry out iterations while a certain property is true.

Question (Q1-P2-T2):

Consider the code excerpt below:

```
count = 1
while count % 10 != 0:
    print(count)
    count += 1
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 44. Question Q1-P2-T2

Question (Q2-P2-T2):

Consider the code excerpt below:

```
d = 1
powers = []
while d < limit:
    powers.append(d)
    d *= 2
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 45. Question Q2-P2-T2

Question (Q3-P2-T2):

Consider the code excerpt below:

```
names = ["Anna", "Baariz", "Consuelo", "Daisuke"]
while len(names) > 0:
    print(names[0])
    names = names[1:]
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 46. Question Q3-P2-T2

Question (Q4-P2-T2):

Consider the code excerpt below:

```
data = []
a = 0
while a != -1:
    print("Type a non-negative integer or -1 to quit.")
    a = int(input())
    data.append(a)
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 47. Question Q4-P2-T2

Question (Q5-P2-T2):

Consider the code excerpt below:

```
while n % 2 == 0:
    n = n // 2
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 48. Question Q5-P2-T2

Question (Q6-P2-T2):

Consider the code excerpt below:

```
a = []
while len(a) <= 10:
    a.append(input())
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 49. Question Q6-P2-T2

Question (Q7-P2-T2):

Consider the code excerpt below:

```
i = 0
while hitpoints > 0 and i < len(attacks):
    hitpoints -= attacks[i]
    i += 1
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 50. Question Q7-P2-T2

Question (Q8-P2-T2):

Consider the code excerpt below:

```
while b != 0:
    t = b
    b = a % b
    a = t
```

What does the above code do?

a) Carry out iterations while a certain property is true.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 51. Question Q8-P2-T2

3.2.3) Pattern P3-T2: Carry out iterations until a certain situation is identified.

Question (Q1-P3-T2): Prime Numbers

Consider the code excerpt below:

```
is_prime = True
for i in range(2,x):
  if x % i == 0:
       is_prime = False
       break
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 52. Question Q1-P3-T2

Question (Q2-P3-T2):

Consider the code excerpt below:

```
for i in range(len(array_a)):
    a = input()
    if ' ' in a:
        print("Error: multiple words typed")
        break
    array_a[i] = a
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 53. Question Q2-P3-T2

Question (Q3-P3-T2):

Consider the code excerpt below:

```
for i in range(len(array_a)):
    if array_a[i] == 0:
        break
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 54. Question Q3-P3-T2

Question (Q4-P3-T2):
Consider the code excerpt below:

`for i in range(0, len(points), 2):`
`    if points[i] > 0 and points[i + 1] < 0:`
`        break`

What does the above code do?
a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 55. Question Q4-P3-T2

Question (Q5-P3-T2):
Consider the code excerpt below:

`for i in range(len(bets)):`
`    if total_wagered + bets[i] > balance:`
`        break`
`    total_wagered += bets[i]`

What does the above code do?
a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 56. Question Q5-P3-T2

Question (Q6-P3-T2):

Consider the code excerpt below:

```
for x in range(-10,11):
    if x**2 - 6*x + 5 == 0:
        integer_root = x
        break
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 57. Question Q6-P3-T2

Question (Q7-P3-T2):

Consider the code excerpt below:

```
sum_value = 0.0
for i in range(len(weights)):
    if sum_value + weights[i] > capacity:
        break
    else:
        sum_value += weights[i]
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 58. Question Q7-P3-T2

Question (Q8-P3-T2):

Consider the code excerpt below:

```
for i in range(len(names)):
    if student_id[i] == "null":
        break
```

What does the above code do?

a) Carry out iterations until a certain situation is identified.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 59. Question Q8-P3-T2

3.3) Topic T3: Conditionals

3.3.1) Pattern P1-T3: Check whether a number is in a certain interval

Question (Q1-P1-T3):

Consider the code excerpt below:

```
n = 5
if 0 < n and n < 10:
  print("n is between 0 and 10")
```

What does the above code do?

a) Check whether a number is in a certain interval
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 60. Question Q1-P1-T2

Question (Q2-P1-T3):

Consider the code excerpt below:

```
print("Please enter a positive integer.")
i = int(input())
if i <= 0:
  print("Invalid input.")
```

What does the above code do?

a) Check whether a number is in a certain interval
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 61. Question Q2-P1-T3

Question (Q3-P1-T3):

Consider the code excerpt below:

```
a = 5
if a > 10:
  a = a % 10
```

What does the above code do?

a) Check whether a number is in a certain interval
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 62. Question Q3-P1-T3

Question (Q4-P1-T3):
Consider the code excerpt below:

`d = -4`
`if d <= -5 or d >= 5:`
`  print("d is not between -5 and 5.")`

What does the above code do?
a) Check whether a number is in a certain interval
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 63. Question Q4-P1-T3

3.3.2) Pattern P2-T3: Choose an alternative from a set of options.

Question (Q1-P2-T3):
Consider the code excerpt below:

`if choice == 1:`
`    result = 10`
`elif choice == 2:`
`    result = 20`
`else:`
`    result = 0`

What does the above code do?
a) Choose an alternative from a set of options.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 64. Question Q1-P2-T3

Question (Q2-P2-T3):

Consider the code excerpt below:

```
if len(array_a) == 0:
  array_a.append(0)
else:
  array_a[0] = 3
```

What does the above code do?

a) Choose an alternative from a set of options.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 65. Question Q2-P2-T3

Question (Q3-P2-T3):

Consider the code excerpt below:

```
if month == 2:
    if leap_year:
        days = 29
    else:
        days = 28
elif month in [4, 6, 9, 11]:
    days = 30
else:
    days = 31
```

What does the above code do?

a) Choose an alternative from a set of options.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 66. Question Q3-P2-T3

Question (Q4-P2-T3):

Consider the code excerpt below:

```
if 0 < x and x < pi/2:
    quadrant = 1
elif pi/2 < x and x < pi:
    quadrant = 2
elif pi < x and x < 3*pi/2:
    quadrant = 3
elif 3*pi/2 < x and x < 2*pi:
    quadrant = 4
else:
    quadrant = 0
    print("x coincides with an axis.")
```

What does the above code do?

a) Choose an alternative from a set of options.
b) to e) other patterns following the constraints tables (see Appendix 1)

Table 67. Question Q4-P2-T3

4. Acknowledgements

This research is supported by grant #2014/07502, São Paulo Research Foundation (FAPESP). The opinions, hypotheses, conclusions, or recommendations expressed in this material are the responsibility of the author(s) and do not necessarily reflect the views of FAPESP. Additional support was provided by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES), the National Council of Technological and Scientific Development (CNPq) and the University of Campinas (Unicamp).

5. References

[1] Kellman, Philip J.; Massey, Christine M. & Son, Ji Y. (2010). Perceptual Learning Modules in Mathematics: Enhancing Students Pattern Recognition, Structure Extraction, and Fluency. In Topics in Cognitive Science 2 (2):285-305. Online ISSN:1756-8765

[2] Mattew Hertz (2010). What do "CS1" and "CS2" mean?: investigating differences in the early courses. *In Proceedings of the 41st ACM technical symposium on Computer science education. pp199-203,* New York, NY, USA, ISBN: 978-1-4503-0006-3 DOI: 10.1145/1734263.1734335

[3] Kellman, P. J., & Kaiser, M. K. (1994). Perceptual learning modules in flight training. Proceedings of the 38thAnnual Meeting of the Human Factors and Ergonomics Society, 2, 1183–1187.

[4] Guerlain, S., La Follette, M., Mersch, T. C., Mitchell, B. A., Poole, G. R., Calland, J. F., Jianhong, Lv, & Chekan, E. G. (2004). Improving surgical pattern recognition through repetitive viewing of video clips. IEEE Transactions on Systems, Man, and Cybernetic – Part A: Systems and Humans, 34(6), 699–707.

[5] Tallal, P., Merzenich, M., Miller, S., & Jenkins, W. (1998). Language learning impairment: Integrating research and remediation. Scandinavian Journal of Psychology, 39(3), 197–199.

6. Appendix 1 – Constraints

During the research, it was identified that some patterns could – in some occasions – be considered true for determined programming codes related to other patterns. For example, the following programming code was designed to match pattern P1-T1: *"Iterate over the array until the first element to satisfy a certain property is located."*

```
for i in range(n):
  if array[i] > 3:
        break
```

However, it also does satisfy the pattern P3-T2: *"Carry out iterations until a certain situation is identified."*. Therefore, to avoid misleading the students, we defined a constraints table (see Table 68), that defines, for each pattern, which patterns **should not** be used as wrong choices.

		Patterns							
		P1-T1	**P2-T1**	**P3-T1**	**P1-T2**	**P2-T2**	**P3-T2**	**P1-T3**	**P2-T3**
Patterns	**P1-T1**	▨				C5	C1		
	P2-T1		▨		C7	C8			C10
	P3-T1			▨					
	P1-T2				▨	C4	C3	C6	
	P2-T2					▨	C2	C9	
	P3-T2						▨		
	P1-T3							▨	C11
	P2-T3								▨

Table 68. Constraints table, defining for each pattern which patterns must not be used as wrong choices in questions related to that pattern. For example, questions related to pattern P1-T1 should avoid patterns P2-T2 and P3-T2 as wrong choices. The diagonal grey cells indicate that a pattern can't be used as the wrong choice in a question mapped to the same pattern (*e.g.* a P1-T1 question can't have a P1-T1 pattern as the wrong choice). The grey cells left to the diagonal were included for aesthetic purposes, avoiding the display of duplicated information in the table. Constraints were labeled from C1 to C11.

Therefore, the only restriction to define the PLM questions wrong choice is the respect of the constraints defined in Table 68. In our study, however, we opted to try to create an arrangement in which the number of times each pattern is referred to as a wrong choice is as homogeneous as possible. This arrangement is described in Table 69.

Questions	**Patterns (wrong choices)**							
	P1-T1	**P2-T1**	**P3-T1**	**P1-T2**	**P2-T2**	**P3-T2**	**P1-T3**	**P2-T3**
Total →	*25*	*47*	*41*	*19*	*6*	*20*	*49*	*49*
Q1-P1-T1	**C**	1	0	1	0	**X**	1	1
Q2-P1-T1	**C**	0	1	1	**X**	**X**	1	1
Q3-P1-T1	**C**	1	1	0	**X**	**X**	1	1
Q4-P1-T1	**C**	0	1	1	**X**	**X**	1	1
Q5-P1-T1	**C**	1	0	1	0	**X**	1	1
Q6-P1-T1	**C**	1	1	1	**X**	**X**	0	1
Q7-P1-T1	**C**	0	1	1	**X**	**X**	1	1
Q8-P1-T1	**C**	1	0	1	**X**	**X**	1	1
Q9-P1-T1	**C**	1	1	1	**X**	**X**	1	0
Q10-P1-T1	**C**	1	1	1	**X**	**X**	1	0
Q11-P1-T1	**C**	0	1	1	**X**	**X**	1	1
Q12-P1-T1	**C**	1	1	1	**X**	**X**	1	0
Q1-P2-T1	**X**	**C**	1	**X**	**X**	1	1	1
Q2-P2-T1	**X**	**C**	1	**X**	**X**	1	1	1
Q3-P2-T1	**X**	**C**	1	**X**	**X**	1	1	1
Q4-P2-T1	**X**	**C**	1	**X**	**X**	1	1	1
Q1-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q2-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q3-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q4-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q5-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q6-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1
Q7-P3-T1	**X**	1	**C**	**X**	**X**	1	1	1

Q8-P3-T1	X	1	C	X	X	1	1	1
Q9-P3-T1	X	1	C	X	X	1	1	1
Q10-P3-T1	X	1	C	X	X	1	1	1
Q11-P3-T1	X	1	C	X	X	1	1	1
Q12-P3-T1	X	1	C	X	X	1	1	1
Q1-P1-T2	1	1	1	C	X	X	0	1
Q2-P1-T2	1	1	1	C	X	X	0	1
Q3-P1-T2	1	1	1	C	X	X	0	1
Q4-P1-T2	1	1	1	C	X	X	1	0
Q5-P1-T2	1	1	1	C	X	X	1	0
Q6-P1-T2	1	1	0	C	X	X	1	1
Q7-P1-T2	1	1	1	C	X	X	1	0
Q8-P1-T2	1	0	1	C	X	X	1	1
Q9-P1-T2	1	1	1	C	X	X	X	1
Q10-P1-T2	1	1	1	C	X	X	X	1
Q11-P1-T2	1	X	1	C	X	X	1	1
Q12-P1-T2	1	1	1	C	X	X	1	0
Q1-P2-T2	1	0	1	0	C	X	1	1
Q2-P2-T2	1	X	1	1	C	X	1	0
Q3-P2-T2	1	1	0	1	C	X	0	1
Q4-P2-T2	1	0	0	1	C	X	1	1
Q5-P2-T2	1	1	0	0	C	X	1	1
Q6-P2-T2	1	X	1	X	C	X	1	1
Q7-P2-T2	1	1	1	X	C	X	X	1
Q8-P2-T2	1	1	0	1	C	X	1	0
Q1-P3-T2	X	1	1	X	X	C	1	1
Q2-P3-T2	X	1	1	X	X	C	1	1
Q3-P3-T2	X	1	1	X	X	C	1	1
Q4-P3-T2	X	1	1	X	X	C	1	1
Q5-P3-T2	X	1	1	X	X	C	1	1
Q6-P3-T2	X	1	1	X	X	C	1	1

Q7-P3-T2	X	1	1	X	X	C	1	1
Q8-P3-T2	X	1	1	X	X	C	1	1
Q1-P1-T3	0	1	1	0	0	1	C	1
Q2-P1-T3	1	0	0	1	1	0	C	1
Q3-P1-T3	1	1	1	0	1	0	C	0
Q4-P1-T3	0	1	1	1	1	0	C	0
Q1-P2-T3	1	0	1	1	1	0	0	C
Q2-P2-T3	0	X	1	0	1	1	1	C
Q3-P2-T3	1	1	0	1	0	1	0	C
Q4-P2-T3	1	1	0	0	1	1	X	C

Table 69. Wrong choices organization in the 64 PLM questions. Captions: letter **C** stands for **C**orrect; **X** stands for a constraint, as defined in Table 68; **1** stands for a pattern used as the wrong choice and; **0** stands for a pattern not used as wrong choice. For example, in the Q1-P1-T1, the correct answer is the pattern P1-T1 (indicated with a **C**); the pattern P3-T2 is a constraint for that question (indicated with an **X**), therefore this pattern can't be used as wrong choice in this question; the patterns P2-T1, P1-T2, P1-T3 and P2-T3, marked with **1**, were mapped to wrong choices in that question; and the patterns P3-T1 and P2-T2 could be, but were not, mapped to wrong choices.